Proceedings of the 24th Python in Science Conference

Proceedings of the 24th Python in Science Conference

Edited by Amey Ambade, Ana Comesana, Charles Lindsey, Franklin Koch, Hongsup Shin, Rowan Cockett, Sanhita Joshi, and Sean Freeman

SciPy 2025
Tacoma, Washington
July 7 - July 13, 2025

Copyright © 2025. The articles in the Proceedings of the Python in Science Conference are copyrighted and owned by their original authors.

This is an open-access publication and is distributed under the terms of the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.

For more information, please see: https://creativecommons.org/licenses/by/4.0/

ISSN:2575-9752
https://doi.org/10.25080/hptk5424

Organization

Conference Co-Chairs

- Julie Krugler Hollek, *Mozilla*
- Madicken Munk, *Oregon State University*

Program

- Naty Clement, *NVIDIA*
- Matthew Feickert, *University of Wisconsin-Madison*
- Gil Forsythe, *NVIDIA*
- Ariana Mendible, *Seattle University*

Tutorial

- Paul Anzel, *Rev*
- Matt Craig, *Minnesota State University Moorhead*
- Allen Harvey, *Applied Research Associates, Inc*
- Tetsuo Koyama, *PyVista community*
- Inessa Pawson (Teen Track), *OpenTeams*

Hybrid

- Rebecca BurWei, *Northwestern Medicine*
- David Nicholson, *BAE FASTLabs*
- Neelima Pulagam, *Ford Motor Company*
- Shubham Sharma, *GalaxyEye Space*

Proceedings

- Amey Ambade, *SLB*
- Rowan Cockett, *Curvenote*
- Ana Comesana, *Lawrence Berkeley National Laboratory*
- Sean Freeman, *University of Alabama*
- Sanhita Joshi, *Deloitte*
- Franklin Koch, *Curvenote*
- Charles Lindsey, *Aptos*
- Hongsup Shin, *Arm*

Community Outreach

- Juanita Gomez, *University of California, Santa Cruz*
- James Lamb, *NVIDIA*
- Cynthia Chen, *Stowers Institute for Medical Research*
- Juan Cabanela, *Minnesota State UniversityMoorhead*

Birds-of-a-Feather

- Michael Akerman, *Novonesis*
- Mike Droettboom, *Microsoft*

Sprints

- Hannah Aizenman, *Matplotlib*
- Nathan Goldbaum, *Quantsight*
- Brigitta Sipőcz, *Caltech/IPAC*

Activities

- Matt Davis, *Populus*
- Ed Rogers, *Majesco*
- Oleksandr Yardas, *University of Illinois Urbana-Champaign*

Financial Aid

- Eric Ma, *Moderna*
- Nadia Tahiri, *University of Sherbrooke*

Sponsors/Financial/Logistics

- Jim Weiss, *NumFOCUS*

Proceedings Reviewers

- Tolulade Ademisoye
- Nitin Appiah
- Georgios Artavanis
- Himaghna Bhattacharjee
- Matthew Feickert
- Bhaskar Goyal
- Allen S. Harvey Jr.
- Mike He
- David Heagy
- Jennifer Iloekwe
- Aditi Juneja
- Madhav Kashyap
- Sanjay Belaturu Krishnegowda
- James Lamb
- Matthew Lesko
- Jiahang Li
- Smit Lunagariya
- Mihai Maruseac
- Kurt McKee
- Manoj Krishna Mohan
- Steven Moss
- Chong Shen Ng
- Andrei Paleyes
- Yan Ru Pei
- Pranoy Ray
- Kyohei Sahara
- Will Sanger
- Praveen Gupta Sanka
- Pushkar Sathe
- Henry Schreiner
- Chirag Shah
- Neel Shah
- Anil Sharma
- Yash Sharma
- Varun Shivashankar

- Anmol Singhal
- Hansila Sudasinghe
- Nadia Tahiri
- Naomi Tang
- Dinesh Rathinasamy Thangavel
- Jennifer Yoon

Scholarship Recipients

- Fernando Cervantes-Sánchez, *The Jackson Laboratory*
- Sinclair Combs, *Colorado School of Mines*
- Cainã Max Couto da Silva, *UW-Madison*
- C.A.M. Gerlach, *Python and Spyder core team*
- Juanita Gomez, *University of California, Santa Cruz*
- Lilinoe Harbottle, *Independent*
- Arnaud Kayonga, *MeshPower/Mbaza NLP*
- Anastasiia Kostiv, *DataRacerEdu*
- Tetsuo Koyama, *Akari Inc.*
- Neha Gupta, *Carnegie Mellon University*
- Stephanie Rideout, *Independent*
- Sanjiban Sengupta, *CERN, The University of Manchester*
- Arkadiusz Trawinski, *ING*

Posters and Slides

Keynote Presentations

What we maintain, we defend

Scientific Python is not only at the heart of discovery and advancement, but also infrastructure. This talk will provide a perspective on how open-source Python tools that are already powering real-world impact across the sciences are also supportive of public institutions and critical public data infrastructure. Drawing on her previous experience leading policy efforts in the Department of Energy as well as her experience in open-source scientific computing, Katy will highlight the indispensable role of transparency, reproducibility, and community in high-stakes domains. This talk invites the SciPy community to recognize its unique strengths and to amplify their impact by contributing to the public good through technically excellent, civic-minded development.

Kathryn Huff

https://www.youtube.com/watch?v=pgv22OBqv9s

The Myth of Artificial: Spotlighting Community Intelligence for Responsible Science

The widespread fascination with AI often fuels a “myth of the artificial”, the belief that scientific and technological progress stems solely from algorithms and large tech breakthroughs. This talk challenges that notion, arguing that truly responsible and impactful science is fundamentally built upon and sustained by the resilient, collective intelligence of the scientific and research community.

Malvika Sharan

https://www.youtube.com/watch?v=pr8DCxivxDs

My Dinner with Numeric, Numpy, and Scipy: A Retrospective from 2001 to 2025 with Comments and Anecdotes

This keynote will trace the personal journey of NumPy’s development and the evolution of the SciPy community from 2001 to the present. Drawing on over two decades of involvement, I’ll reflect on how a small group of enthusiastic contributors grew into a vibrant, global ecosystem that now forms the foundation of scientific computing in Python. Through stories, milestones, and community moments, we’ll explore the challenges, breakthroughs, and collaborative spirit that shaped both NumPy and the SciPy conventions over the years.

Charles R. Harris

https://www.youtube.com/watch?v=xapt7HOB-P4

Python at the Speed of Light: Accelerating Science with CUDA Python

NVIDIA’s CUDA platform has long been the backbone of high-performance GPU computing, but its power has historically been gated behind C and C++ expertise. With the recent introduction of native Python support, CUDA is more accessible to the programming language you know and love, ushering in a new era for scientific computing, data science, and AI development.p

Christopher Lamb

https://www.youtube.com/watch?v=8n_zfZm1tQ0

Rubin Observatory: What will you discover when you’re always watching

After two decades of planning, Rubin Observatory is finally observing the sky. Built to image the entire southern hemisphere every few nights with a 3.2-gigapixel camera, Rubin will produce a

time-lapse of the Universe, revealing moving asteroids, pulsing stars, supernovae, and rare transients that you only catch if you're always watching. In this talk, I'll share the "first look" images from Rubin Observatory as well as what it took to get here: from scalable algorithms to infrastructure that moves data from a mountaintop in Chile to scientists around the world in seconds. I'll reflect on what we learned building the data management system in Python over the years, including stories of choices that impacted scalability, interfaces, and maintainability. Rubin Observatory is here. And it's for you.

Yusra AlSayyad

https://www.youtube.com/watch?v=bjuL0Y2Ulx0

Accepted Talks

An Active Learning Plugin in napari to Fine-Tune Models for Large-scale Bioimage Analysis

The "napari-activelearning" plugin provides a framework to fine tune deep learning models for large-scale bioimage analysis, such as digital pathology Whole Slide Images (WSI). This plugin was developed with the motivation of easing the integration of deep learning tools into bioimage analysis workflows. This plugin implements the concept of Active Learning for reducing the time spent on labeling samples when fine tuning models. Because this plugin is integrated into Napari and leverages the use of next generation file formats (Zarr), it is suitable for fine tuning deep learning models on large-scale images with little image preparation.

Fernando Cervantes

https://doi.org/10.5281/zenodo.17297859

Unlocking the Missing 78%: Inclusive Communities for the Future of Scientific Python

Scientific Python teams are leaving value on the table—the Missing78%—and it shows up in slower product cycles and brittle processes. This talk introduces the VIM framework (Visibility, Impact, Mechanisms) as a lightweight operating system for product development: define visible contribution ladders, focus on measurable impact, and install mechanisms that make responsible AI practices repeatable. Attendees leave with a one-page checklist, repo templates, and a 30-day plan to ship scientifically grounded, responsible-AI improvements—without extra headcount.

Noor Aftab

https://doi.org/10.5281/zenodo.17297863

The-Silmaril: Practice #ontology engineering with Python (and other languages).

Ontologies provide a powerful way to structure knowledge, enable reasoning, and support more meaningful queries compared to traditional data models. Recently, interest in ontologies has resurged, driven by advancements in language models, reasoning capabilities, and the growing adoption of platforms like Palantir Foundry. Here we explore ontology development across multiple domains using a variety of Python-based tools such as rdflib, Owlready2, PySpark, Pandas, and SciPy. We will learn how ontologies facilitate semantic reasoning, improve data interoperability, and enhance query capabilities. Additionally, we will build a rudimentary reasoning engine to better understand inference mechanisms. The presentation emphasizes practical applications and comparisons with conventional data representations, making it ideal for researchers, data engineers, and developers interested in knowledge representation and reasoning.

SHAURYA AGARWAL

https://doi.org/10.5281/zenodo.17297865

Cubed: Scalable array processing with bounded-memory in Python

Cubed is a framework for distributed processing of large arrays without a cluster. Designed to respect memory constraints at all times, Cubed can express any NumPy-like array operation as a series of embarrassingly-parallel, bounded-memory steps. By using Zarr as persistent storage between steps, Cubed can run in a serverless fashion on both a local machine and on a range of Cloud platforms. After explaining Cubed's model, we will show how Cubed has been integrated with Xarray and demonstrate its performance on various large array geoscience workloads.

Tom Nicholas, Tom White

https://doi.org/10.5281/zenodo.17297867

Scaling NumPy for Large-Scale Science: The cuPyNumeric Approach

Many scientists rely on NumPy for its simplicity and strong CPU performance, but scaling beyond a single node is challenging. The researchers at SLAC need to process massive datasets under tight beam time constraints, often needing to modify code on the fly. This is where cuPyNumeric comes in—a drop-in replacement for NumPy that distributes work across CPUs and GPUs. With its familiar NumPy interface, cuPyNumeric makes it easy to scale computations without rewriting code, helping scientists focus on their research instead of debugging. It's a great example of how the SciPy ecosystem enables cutting-edge science.

Irina Demeshko, Quynh L. Nguyen

https://doi.org/10.5281/zenodo.17297869

Python is all you need: an overview of the composable, Python-native data stack

For the past decade, tools like dbt have formed a cornerstone of the modern data stack, and Python-first alternatives couldn't compete with the scale and performance of modern SQL—until now. New integrations with Ibis, the portable Python dataframe library, enable building and orchestrating scalable data engineering pipelines using existing open-source libraries like Kedro, Pandera, and more.

Deepyaman Datta

https://doi.org/10.5281/zenodo.17297871

Processing Cloud-optimized data in Python with Serverless Functions (Lithops, Dataplug)

Cloud-optimized (CO) data formats are designed to efficiently store and access data directly from cloud storage without needing to download the entire dataset. These formats enable faster data retrieval, scalability, and cost-effectiveness by allowing users to fetch only the necessary subsets of data. They also allow for efficient parallel data processing using on-the-fly partitioning, which can considerably accelerate data management operations. In this sense, cloud-optimized data is a nice fit for data-parallel jobs using serverless. FaaS provides a data-driven scalable and cost-efficient experience, with practically no management burden. Each serverless function will read and process a small portion of the cloud-optimized dataset, being read in parallel directly from object storage, significantly increasing the speedup. In this talk, you will learn how to process cloud-optimized data formats in Python using the Lithops toolkit. Lithops is a serverless data processing toolkit that is specially designed to process data from Cloud Object Storage using Serverless functions. We will also demonstrate the Dataplug library that enables Cloud Optimized data management of scientific settings such as genomics, metabolomics, or geospatial data. We will show different data

processing pipelines in the Cloud that demonstrate the benefits of cloud-optimized data management.

Enrique Molina-Giménez, Pedro García-López

https://doi.org/10.5281/zenodo.17297875

GBNet: XGBoost and LightGBM PyTorch Modules

GBNet is a Python package that integrates XGBoost and LightGBM with PyTorch. By leveraging PyTorch auto-differentiation, GBNet enables novel architectures for GBMs that were previously exclusive to pure Neural Networks. The result is a greatly expanded set of applications for GBMs and an improved ability to interpret expressive architectures due to the use of GBMs.

Michael Horrell

https://doi.org/10.5281/zenodo.17297855

Accelerated DataFrames for all: Bringing GPU acceleration to pandas and Polars

In Python, data analytics users often prioritize convenience, flexibility, and familiarity over pure performance. The cuDF DataFrame library provides a pandas-like experience with from 10x up to 50x performance improvements, but subtle differences prevent it from being a true drop-in replacement for many users. This talk will showcase the evolution of this library to provide zero-code change experiences, first for pandas users and now for Polars. We will provide examples of this usage and a high level overview of how users can make use of these today. We will then delve into the details of how GPU acceleration is implemented differently in pandas and Polars, along with a deep dive into some of the different technical challenges encountered for each. This talk will have something for both data practitioners and library developers.

Vyas Ramasubramani

https://doi.org/10.5281/zenodo.17297877

Keeping Python Fun: Using Robotics Competitions to Teach Data Analysis and Application Development

The Issaquah Robotics Society (IRS) has been teaching Python and data analysis to high school students since 2016. Our presentation will summarize what we've learned from nine years of combining Python, competitive robotics, and high school students with no prior programming experience. We'll focus on the importance of keeping it fun, learning the tools, and how to provide useful feedback without making learning Python feel like just another class. We'll also explain how Python helps us win robotics competitions.

Olivia Yang, Brianna Choy, Archit Gouda, Ela Sharma, Stacy Irwin

https://doi.org/10.5281/zenodo.17297879

Using Discrete Global Grid Systems in the Pangeo ecosystem

Over the past few years, Discrete Global Grid Systems (DGGS) that subdivide the earth into (roughly) equally sized faces have seen a rise in popularity. However, their in-memory representation is different from traditional projection-based data, which is either comprised of evenly shaped rectangular grid (aka raster) or discrete geometries (aka vector), and thus requires specialized tooling. In particular, this includes libraries that can work on the numeric cell ids defined by the specific DGGS. xdggs is a library that provides a unified interface for xarray that allows working with and visualizing a variety of DGGS-indexed data sets.

Justus Magin, Alexander Kmoch, Benoît Bovy, Jean-Marc Delouis, Anne Fouilloux, Tina Odaka

https://doi.org/10.5281/zenodo.17297883

Learning the art of fostering open-source communities

Open-source projects are intricate ecosystems that consist of humans contributing in a diverse manner. These contributions are one of the essential elements driving the projects and must be encouraged. The humans behind these contributions play a vital role in constituting the lively and diverse community of the project. Both the humans and their contributions must be preserved and handled with utmost care for the success and evolution of the project. As with every community, certain best practices should be followed to maintain its health, and certain pitfalls should be avoided. In this talk, I'll share what I have learned from maintaining the vibrant and wonderful Zarr project and its community over the years.

Sanket Verma

https://doi.org/10.5281/zenodo.17297885

Unlocking AI Performance with NeMo Curator: Scalable Data Processing for LLMs

A presentation about data processing for training or fine-tuning LLMs using NeMo Curator

Allison Ding

https://doi.org/10.5281/zenodo.17297857

Accelerating Genomic Data Science and AI/ML with Composability

The practice of data science in genomics is fraught with friction. This is largely due to a tight coupling of bioinformatic tools to file input/output, where storage formats are frequently standardized but complex and siloed. In this talk, I argue that the adoption of open standards not restricted to bioinformatics can help better integrate bioinformatic workflows into the wider data science, visualization, and AI/ML ecosystems by promoting "composable" architectures. I present two bridge libraries as vignettes for composable genomic data science. The first, Anywidget, is an architecture and toolkit based on modern web standards for sharing interactive widgets across all Jupyter-compatible runtimes, including JupyterLab, Google Colab, VSCode, Marimo, and more. The second, Oxbow, is a Rust and Python-based library that unifies access to common NGS data formats by efficiently transforming queries into Apache Arrow, a standard in-memory columnar representation for tabular data analytics. I demonstrate the use of these libraries to build custom connected genomic analysis and visualization environments. I propose that bridge libraries such as these, which leverage domain-agnostic standards to unbundle specialized file manipulation, analytics, and web interactivity, can serve as the glue for assembling reusable and flexible genomic data analysis and machine learning workflows as well as systems for exploratory data analysis and visualization.

Nezar Abdennur

https://doi.org/10.5281/zenodo.17297887

Noise Resilient Quantum Computing with Python

Today's quantum computers are far noisier than their classical counterparts. Unlike traditional computing errors, quantum noise is more complex, arising from decoherence, crosstalk, and gate imperfections that corrupt quantum states. Error mitigation has become a rapidly evolving field, offering ways to address these errors on existing devices. New techniques emerge regularly,

requiring flexible tools for implementation and testing. This talk explores the challenges of mitigating noise and how researchers and engineers use Python to iterate quickly while maintaining reliable and reproducible workflows.

Nate Stemen

https://doi.org/10.5281/zenodo.17297889

SciPy's New Infrastructure for Probability Distributions and Random Variables

The SciPy library provides objects representing well over 100 univariate probability distributions. These have served the scientific Python ecosystem for decades, but they are built upon an infrastructure that has not kept up with the demands of today's users. To address its shortcomings, SciPy 1.15 includes a new infrastructure for working with probability distributions. This talk will introduce users to the new infrastructure and demonstrate its many advantages in terms of usability, flexibility, accuracy, and performance.

Albert Steppi, Matt Haberland, Pamphile Roy

https://doi.org/10.5281/zenodo.17352258

Numba v2: Towards a SuperOptimizing Python Compiler

The rapidly evolving Python ecosystem presents increasing challenges for adapting code using traditional methods. Developers frequently need to rewrite applications to leverage new libraries, hardware architectures, and optimization techniques. To address this challenge, the Numba team is developing a superoptimizing compiler built on equality saturation-based term rewriting. This innovative approach enables domain experts to express and share optimizations without requiring extensive compiler expertise. This talk explores how Numba v2 enables sophisticated optimizations —from floating-point approximation and automatic GPU acceleration to energy-efficient multiplication for deep learning models—all through the familiar NumPy API. Join us to discover how Numba v2 is bringing superoptimization capabilities to the Python ecosystem.

Siu Kwan Lam

https://doi.org/10.5281/zenodo.17297895

Taming the Wild West of ML: From Model to Trust

Three years ago, with the arrival of ChatGPT, the world of AI entered a new era. Apllications and tools using AI flourished. However, the security risks of using AI are still not all fully understood. In particular, there is little thought paid to the supply chain security of a model. Tampering with a model has significant impact downstream. In this talk, we show how building upon tamper-proof metadata for ML artifacts, we can secure the ML supply chain. That is, we show how the OpenSSF model signing secures the most important part of the ML supply chain: the model.

Mihai Maruseac

https://doi.org/10.5281/zenodo.17297897

Burning fuel for cheap! Transport-independent depletion in OpenMC

We have added functionality for running depletion simulations independently of neutron transport in OpenMC, an open source Monte Carlo particle transport code with an internal depletion module. Transport-independent depletion uses pre-computed static multigroup cross sections and fluxes to calculate reaction rates for OpenMC's depletion matrix solver. This accelerates the depletion calculation, but removes the spatial coupling between depletion and neutron transport. We used a

simple PWR pincell to validate the method against the existing transport-coupled depletion method. Nuclide concentration errors roughly scale with depletion time step size and are inversely proportional to the amount of the nuclide present in a depletable material. The magnitude of concentration error depends on the nuclide of interest. Concentration errors for low-abundance nuclides at longer (30-day) time steps exhibit large negative initial concentration the becomes more positive with time due to overestimation of nuclide production stemming from the lack of spatial coupling to neutron transport. For ten 3-day time steps, fission product concentration errors are all under 3%. Actinide concentration errors range from 10-15% for Am and Cm, 5-7% for Pu and Np, and 2% and less for U. Surprisingly, the numbers are similar for 30-day time steps. These results demonstrate the potential of this new method with moderate accuracy and extraordinary time savings for low and medium fidelity simulations. Concentration error characterization on larger models remains an open area of interest.

Oleksandr Yardas, Paul Romano, Madicken Muk, Kathryn Huff

https://doi.org/10.5281/zenodo.17297899

Zamba: Computer vision for wildlife conservation

Camera traps are an essential tool for wildlife research. Zamba is an open source Python package that leverages machine learning and computer vision to automate time-intensive processing tasks for wildlife camera trap data. This talk will dive into Zamba's capabilities and key factors that influenced its design and development. Topics will include the importance of code-free custom model training, Zamba's origins in an open machine learning competition, and the technical challenges of processing video data. Attendees will walk away with a better understanding of how machine learning and Python tools can support conservation efforts.

Emily Dorne, Jay Qi

https://doi.org/10.5281/zenodo.17297903

zfit: scalable pythonic likelihood fitting

This talk presents zfit with the newest improvements, a general purpose distribution fitting library for complicated model building beyond fitting a normal distribution. The talk will cover all aspects of fitting with a focus on the strong model building part in zfit; composable distributions with sums, products and more, build and mix binned and unbinned, analytic and templated functions in multiple dimensions. This includes the creation of arbitrary, custom distributions with minimal effort that fulfils everyones need. Thanks to the numpy-like backend used by TensorFlow, zfit is highly performant by using JIT compiled code on CPUs and even GPUs, a showcase for scientific computing faster than numpy.

Jonas Eschle, Albert Puig Navarro, Rafael Silva Coutinho, Nicola Serra, Matthieu Marinangeli, Iason Krommydas

https://doi.org/10.5281/zenodo.17297909

Accepted Posters

A Practical Guide to Data Quality: Problems, Detection, and Strategy

The proliferation of data-driven systems, from machine learning models to business intelligence platforms, has placed unprecedented importance on the quality of the underlying data. The principle of "garbage in, garbage out" has never been more relevant, as poor data quality can lead to flawed models, incorrect business decisions, and a fundamental lack of trust in data systems.

Edson Bomfim

https://doi.org/10.5281/zenodo.17297931

I Upgraded My AI...Now It's Funnier Than Me! A Beginner's Guide to Prompt Engineering for Playfulness

Generative Al (GenAl) is revolutionizing creativity, but can it be funny? Absolutely — but only if you prompt it the right way! Using your funny bone is a creative endeavor. It is a great exercise in getting your mind to think outside the box and Al can accelerate that process. While you probably cannot put a joke on every slide in a business presentation, Al playfulness can help you make your explanations and titles more approachable and more unique.

Jarai Carter

https://doi.org/10.5281/zenodo.17297933

SymPy Tensor Module: A Bug Fix for High-Rank Metric Contractions

In symbolic physics computations, handling tensors of varying ranks—such as vectors, spinors, metrics, and gamma matrices—is essential. A long-standing bug in SymPy's tensor module caused incorrect behavior when contracting metrics with tensors of rank higher than two. This poster presents the nature of the bug, its implications for physics computations, and the elegant one-line fix introduced in Pull Request #28240. The correction ensures proper permutation and contraction of tensor indices, restoring expected behavior in symbolic tensor algebra. This contribution improves the reliability of SymPy for high-energy physics and general relativity applications, where tensor manipulations are fundamental.

Arkadiusz P. Trawiński

https://doi.org/10.5281/zenodo.17297923

Explore Solvable and Unsolvable Equations with SymPy

Why can we solve some equations easily while others seem impossible? This poster explores how Python's free computer algebra system, SymPy, helps determine when a closed-form solution exists and when numerical methods are necessary. Instead of relying on advanced mathematics like Galois theory, we take a fun, hacker-style approach—letting SymPy do the work. We'll analyze different equation types, from polynomials to exponentials and trigonometric functions, using real-world examples like Kepler's equation. Whether you're a scientist, engineer, or Python enthusiast, this poster will sharpen your intuition about solvability and show how to make the most of symbolic and numerical solutions in SymPy.

Carl Kadie

https://doi.org/10.5281/zenodo.17297935

Carmina: Introducing Programming to Latin Poetry

Computational thinking is a problem-solving framework traditionally associated with computer science, but it can be applied to other fields such as humanities computing. Carmina, whose name comes from the Latin word for "songs" or "poems", is a new library under development to demonstrate the algorithmic nature of analytic tasks related to Latin poetry. Carmina has practical use in automating processes like scansion and alliteration finding; further, it has educational utility for classical scholars by translating familiar tasks to a coding context.

Suh Young Choi

https://doi.org/10.5281/zenodo.17297937

Serverless Data Processing with Lithops

Lithops enables users to seamlessly scale single-machine Python code across thousands of serverless functions on AWS Lambda, IBM Cloud Functions, and Google Cloud Functions—eliminating the need to manage clusters. It automates function deployment, dependency management, and data partitioning, allowing fast and scalable execution of tasks such as exploratory data analysis, Monte Carlo simulations, sentiment analysis, and hyperparameter tuning. By integrating directly with Jupyter notebooks, Lithops democratizes access to cloud-scale computing, making high-performance distributed processing accessible to any Python user.

Enrique Molina-Giménez, Pedro García-López

https://doi.org/10.5281/zenodo.17297939

Advancing High Energy Physics Data Analysis with Julia – A Case for JuliaHEP

The computational challenges in HEP require optimized solutions for handling complex data structures, parallel computing, and Just-in-Time (JIT) compilation. While Python and C++ remain the standard, Julia presents an opportunity to improve performance while maintaining usability.

Ianna Osborne, Jerry 🐲 Ling

https://doi.org/10.5281/zenodo.17297941

Weather Classification with CNNs: Successes, Surprises, and Sunrise Confusion

Image-based weather classification has important applications in environmental monitoring and intelligent systems. In this study, a Convolutional Neural Network (CNN) model was developed to classify weather conditions into five classes (cloudy, foggy, rainy, shine, sunrise) using an open-source Kaggle dataset of approximately 1500 labeled images. The dataset was preprocessed, split into training and validation sets using an 85/15 ratio, and augmented to improve generalization. The CNN architecture was implemented in TensorFlow/Keras with data augmentation that included rescaling, rotation, zooming, and horizontal flipping. Model training employed early stopping to prevent overfitting. The model achieved good performance on validation data, with loss and accuracy curves indicating stable learning. However, evaluation on an independent test set highlighted difficulties in accurately classifying the ‘sunrise’ class. These results illustrate both the potential and challenges of applying CNNs to multiclass weather classification.

Jennifer Iloekwe

https://doi.org/10.5281/zenodo.17297946

SciPy Optimize and Reachability of Nonlinear Systems

In this paper, CFSpy a package in Python that numerically computes Chen-Fliess series is presented. The reachable sets of non-linear systems are also calculated with this package. The method used obtains batches of iterated integrals of the same length instead one of at a time. For this, we consider the alphabetical order of the words that index the series. By redefining the iterated integral reading the index word in the opposite direction from right to left, we allow the broadcasting of the computation of the iterated integral to all permutations of a length. Assuming the input is sufficiently well approximated by piecewise step functions in a fine partition of the time interval, the minimum bounding box of a reachable set is computed by means of polynomials in terms of the inputs. To solve the optimization problem, the SciPy library is used.

Ivan Perez Avellaneda

https://doi.org/10.5281/zenodo.17297948

Fast and scalable general geospatial regridding

Being able to regrid between various grid types is very important in geoscience research. While the scientific python ecosystem includes numerous geospatial regridding packages, most of them are tailored to only a few specific grid types. Additionally, very few of them are designed to handle regridding of grids that are too big to fit into memory using distributed computation frameworks like dask. grid-indexing and grid-weights are a set of rust-based libraries that implement regridding between arbitrary grids using a RTree and rely on dask to scale for larger-than-memory grids.

Justus Magin

https://doi.org/10.5281/zenodo.17297950

Quantum Chemistry Acceleration: Comparative Performance Analysis of Modern DFT Implementations

This poster examines the acceleration of quantum chemistry calculations through modern implementations of Density Functional Theory (DFT). We present a comparative performance analysis between traditional frameworks and optimized implementations, demonstrating substantial computational efficiency gains. Applications to electrolyte membrane structure analysis illustrate practical benefits, enabling more extensive simulations in reduced timeframes. Benchmarks highlight speedups achieved via modern code optimization techniques in Python-based quantum chemistry environments.

Kyohei Sahara

https://doi.org/10.5281/zenodo.17353419

Missing data? MArray adds mask support to any array/tensor backend

Masked arrays enable the performance and convenience of (rectangular) array computing where missing data would otherwise result in ragged arrays. Some array libraries, such as NumPy, PyTorch, and Dask, offer partial support for masked versions of their arrays/tensors, but these often lack important features, have APIs that are inconsistent with those of the parent namespace, and see only limited support in downstream libraries (e.g. SciPy). Furthermore, there is no standard interface specification, some of these implementations are treated as “experimental” or are maintained less than the parent library, and some automatically mask the result of invalid numerical calculations, hiding bugs and leading to spurious results. MArray offers a new way forward by adding mask support to any array backend that is compatible with the Array API Standard.

Matt Haberland, Lucas Colley, Justus Magin

https://doi.org/10.5281/zenodo.17352223

Scientific Publishing with MyST Markdown

Current High Energy Physics measurements rely on analysing the data by using underlying theoretical assumptions in model building that do not account for possible deviations from the nominal model on which the assumptions are based. This approach can introduce potential bias in parameter extraction and complicate the reinterpretation of measurements without fully

reanalyzing the data. Redist is a package for building model-agnostic binned-likelihood fits, allowing combination and enhancing the reinterpretation of datasets. Redist allows direct inference of theoretical parameters of any beyond the Standard Model scenario through histogram reweighting, properly taking into account changes in kinematic distributions. By leveraging truth-level observables and capturing correlations, Redist can be interfaced with different theoretical backends to connect theoretical parameters with the pyhf environment for fitting. We present a Redist-HAMMER interface that has been developed to address direct Beyond the Standard Model measurements by reinterpreting existing datasets collected by various High Energy Physics collaborations, using the increasingly popular theoretical backend HAMMER. This enables model-agnostic interpretation of data—a crucial step for advancing precision flavor physics. Further applications of the Redist package in different fields, along the lines of the Redist-HAMMER interface in High Energy Physics, to extend pyhf models are possible and straightforward to implement.

Marco Colonna, Johannes Albrecht, Florian Bernlochner, Lorenz Gärtner, Abhijit Mathad, Biljana Mitreska

https://doi.org/10.5281/zenodo.17297952

Performing Object Detection on Drone Orthomosaics with Meta's Segment Anything Model (SAM)

Accurate and efficient object detection and spatial localization in remote sensing imagery is a persistent challenge. In the context of precision agriculture, the extensive data annotation required by conventional deep learning models poses additional challenges. This paper presents a fully open source workflow leveraging Meta AI's Segment Anything Model (SAM) for zero-shot segmentation, enabling scalable object detection and spatial localization in high-resolution drone orthomosaics without the need for annotated image datasets. Model training and/or fine-tuning is rendered unnecessary in our precision agriculture-focused use case. The presented end-to-end workflow takes high-resolution images and quality control (QC) check points as inputs, automatically generates masks corresponding to the objects of interest (empty plant pots, in our given context), and outputs their spatial locations in real-world coordinates. Detection accuracy (required in the given context to be within 3 cm) is then quantitatively evaluated using the ground truth QC check points and benchmarked against object detection output generated using commercially available software. Results demonstrate that the open source workflow achieves superior spatial accuracy — producing output 20% more spatially accurate, with 400% greater IoU — while providing a scalable way to perform spatial localization on high-resolution aerial imagery (with ground sampling distance, or GSD, < 30 cm).

Nicholas McCarty

https://doi.org/10.5281/zenodo.17297956

Phlower: A Deep Learning Framework Supporting PyTorch Tensors with Physical Dimensions

We present Phlower, an open-source deep learning library that extends PyTorch tensors to incorporate physical dimensions — such as time (T), mass (M), and length (L) — and enforces dimensional consistency throughout computations. When applying deep learning to physical systems, tensors often represent quantities with physical dimensions. Ensuring dimensional correctness in such operations is crucial to maintaining physical validity. To address this challenge, Phlower provides PhlowerTensor, a wrapper class of PyTorch tensor that tracks physical dimensions and ensures that tensor operations comply with dimensional consistency rules. This poster

introduces the core features of Phlower and demonstrates its capability to maintain dimensional correctness through representative examples.

Riku Sakamoto

https://doi.org/10.5281/zenodo.17297960

Imprecise uncertainty management with uncertain numbers to facilitate trustworthy computations

Uncertain number enables faithful characterization and rigorous computation under mixed uncertainties and unknown dependencies, given partial information.

Yu Chen, Scott Ferson

https://doi.org/10.5281/zenodo.17297963

Wavefront-Based Visual Acuity Estimation Using Machine Learning

ARA, Inc. and AFRL RHDO collaboratively develop RECOIL, a machine learning, Python-based application to categorize Air Force (AF) prototype eyewear consistent with military standard MIL-DTL-43511D. RECOIL evaluates the interferogram obtained from a Ronchi test by using computer vision algorithms to identify its aperture and maximize the assignment of pixels to each fringe within it. Metrics are derived from the set of fringes in each image. RECOIL also can fit Zernike polynomials to the wavefront of light that passes through a lens. RECOIL allows users to conduct virtual experiments by deriving and modifying Zernike polynomial coefficients. Recent developments to RECOIL now allow its users to predict visual acuity by conducting a simulated eye exam or Snellen test. Artificial intelligence models, trained to perform similar to but not better than a human with normal vision, serve as human surrogates to classify letters or directions letter face from simulated distortions of letters in an eye exam chart.

Allen S. Harvey Jr., Michael A. Patel, Jason J. Jerwick, Jennifer T. Nguyen, Clare M. Egan, Danny Hinojosa, Brenda J. Novar, Lindsey M. Ferris, Michael C. Cook

https://doi.org/10.5281/zenodo.17297925

Explaining ML predictions with SHAP

As machine learning models become increasingly accurate and complex, explainability has become essential to ensure trust, transparency, and informed decision-making. SHapley Additive exPlanations (SHAP) provide a rigorous and intuitive approach for interpreting model predictions, delivering consistent and theoretically grounded feature attributions. This article demonstrates the application of SHAP across two representative model types: boosted decision trees and neural networks.

Avik Basu

https://doi.org/10.5281/zenodo.17297965

Enhancing Curve Fitting with SciPy: Interactive Spline Modeling and Reproducibility with SplineCloud

Curve fitting is a fundamental task in data science, engineering, and scientific computing, enabling researchers to extract meaningful relationships from data. However, selecting and tuning the right fitting model for complex, noisy, or multidimensional data remains a significant challenge. SciPy plays a critical role in addressing these challenges by providing robust spline fitting methods that offer flexibility and precision. Yet, fine-tuning spline parameters, ensuring stability in extrapolation,

and sharing fitted models for reproducibility remain open problems. In order to address these challenges, we developed SplineCloud - an open platform that provides interactive spline fitting capabilities and uses SciPy on the backend. SplineCloud allows the construction, analysis, and exchange of spline-based regression models using SciPy's interpolate module. SplineCloud's curve fitting tool extends the capabilities of SciPy spline fitting methods by enabling researchers to fine-tune spline parameters: knot vector, control points, and degree interactively, instantly analyzing the accuracy of models. Models constructed on the platform obtain their unique identifiers and become reusable in code, fostering better collaboration and knowledge transfer. Reusability of spline curves and underlying datasets in code is enabled via the open-source SplineCloud client library called splinecloud-scipy, which is also based on SciPy. The proposed approach of interactive cloud-based fitting improves data processing workflows, allows separating data preparation and approximation routines from the main code, and brings FAIR principles to the curve fitting, enabling researchers to construct and share libraries of empirical data relations.

Vadym Pasko

https://doi.org/10.5281/zenodo.17297967

Building Responsible AI: An AI-Driven Framework for Sustainable, Ethical, and Privacy-Preserving IoT Systems

This study presents a multidisciplinary approach to AI system design in IoT ecosystems, focusing on computational efficiency, environmental sustainability, and ethical responsibility. The framework emphasizes four key components: multidisciplinary integration, which utilizes low-power systems, privacy-preserving machine learning architectures, and transparent governance for efficiency and oversight; sustainable practices, which align IoT/AI solutions with ethical principles and privacy preservation; practical implementation, leveraging tools like CarbonTracker for emission tracking, SHAP for interpretability, and PySyft for secure computation in healthcare and urban management; and measurable performance, ensuring accountability through benchmarks. This approach harmonizes technological innovation with sustainability, privacy, and ethical responsibility.

Ying-Jung Chen

https://doi.org/10.5281/zenodo.17297929

AI for Wearable ECG Prototype: Quantified Health

We are exploring how to apply deep learning models and data augmentation methods to a 12-lead ECG prototype embedded in a T-shirt built by Areteus. Combining wearable ECG devices with deep learning may enable the at-home detection of abnormal heartbeats. This technology could expand access for at-risk heart patients, high-performance athletes, and clinicians serving low-income communities. While a formal diagnosis still requires a physician, wearable ECGs can support early warnings and continuous at-home monitoring. For preliminary testing, we use a one-dimensional convolutional neural network (1D-CNN). The model's input data is sliced into one-heartbeat windows, centered using the maximum value, and zero-padded on both sides. To simulate real-world conditions, high-quality 12-lead hospital ECG data was aggressively augmented to mimic artifacts such as chest lead movement, electrical noise from muscle activity, signal drops from poor contact, baseline wander, and powerline interference. Early findings from the simple 1D-CNN model are promising. Even with aggressively augmented, noisy ECG data, the model achieved high diagnostic classification accuracy, sensitivity, and specificity. Future research will include exploring time-series and ResNet-SE models. Real-world testing of the T-shirt prototype is also planned with volunteers in Point Reyes, California.

Jennifer Yoon

https://doi.org/10.5281/zenodo.17297970

SciPy Tools Plenaries

Lightning Talks

Enhancing Curve Fitting with SciPy: Interactive Spline Modeling and Reproducibility with SplineCloud

Curve fitting is a fundamental task in data science, engineering, and scientific computing, enabling researchers to extract meaningful relationships from data. However, selecting and tuning the right fitting model for complex, noisy, or multidimensional data remains a significant challenge. SciPy plays a critical role in addressing these challenges by providing robust spline fitting methods that offer flexibility and precision. Yet, fine-tuning spline parameters, ensuring stability in extrapolation, and sharing fitted models for reproducibility remain open problems. In order to address these challenges, we developed SplineCloud - an open platform that provides interactive spline fitting capabilities and uses SciPy on the backend. SplineCloud allows the construction, analysis, and exchange of spline-based regression models using SciPy's interpolate module. SplineCloud's curve fitting tool extends the capabilities of SciPy spline fitting methods by enabling researchers to fine-tune spline parameters: knot vector, control points, and degree interactively, instantly analyzing the accuracy of models. Models constructed on the platform obtain their unique identifiers and become reusable in code, fostering better collaboration and knowledge transfer. Reusability of spline curves and underlying datasets in code is enabled via the open-source SplineCloud client library called splinecloud-scipy, which is also based on SciPy. The proposed approach of interactive cloud-based fitting improves data processing workflows, allows separating data preparation and approximation routines from the main code, and brings FAIR principles to the curve fitting, enabling researchers to construct and share libraries of empirical data relations.

Vadym Pasko

https://doi.org/10.5281/zenodo.17297836

Table of Contents

1 *A Lightweight Pipeline for Rewards-Guided Synthetic Text Generation Using NeMo and RAPIDS*

JIAJIA DING, ARHAM MEHTA, NIRMAL JULURU

14 *Explaining ML predictions with SHAP*

AVIK BASU

38 *Explore Solvable and Unsolvable Equations with SymPy*

CARL KADIE

54 *Ocetrac: An Object-based Framework for Tracking and Quantifying Climate Events in Gridded Datasets*

CASSIA CAI, JACOB T. COHEN, HILLARY SCANNELL, VALENTINA STANEVA, LUANNE THOMPSON

67 *OptiMask: Efficiently Finding the Largest NaN-Free Submatrix*

CYRIL JOLY

75 *Python is all you need: an overview of the composable, Python-native data stack*

DEEPYAMAN DATTA

85 *Zamba: Computer vision for wildlife conservation*

EMILY DORNE, JAY QI, PETER BULL, COLLEEN STEPHENS, MATTIA BESSONE, BENJAMIN DEBETENCOURT, BARBARA FRUTH, DAVID MORGAN, MEREDITH S. PALMER, CRICKETTE SANZ, JANIKA WENDEFEUER, CATHERINE CROCKFORD, TOBIAS DESCHNER, KEVIN E. LANGERGRABER, ALEX K. PIEL, MARTHA ROBBINS, VOLKER SOMMER, FIONA A. STEWART, ROMAN M. WITTIG, KLAUS ZUBERBUEHLER, HJALMAR S. KÜHL, MIMI ARANDJELOVIC

112 *A Practical Guide to Data Quality: Problems, Detection, and Strategy*

EDSON BOMFIM

124 *On the Path to Seamless Python Serverless Data Analytics*

ENRIQUE MOLINA-GIMÉNEZ, GERMAN T. EIZAGUIRRE, PEDRO GARCÍA-LÓPEZ

138 *An Active Learning Plugin In Napari To Fine-Tune Models For Large-scale Bioimage Analysis*

FERNANDO CERVANTES-SANCHEZ

149 *Advancing High Energy Physics Data Analysis with Julia -- A Case for JuliaHEP*

IANNA OSBORNE, JERRY 🌶 LING

153 *CFSpy: A Python Library for the Computation of Chen-Fliess Series*

IVAN PEREZ AVELLANEDA

173 *Jupyter Book 2 and the MyST Document Stack*

194 *Eyes in the sky: Estimating Inland Water Quality Using Landsat Data*

KEDAR DABHADKAR

202 *Quantum Chemistry Acceleration: Comparative Performance Analysis of Modern DFT Implementations*

KYOHEI SAHARA

207 *Imprecise uncertainty management with uncertain numbers to facilitate trustworthy computations*

YU CHEN, SCOTT FERSON

222 *Redist, a python tool for model-agnostics binned-likelihood fits in High Energy Physics*

MARCO COLONNA, JOHANNES ALBRECHT, FLORIAN BERNLOCHNER, LORENZ GÄRTNER, ABHIJIT MATHAD, BILJANA MITRESKA

232 *Reproducible Machine Learning Workflows for Scientists with Pixi*

MATTHEW FEICKERT, RUBEN ARTS, JOHN KIRKHAM

245 *NeuroConv: Streamlining Neurophysiology Data Conversion to the NWB Standard*

HEBERTO MAYORQUIN, CODY BAKER, PAUL ADKISSON-FLORO, SZONJA WEIGL, ALESSANDRA TRAPPANI, LUIZ TAUFFER, OLIVER RÜBEL, BENJAMIN DICHTER

259 *Performing Object Detection on Drone Orthomosaics with Meta's Segment Anything Model (SAM)*

NICHOLAS MCCARTY

269 *Extension of the OpenMC depletion module for transport-independent depletion*

OLEKSANDR YARDAS, PAUL ROMANO, MADICKEN MUK, KATHRYN HUFF

278 *Pipeline-level differentiable programming for the real world*

ANDRIN REHMANN, DION HÄFNER, ALESSANDRO ANGIOI, YONGQUAN QU, ANDREI PALEYES

286 *Phlower: A Deep Learning Framework Supporting PyTorch Tensors with Physical Dimensions*

RIKU SAKAMOTO

294 *SciPy Proceedings: An Exemplar for Publishing Computational Open Science*

303 *Empowering Learners - Teaching Reproducible Research with Open-Source Tools*

DEBORAH KHIDER, JULIEN EMILE-GEAY, DAVID EDGE, NICHOLAS MCKAY

311 *Numba v2: Towards a SuperOptimizing Python Compiler*
SIU KWAN LAM, STUART ARCHIBALD, STAN SEIBERT

331 *Challenges and Implementations for ML Inference in High-energy Physics*
SANJIBAN SENGUPTA, LORENZO MONETA

342 *Unlocking the Missing 78%: Inclusive Communities for the Future of Scientific Python*
NOOR AFTAB

359 *Enhancing Curve Fitting with SciPy: Interactive Spline Modeling and Reproducibility with SplineCloud*
VADYM PASKO

Open Access | https://doi.org/10.25080/hprh2773

SciPy 2025

July 7 - July 13, 2025

Proceedings of the 24th Python in Science Conference
ISSN: 2575-9752

A Lightweight Pipeline for Rewards-Guided Synthetic Text Generation Using NeMo and RAPIDS

Jiajia Ding[1], **Arham Mehta**[1], and **Nirmal Juluru**[1]

[1]NVIDIA Corporation

Abstract

Synthetic Data Generation (SDG) plays an increasingly important role in modern machine learning workflows. Although progress has been made in structured data—tabular data with fixed schemas—especially with tools like Synthetic Data Vault (SDV) and Conditional Tabular Generative Adversarial Network (CTGAN), the generation of high-quality synthetic text remains a challenging and underdeveloped area. Many existing SDG pipelines lack mechanisms for semantic control, offer limited quality assurance, and are not designed with computational efficiency in mind.

To address these shortcomings, we introduce a lightweight and modular pipeline designed specifically for synthetic text data generation. Our approach integrates instruction-tuned large language models (LLMs) and reward-scoring models from NVIDIA NeMo with GPU-accelerated data processing tools from the RAPIDS ecosystem including cuDF and cuML.

The pipeline is implemented entirely within a Jupyter Notebook, allowing for transparency, reproducibility, and ease of use. It includes stages for data cleaning, semantic deduplication, reward-guided generation, and iterative data augmentation. It leverages RAPIDS libraries to accelerate operations such as clustering and filtering, enabling large-scale experimentation. Empirical results from applying this pipeline to a legal QA dataset demonstrate improved data quality, enhanced semantic diversity, and efficient execution on GPU hardware. The solution is designed to be accessible to data scientists who may not specialize in LLM development, thereby filling a crucial usability gap in the SDG landscape.

Keywords Synthetic Data Generation, Reward Scoring, Semantic Deduplication, GPU Acceleration, NeMo, RAPIDS, cuML, cuDF

Published Jul 10, 2025

Correspondence to
Jiajia Ding
allisond@nvidia.com

Open Access

Copyright © 2025 Ding *et al.*. This is an open-access article distributed under the terms of the Creative Commons Attribution 4.0 International license, which enables reusers to distribute, remix, adapt, and build upon the material in any medium or format, so long as attribution is given to the creator.

1. Introduction

Synthetic data is indispensable for developing machine learning models, when real data is scarce, sensitive, or restricted by privacy regulations. The ability to generate realistic, high-quality synthetic samples enables model development, fine-tuning, and benchmarking without exposing sensitive content. In structured data domains, tools such as SDV, CTGAN, and Gretel have emerged as practical solutions for generating tabular data that preserves statistical properties while mitigating privacy risks [1], [2]. However, the generation of unstructured text or semi-structured data (with flexible, implicit schemas), such as synthetic text, presents a far greater challenge. This complexity arises from the inherently semantic nature of language, which makes quality difficult to assess and even harder to enforce.

Text generation pipelines face several limitations that reduce their utility in practical workflows. First, many systems rely on large language models without sufficient post-generation validation, resulting in outputs that may be fluent but semantically irrelevant or redundant.

Second, few pipelines offer robust mechanisms to filter out low-quality or factually incorrect samples, which is essential when synthetic data is meant to support downstream tasks such as supervised learning or knowledge extraction. Third, most available implementations are not optimized for interactive or scalable use, often lacking GPU acceleration or modularity for iteration and analysis.

To address these gaps, we introduce a pipeline that is reward-aware, reproducible, and optimized for real-world use. Built entirely within a Jupyter notebook, the pipeline connects NeMo's instruction-tuned language models and reward scorers [3] with RAPIDS' GPU-accelerated data handling and clustering libraries [4]. The design is intentional in its focus on usability, aiming to provide data scientists with a transparent and reproducible tool for generating and curating synthetic text data. The pipeline supports full-cycle generation workflows, from preprocessing and semantic deduplication to filtering and output, all with minimal setup and high computational efficiency.

2. Background and Related Work

Structured data generation has seen considerable progress, with tools like SDV, CTGAN, and Gretel offering accessible APIs and statistical validation mechanisms [1], [2]. These systems have enabled synthetic data use in regulated environments such as healthcare and financial services. In contrast, synthetic text generation has largely been approached through either simple prompt-based generation using pretrained LLMs [5] or through more complex reinforcement learning pipelines, such as reinforcement learning with human feedback (RLHF) [6], [7]. Both approaches have their strengths but also introduce new challenges in terms of complexity, reproducibility, and cost.

Prompt-based pipelines are typically easy to initiate but provide limited control over output quality. Without built-in validation or filtering, the generated text can be inconsistent, incoherent, or repetitive. On the other hand, RLHF frameworks introduce reward models that evaluate generated outputs, allowing quality-guided training or generation. However, implementing RLHF involves substantial engineering efforts, including reward function design, reinforcement learning infrastructure, and hyperparameter tuning, which make it impractical for many real-world cases.

Our approach seeks to fill the methodological gap between these extremes. Rather than training models through reinforcement learning, we integrate pretrained reward models directly into the generation pipeline [3]. This allows us to evaluate and filter synthetic samples after generation using a scalar quality score that encapsulates properties such as helpfulness, coherence, relevance, and alignment with instructions [8]. This mechanism introduces a layer of semantic quality control without the need for retraining or fine-tuning.

In addition, semantic deduplication is incorporated to ensure content diversity. This component addresses the issue of redundancy, which is especially important in iterative generation scenarios. Finally, the pipeline is built around RAPIDS, a suite of GPU-accelerated Python libraries that leverage the same APIs as the traditional pandas dataframe and scikit-learn-based operations [4]. By integrating RAPIDS, we improve the scalability and responsiveness of the pipeline, requiring zero to minimal change in coding habits.

3. Methodology

3.1. Pipeline Overview

The synthetic text generation pipeline comprises five core modular stages: (1) data loading and cleaning, (2) semantic deduplication, (3) prompt-based generation, (4) reward scoring,

and (5) integration of accepted samples. To ensure computational efficiency, these stages are GPU-accelerated using the RAPIDS ecosystem.

The entire pipeline is orchestrated within a single notebook and supports interactive development, reproducibility, and iteration. This structure is essential for fast experimentation, especially when working with domain-specific data such as legal documents or biomedical QA pairs.

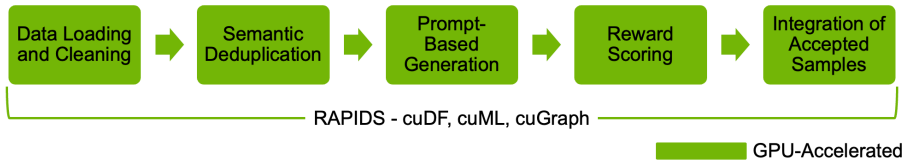


Figure 1. *The Synthetic Text Generation Pipeline*

3.2. Data Loading and Cleaning

The initial stage involves loading raw data, typically stored in JSON Lines (JSONL). The dataset we use as an example contains question-answer pairs and optional metadata such as titles, tags, or confidence scores.

Examples from the legal QA dataset.
Q: What is the difference between a bench trial and a jury trial?
A: In a bench trial the judge is the fact-finder, whereas in a jury trial the jury determines the facts while the judge rules on matters of law.

The data then undergoes a multi-step cleaning process to remove HTML tags, correct malformed Unicode characters, and apply filters based on word count and quality scores. For example, questions and answers are retained only if their lengths fall within acceptable ranges and their associated scores exceed a defined threshold. This step is crucial for establishing a high-quality seed set and preventing malformed data from degrading the performance of downstream generating and scoring models. NeMo Curator supports heuristic filters such as length constraints and reward-score thresholds to eliminate low-information or off-topic samples, improving both embedding quality and reward scoring.

```
# Load JSONL data and preview schema

# Import data
import pandas as pd

path = './peft-curation-with-sdg-70b/data/raw/splits/law-qa-train.jsonl'

# Load your initial dataset
dataset = pd.read_json(
   path, lines = True
)
```

```
# Define HTML/Unicode cleaners and simple filters

from bs4 import BeautifulSoup
import re

def clean_html(text):
   if not isinstance(text, str):
       return ""
   # Use BeautifulSoup to strip HTML tags
   text = BeautifulSoup(text, "lxml").get_text()
   # Replace multiple spaces or newlines with a single space and trim
   return re.sub(r"\s+", " ", text).strip()

import ftfy
from ftfy import TextFixerConfig

# Configuration for fixing Unicode text issues
fix_config = TextFixerConfig()

def fix_unicode(text):
   # Fix encoding and Unicode errors in text using ftfy.
   if not isinstance(text, str):
       return ""
   return ftfy.fix_text(text, config=fix_config)

def word_count(text, min_words=50, max_words=500):
   words = text.strip().split()
   # Check if the number of words in text falls within a specified range.
   return min_words <= len(words) <= max_words

def filter_low_score(score, threshold=0):
   # Determine if a numeric score meets or exceeds a given threshold.
   return score >= threshold
```

```
# Apply row-wise cleaning and content filters

def clean_filter_by_row(row):
   # Clean + fix each text field
   for field in ["title", "question", "answer"]:
       text = clean_html(row[field])
       text = fix_unicode(text)
       row[field] = text

   # Apply filters
   question_bool = word_count(row["question"]) and filter_low_score(float(row["question_score"]))
   answer_bool = word_count(row["answer"]) and filter_low_score(float(row["answer_score"]))

   return question_bool and answer_bool

def data_clean(dataset):
   dataset["keep"] = dataset.apply(clean_filter_by_row, axis=1)
   clean_dataset = dataset[dataset["keep"]].drop(columns=["keep"]).reset_index(drop=True)
   dataset.drop("keep", axis = 1, inplace = True)

   return clean_dataset
```

3.3. Semantic Deduplication

To avoid training or evaluating models on semantically repetitive content, we implement a deduplication step based on sentence embeddings. Each record is converted into a single textual representation by concatenating the title, question, and answer fields. Sentence embeddings are computed using a pretrained SentenceTransformer model and normalized using CuPy to enable efficient GPU operations. K-Means clustering is then applied in the embedding space to identify groups of similar samples. Within each cluster, cosine similarity is used to identify redundant records. Those with high proximity to the cluster centroid are marked for removal. This process significantly improves content diversity and reduces the likelihood of overfitting in downstream tasks. We define "duplicate" as two samples whose concatenated title, question, and answer embeddings have cosine similarity ≥ 0.99 within the same K-Means cluster.

While deduplication is crucial to reduce redundancy and overfitting, very aggressive thresholds can prune in-domain paraphrases or stylistic variations that are valuable for robustness. To mitigate this, we (i) experiment with variable similarity thresholds across data subsets (e.g., by topic/source/length) and (ii) perform human-in-the-loop auditing on a stratified sample to calibrate the operational definition of "semantic duplicate." These safeguards help retain informative variation while removing true repeats.

Examples from the legal QA dataset.
Removed as near-duplicates (≥ 0.99):
Q1: How does a bench trial differ from a jury trial?
Q2: Bench vs. jury trial—who determines the facts?
Kept as non-duplicates (< 0.99):
Q1: When can a defendant waive the right to a jury trial?
Q2: Can a judge overturn a jury verdict?

In tie cases, we retain the item farthest from the cluster centroid to preserve variety. This procedure improves content diversity and reduces overfitting risk in downstream tasks while avoiding the removal of useful paraphrases.

```
# Compute embeddings, cluster with K-Means, and mark near-centroid duplicates

def semantic_dedupe(dataset, n_clusters = 1000, eps_to_extract = 0.01):
   dataset["text"] = dataset["title"] + dataset["question"] + dataset["answer"]


   # Generate normalized sentence embeddings
   from sentence_transformers import SentenceTransformer
   import cupy as cp


   model = SentenceTransformer("all-MiniLM-L6-v2")
   embeddings = model.encode(dataset["text"].tolist(), batch_size = 128, show_progress_bar = True,
convert_to_numpy = True)
   embeddings_norm = cp.array(embeddings)/cp.linalg.norm(cp.array(embeddings), axis = 1, keepdims =
True)


   # Perform KMeans with cosine normalization (Euclidean KMeans on normalized vectors)
   from sklearn.cluster import KMeans
   from sklearn.metrics.pairwise import cosine_similarity


   kmeans = KMeans(n_clusters = n_clusters, random_state = 1234, max_iter = 100)
   labels = kmeans.fit_predict(embeddings_norm)
   centroids = kmeans.cluster_centers_


   # Deduplication logic (find points within eps_to_extract)
   dedup_indices = []
   for cluster_id in range(n_clusters):
       cluster_points = cp.where(cp.array(labels) == cluster_id)[0]
       if len(cluster_points) == 0:
           continue

       cluster_embeddings = embeddings_norm[cluster_points].get()
       centroid = centroids[cluster_id].reshape(1, -1)
       sims = cosine_similarity(cluster_embeddings, centroid).flatten()
       within_eps = cluster_points[sims >= (1 - eps_to_extract)]


       if len(within_eps) > 0:
           dedup_indices.extend(within_eps)

   # Get deduplicated data
   mask = dataset.index.isin(dedup_indices)
   kept_data = dataset[~mask]


   # Drop the added column
   dataset.drop("text", axis = 1, inplace = True)
   kept_data.drop("text", axis = 1, inplace = True)


   return kept_data
```

In the above code snippet, n_clusters (also written as k) is the number of K-Means clusters used to pre-group records in embedding space before duplicate checks; it controls granularity (larger k arrow.r finer, more homogeneous groups; smaller k arrow.r coarser groups). eps_to_extract (ε) is a within-cluster proximity/similarity threshold for marking items as redundant (e.g., "duplicate if cosine similarity $\geq \tau$" or "duplicate if distance to the centroid $\leq \varepsilon$"), which controls how aggressively near-duplicates are removed. Both are user-tunable hyperparameters; appropriate values depend on corpus size, diversity, and the desired precision-recall trade-off for deduplication. Our implementation exposes both as configuration options so readers can adjust them without changing the pipeline.

3.4. Synthetic Data Generation with Rewards

Synthetic text generation is performed using the Llama 3.1 Nemotron 70B instruct model, which supports tasks such as question generation and paraphrasing. Prompt templates are designed for three specific functions: generating questions from answers, paraphrasing questions, and paraphrasing answers. For each function, the generator outputs a candidate sample that is then evaluated by the Llama 3.1 Nemotron 70B Reward model. This reward model assigns a scalar score reflecting the overall quality of the text. Only those samples with scores falling within a specified range are retained. This scoring function enables automated quality control and reduces reliance on manual inspection or external validation. Notably, the Llama 3.1 Nemotron 70B Reward model is ranked No. 1 on the Hugging Face RewardBench leaderboard as of October 1, 2024 [9], outperforming competing reward models across overall, chat, safety, and reasoning categories.

Although our experiments use NVIDIA Nemotron models for both generation and reward scoring, the pipeline is model-agnostic: the generator and reward model are accessed through the same chat-completion interface and can be replaced by any instruction-tuned LLM and any scalar-scoring reward model that expose comparable APIs. Practically, portability requires (i) an instruction-following endpoint with controllable sampling (temperature, top_p, optional seed), (ii) sufficient context length for our prompts, and (iii) a reward signal, either a native reward model or a proxy judge that outputs a single scalar per sample. To remain robust across models, we store raw scores and apply per-model score normalization, selecting candidates by percentile thresholds rather than absolute cutoffs. Prompts are maintained as templated strings and versioned, so swapping models is a configuration change rather than a code change.

```
# Synthetic Data Generation with Rewards

def normalize_score(score, min_val=-34.75, max_val=-5.125):
   # Normalize a scalar score to [-1, 1] range.
   # We use a minimum value of −34.75 and a maximum value of −5.125, computed from the raw dataset,
for normalization in this SDG process.
   normalized = 2 * (score - min_val) / (max_val - min_val) - 1
   return normalized

def initialize_generator():
   # Initialize Nemotron generator and client.
   from openai import OpenAI
   from nemo_curator import OpenAIClient
   from nemo_curator.synthetic import NemotronGenerator

   openai_client = OpenAI(
       base_url="https://integrate.api.nvidia.com/v1",
       api_key=""
   )
   client = OpenAIClient(openai_client)
   generator = NemotronGenerator(client)
   return generator, client

def generate_variants(generator, document, prompt_template, n_variants, sdg_model, sdg_model_kwargs):
   # Generate paraphrased or question variants from a document.
   generated = generator.generate_closed_qa_instructions(
       document=document,
       n_openlines=n_variants,
       prompt_template=prompt_template,
       model=sdg_model,
       model_kwargs=sdg_model_kwargs,
   )
   variant_list = [entry.strip("*").strip() for entry in generated[0].split("\n") if entry]
   return variant_list

def build_synthetic_row(row, i, gen_question, gen_answer, gen_title, rewards_normalized):
   # Build a new row dictionary for synthetic data.
   return {
       "id": f"{row['id']}-synth-{i}",
       "question": gen_question,
       "answer": gen_answer,
       "title": gen_title,
       "file_name": "law-stackexchange-questions-answers.json.synth",
       "tags": row["tags"],
       "question_score": rewards_normalized,
       "answer_score": rewards_normalized
   }

def SDG(dataset, sample_size_perc=0.001, rewards_threshold=0, n_variants=1):
   # Sample rows, generate/paraphrase Q&As, generate reward scores, filter by threshold, and return
curated synthetic pairs as a DataFrame.
   import random
   import pandas as pd

   generator, client = initialize_generator()

   sdg_model = "nvdev/nvidia/llama-3.1-nemotron-70b-instruct"
   sdg_model_kwargs = {
       "temperature": 0.5,
       "top_p": 0.9,
       "max_tokens": 1024,
       "seed": 1234
   }
   reward_model = "nvdev/nvidia/llama-3.1-nemotron-70b-reward"

   PROMPT_GENERATE_QUESTIONS_FROM_ANSWER = """TEXT:
   {document}

   Given the above text, generate exactly {n_openlines} questions that can be answered by the text.
All questions must be answerable by the text and be relevant to the text.
   Do not directly reference the text in the questions.
   Every question should be a complete sentence and end with a question mark. There should be no
other text besides the questions.
   Begin each question with `* ` and end each question with a newline character. Also, each question
```

```
must be concise.
   Make sure to generate exactly {n_openlines} questions.
   """

   PROMPT_PARAPHRASE_TEXT = """TEXT:
   {document}

   Given the above text, paraphrase the text. Produce exactly {n_openlines} variants.
   There should be no other text besides the paraphrased text.
   The paraphrased text must be shorter than the original text. The paraphrased text must be
factually correct and relevant to the original text.
   Begin each variant with `* ` and end each variant with a newline character.
   Make sure to generate exactly {n_openlines} variants.
   """

   output = []
   N = len(dataset)
   n = int(N * sample_size_perc)
   sample_indices = random.sample(range(N), n)

   for idx in sample_indices:
       row = dataset.iloc[idx]
       question = row["question"]
       answer = row["answer"]

       gen_title_lst = generate_variants(generator, answer, PROMPT_GENERATE_QUESTIONS_FROM_ANSWER,
n_variants, sdg_model, sdg_model_kwargs)
       gen_question_lst = generate_variants(generator, question, PROMPT_PARAPHRASE_TEXT, n_variants,
sdg_model, sdg_model_kwargs)
       gen_answer_lst = generate_variants(generator, answer, PROMPT_PARAPHRASE_TEXT, n_variants,
sdg_model, sdg_model_kwargs)

       for i in range(n_variants):
           messages = [
               {"role": "user", "content": f"{gen_title_lst[i]}\n\n{gen_question_lst[i]}"},
               {"role": "assistant", "content": f"{gen_answer_lst[i]}"},
           ]
           rewards = client.query_reward_model(messages=messages, model=reward_model)
           rewards_normalized = normalize_score(rewards)

           if rewards_normalized >= rewards_threshold:
               new_row = build_synthetic_row(row, i, gen_question_lst[i], gen_answer_lst[i],
gen_title_lst[i], rewards_normalized)
               output.append(new_row)

   output_df = pd.DataFrame(output)
   return output_df
```

3.5. Complete SDG Pipeline

The complete pipeline integrates all components into an iterative loop. The process begins with cleaning and deduplicating the original dataset to establish a high-quality baseline. In each subsequent iteration, new synthetic samples are generated and merged with the deduplicated dataset from the previous round. The combined dataset is then subjected to semantic deduplication to eliminate newly introduced redundancy. This iterative procedure supports the progressive enrichment of the dataset while maintaining high semantic diversity and quality standards. All outputs, including intermediate generations and reward scores, are retained to facilitate auditability and further analysis.

```
# Synthetic Data Generation Pipeline

def SDG_Rewards_Pipeline(dataset, round_num, n_clusters = 1000, eps_to_extact = 0.01,
sample_size_perc = 0.001, rewards_threshold = 0, n_variants = 1):

   cleaned_dataset = data_clean(dataset)
   deduped_dataset = semantic_dedupe(dataset, n_clusters = n_clusters, eps_to_extract =
eps_to_extact)
   print(f"After the initial curation, the dataset has {len(deduped_dataset)} records (originally
{len(dataset)}).")

   dataset = deduped_dataset
   for num in range(round_num):

      sdg_dataset = SDG(dataset, sample_size_perc = sample_size_perc, rewards_threshold =
rewards_threshold, n_variants = n_variants)
      dataset = pd.concat([dataset, sdg_dataset], axis = 0, ignore_index = True)
      deduped_dataset = semantic_dedupe(dataset)

      print(f"After round {num + 1}, the dataset has {len(deduped_dataset)} records (originally
{len(dataset)})")

   return deduped_dataset
```

3.6. Accelerating End-to-End Pipeline

To support high throughput execution, the pipeline uses GPU acceleration for key operations through RAPIDS libraries such as cuDF and cuML [4]. With cuDF, common pandas DataFrame operations, including filtering, joining, and aggregating, can be seamlessly accelerated on the GPU. With cuML, machine learning algorithms such as K-Means clustering also benefit from GPU acceleration, resulting in significant reductions in execution time.

Users can enable this acceleration by simply loading the RAPIDS extensions at the beginning of their Jupyter Notebook or JupyterLab, without needing to modify their existing pandas or scikit-learn code. This transparent acceleration allows the pipeline to interactively process tens of thousands or even millions of records, making it practical for rapid prototyping as well as scalable production workflows.

```
# Activate GPU Acceleration of cuML and cuDF in Jupyter Notebook or JupyterLab

%load_ext cuml.accel
%load_ext cudf.pandas
```

For terminal workflows, users can run the same code as a script with the CLI, which activates both the cuML accelerator and the cuDF pandas accelerator for the session.

```
# Activate GPU Acceleration of cuML and cuDF in the terminal

python -m cudf.pandas -m cuml.accel sgd.py
```

4. Experiments and Results

4.1. Use Case Setup

To evaluate the performance and effectiveness of the proposed pipeline, we applied it to a real-world dataset containing 19,474 text records. This dataset served as the basis for a multi-stage process involving semantic deduplication, prompt-based generation, reward scoring, and integration of accepted samples. The evaluation was conducted using a single

GPU-enabled machine to validate the feasibility of interactive, large-scale processing within a notebook environment.

An initial baseline was established by cleaning and deduplicating the original dataset. This step reduced the corpus to 12,244 records, removing approximately 37.2% of entries that failed to meet minimal content quality or uniqueness standards. This baseline served as the foundation for all subsequent generations and filtering stages.

4.2. Quantitative Evaluation

The pipeline was configured to run for 10 rounds, each involving the generation of synthetic samples sampled at a rate of 0.1% from the current dataset, with a number of variants of 2. A reward threshold of 0 was applied to filter outputs based on quality, using normalized scalar reward scores to retain only fluent, relevant, and contextually appropriate text.

As shown in Table 1, each round resulted in a modest increase in the dataset size before deduplication. For example, after round 1, the dataset increased from 12,244 to 12,251 records. This pattern continued across all 10 iterations, with post-generation counts ranging from 12,251 to 12,321. After each merge, semantic deduplication was applied, bringing the dataset size down, confirming that only non-redundant and unique samples were retained.

Table 1. *Dataset size across 10 rounds*

Round	Original Records (Pre-Dedup)	Records After Dedup
Initial	19,474	12,244
1	12,251	12,248
2	12,257	12,257
3	12,266	12,264
4	12,274	12,271
5	12,285	12,282
6	12,293	12,291
7	12,301	12,300
8	12,310	12,309
9	12,313	12,311
10	12,321	12,317

This process illustrates the effectiveness of the deduplication step. Across all 10 rounds, redundant content introduced during generation was systematically identified and removed. Despite generating over 400 synthetic candidates in accumulation, the dataset size only increased slightly, reflecting strict enforcement of semantic quality constraints.

4.3. Visual Outputs

Raw vs Curated. Figure 2 (raw dataset, N=19,474) and Figure 3 (curated dataset, N=77) compare the distribution of the normalized reward score before and after curation. The raw set shows a broad, roughly unimodal spread from $\approx$-0.8 to 0.8 with a mode slightly right of zero and a positive tail to $\sim$0.7–0.8, reflecting high variability in quality. By contrast, the curated set eliminates negatives and concentrates mass in 0.0–0.5 (peaking $\sim$0.10–0.25) with reduced variance—a clear rightward shift.

These visualizations support the filtering logic: clustering and semantic deduplication remove repeated phrasing, and reward-based selection favors higher-coherence, factually

correct content. In short, the procedure is content-selective rather than merely size-reducing, retaining the most valuable and accurate samples for downstream use.

In the raw dataset (Figure 2), the normalized reward score is broadly distributed (≈-0.8 to +0.8) with a single mode slightly right of zero. Density peaks near 0.1 and stays substantial through ∼0.3–0.4; negatives are thinner, while the positive tail extends farther (to ∼0.7–0.8). Overall, the raw set skews mildly positive, with many mid-quality samples and few extreme outliers.

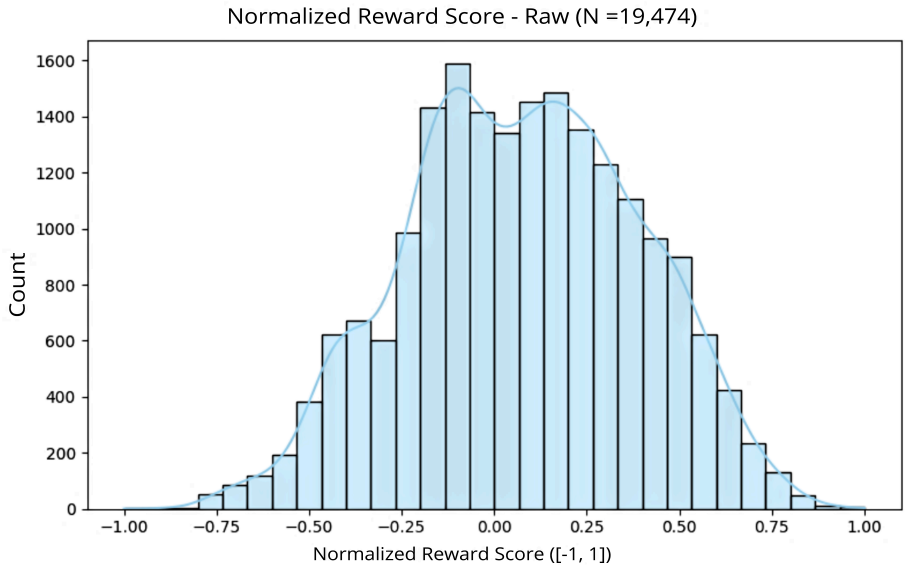


Figure 2. *Normalized Reward Score Distribution of Raw Data*

In the curated dataset (Figure 3), the distribution concentrates at higher scores (≈0.0–0.5), eliminating negatives and tightening the spread. Density peaks around 0.10–0.25, then declines with a thin right tail to ∼0.5; relative to the raw set, it's right-shifted and less variable, indicating curation favored higher-reward examples.

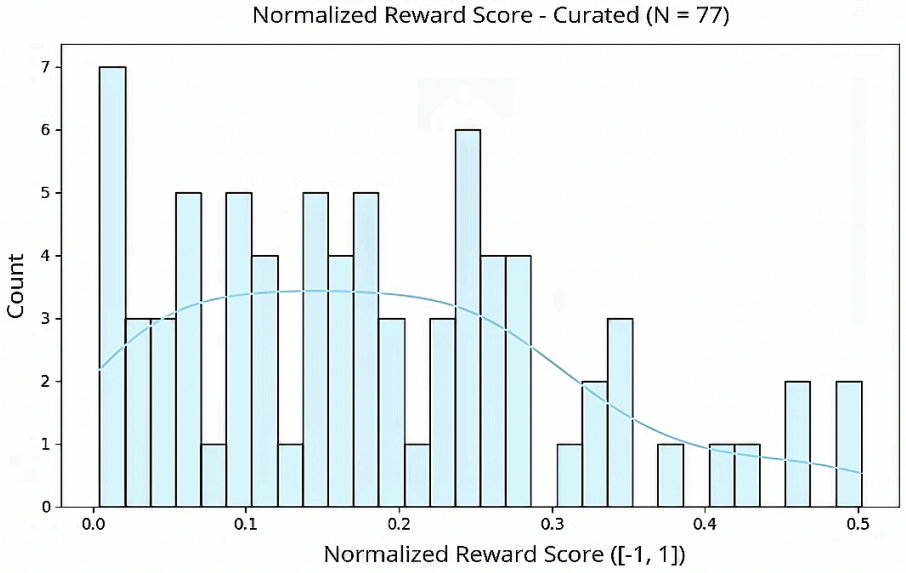


Figure 3. *Normalized Reward Score Distribution of Curated Data*

5. Conclusions

We have presented a reproducible and scalable pipeline for reward-guided synthetic text generation. The pipeline integrates instruction-tuned generation models with semantic deduplication and quality scoring, with end-to-end GPU acceleration. By relying on pre-trained components and avoiding the complexity of reinforcement learning, the pipeline remains accessible to data scientists without deep expertise in LLM training. Its modular structure supports iterative enrichment of datasets while preserving semantic diversity and ensuring quality. All components are implemented in a Jupyter notebook, supporting transparency, auditability, and ease of extension. This work contributes a practical and scientifically grounded approach to responsible synthetic text generation in real-world machine learning settings.

References

[1] N. Patki, R. Wedge, and K. Veeramachaneni, "The synthetic data vault," in *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, 2016, pp. 399–410. doi: 10.1109/DSAA.2016.49.

[2] SDV Developers, "SDV: Synthetic Data Vault." [Online]. Available: https://sdv.dev/

[3] NVIDIA, "NeMo Curator and Nemotron Models." [Online]. Available: https://developer.nvidia.com/nemo

[4] RAPIDS AI, "cuDF and cuML: GPU-accelerated DataFrames and Machine Learning." [Online]. Available: https://rapids.ai/

[5] T. Wolf *et al.*, "Transformers: State-of-the-art natural language processing," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 2020, pp. 38–45. doi: 10.18653/v1/2020.emnlp-demos.6.

[6] P. F. Christiano, J. Leike, T. B. Brown, M. Martic, S. Legg, and D. Amodei, "Deep Reinforcement Learning from Human Preferences," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2017. doi: 10.48550/arXiv.1706.03741.

[7] L. Ouyang, J. Wu, X. Jiang, and others, "Training Language Models to Follow Instructions with Human Feedback," *arXiv preprint arXiv:2203.02155*, 2022, doi: 10.48550/arXiv.2203.02155.

[8] H. Gadre, S. Shen, W. L. Hamilton, and C. Raffel, "RewardBench: Benchmarking Alignment Rewards for Language Models," *arXiv preprint arXiv:2410.01257*, 2024, doi: 10.57967/hf/2457.

[9] Hugging Face, "RewardBench leaderboard: Llama 3.1 Nemotron 70B Reward." [Online]. Available: https://huggingface.co/nvidia/Llama-3.1-Nemotron-70B-Reward

Open Access | https://doi.org/10.25080/mhum9729

Explaining ML predictions with SHAP

Avik Basu[1,2]

[1]Intuit, Inc., [2]PyOpenSci

SciPy 2025

July 7 - July 13, 2025

Proceedings of the 24th Python in Science Conference

ISSN: 2575-9752

Abstract

As machine learning models become increasingly accurate and complex, explainability has become essential to ensure trust, transparency, and informed decision-making. SHapley Additive exPlanations (SHAP) provide a rigorous and intuitive approach for interpreting model predictions, delivering consistent and theoretically grounded feature attributions. This article demonstrates the application of SHAP across two representative model types: boosted decision trees and neural networks.

We utilize the UCI Adult Income dataset with an XGBoost model to predict if a client will subscribe to a term deposit in a bank and the Human Activity Recognition Using Smartphones dataset with a convolutional neural network (CNN) to classify activities (e.g., walking, sitting) based on sensor data. The paper concludes with a discussion of SHAP's practical utility, strengths, and limitations, guiding readers on effective usage in real-world scenarios.

Keywords explainable AI, Shapley values

1. Introduction

Machine Learning models have advanced to the point where they are being used in high-stakes decision-making processes. Finance, healthcare and technology are just a few examples of industries that are using machine learning to make decisions that affect people's lives. However, the increase in performance of these models have come at the cost of interpretability. This is especially true for models with complex architectures such as deep neural networks.

In the industry, model decisions can influence loan approvals, medical diagnoses or hiring decisions. Therefore, an inability to explain why a model produced a certain output can hinder adoption, erode user trust and raise ethical or compliance concerns. Deep neural networks and Large Language Models (LLMs) despite being powerful tools, are often treated as black boxes.

SHapley Additive exPlanations (SHAP [1]) is one such explainability method that aims to make models more interpretable. It uses a game-theoretic approach that provides a way to interpret the predictions of any machine learning model. SHAP provides a mechansim to understand the contribution of each feature to the prediction of a model. It provides different levels of interpretability ranging from a global view of the model to local explanations for individual predictions.

In this article, we demonstrate how SHAP can be used to interpret two classes of models that are especially common in industry:

- **Gradient Boosted Decision Trees** (GBDTs), which are widely used for structured tabular data due to their strong performance and relative robustness
- **Convolutional Neural Networks** (CNNs), which are popular in domains like image and sensor data where spatial relationships matter.

Published Jul 10, 2025

Correspondence to
Avik Basu
avik_basu@intuit.com

Open Access

Copyright © 2025 Basu. This is an open-access article distributed under the terms of the Creative Commons Attribution 4.0 International license, which enables reusers to distribute, remix, adapt, and build upon the material in any medium or format, so long as attribution is given to the creator.

We illustrate the use of SHAP with two representative datasets:

- The Bank Marketing dataset [2] for the GBDT model, where the task is to predict whether a customer will subscribe to a term deposit based on demographic and interaction data.
- The Human Activity Recognition Using Smartphones dataset [3] for the CNN model, where the goal is to classify 6 different physical activities from time-series sensor data collected from mobile devices.

Through these use cases, we will demonstrate the application of SHAP to two different model types and data modalities. Finally, we will discuss the practical utility, strengths, and limitations of SHAP, and provide guidelines for its effective use in real-world scenarios.

2. Core Concepts

In this section, we shall briefly go over the key concepts necessary to understand the approach of SHAP, and its application to neural networks and GBDTs.

2.1. Shapley Values

Shapley values, originating from cooperative game theory, were introduced by Lloyd Shapley [4]. They provide a consistent method to fairly distribute credit or reward among players in a cooperative game. In the context of machine learning, each "player" corresponds to a feature, and the "game" is the prediction task. The Shapley value for a feature quantifies its average marginal contribution to the prediction across all possible coalitions of features.

Formally, the Shapley value for a feature i is defined as:

$$\phi_i = \sum_{S \subseteq N \setminus \{i\}} \frac{|\,S\,|!\,(\,|\,N\,| - |\,S\,| - 1\,)!}{|\,N\,|!} \Big[\, f_{S\cup\{i\}}(x_{S\cup\{i\}}\,) - f_S(x_S) \,\Big] \tag{1}$$

where

- N is the set of all features
- S is a subset of features excluding feature i
- f_S is the model's prediction function when only the features in S are used
- x_S is the input vector with only the features in S.

2.2. SHAP Values

SHAP values as introduced by Lundberg and Lee [1] are a specific implementation of Shapley values to explain predictions from machine learning models. It defines the SHAP values as a linear model of feature contributions [5].

The SHAP explanation model g is defined as:

$$g(z) = \phi_0 + \sum_{i=1}^{M} \phi_i(i) z_i \tag{2}$$

where

- z_i is a binary coalition variable indicating whether feature i is present (1) or absent (0)
- $\phi_i(i)$ is the SHAP value for feature i when it is present
- ϕ_0 is the baseline prediction
- M is the number of features

For a specific input x, the above equation simplifies to:

$$g(x) = \phi_0 + \sum_{i=1}^{M} \phi_i(x) \tag{3}$$

where

- ϕ_0 is the baseline prediction, typically the mean prediction over the dataset
- $\phi_i(x)$ is the SHAP value for feature i for input x
- M is the number of features

2.3. Estimation of SHAP Values

The exact calculation of SHAP values is computationally infeasible for most models due to the combinatorial nature of the Shapley value formula. The computational complexity is $O(2^M)$, where M is the number of features. Therefore, various approximation methods are used to estimate the SHAP values, each tailored to different model types.

- **Kernel SHAP**: Uses a weighted linear regression to approximate shap values for arbitrary models. Although it is applicable to any model type, it can be computationally expensive for large datasets.
- **Tree SHAP**: This method [6] is specifically designed for tree-based models such as decision trees, random forests, and gradient boosted decision trees. It exploits the herarchical structure of decision trees to efficiently compute SHAP values [7].
- **Deep SHAP**: Approximation method that uses a modified version of the DeepLIFT method [8] to estimate SHAP values for deep neural networks including CNNs. It leverages backpropagation to efficiently compute feature attributions.

In this paper, using the `shap` Python library, we will specifically employ Tree SHAP for GBDTs and Deep SHAP for CNNs.

3. SHAP for Decision Trees

Gradient Boosted Decision Trees (GBDTs) [9], [10], [11] are powerful, robust and interpretable models that are widely used in the industry. They are highly effective for structured tabular data, and can handle both numerical and categorical features in a seamless manner. GBDTs in general are interpretable models, and can be interpreted by analyzing the nodes and splits in the decision trees. However, as the complexity of the model as well as the number of features increases, it becomes more valuable to use a more principled approach like SHAP for interpretation.

3.1. Dataset

The Bank Marketing dataset [2] contains information about direct marketing campaigns (phone calls) of a Portuguese banking institution. It contains customer demographic information, financial details and interation history. The goal is to predict whether a customer will subscribe to a term deposit [12]. The dataset is available on the UCI Machine Learning Repository [13].

The key details of the dataset are:

- **Number of samples**: 45,211
- **Number of features in the dataset**: 16
    - Number of features used in the model: 15
    - Removed feature: `duration` (in seconds)
- **Number of classes**: 2 (yes: 1, no: 0)
    - yes: the client will subscribe to a term deposit

    - no: the client will not subscribe to a term deposit
- **Objective**: Binary classification

Note that the `duration` feature is removed from the model training since it is not available before the call is made, and hence cannot be used for prediction. Moreover, it is highly correlated with the target variable.

The dataset can be downloaded from the UCI Machine Learning Repository using the `ucimlrepo` Python library.

```
import pandas as pd
from ucimlrepo import fetch_ucimlrepo

bank_marketing_ds = fetch_ucimlrepo(id=222)

x_df: pd.DataFrame = bank_marketing_ds.data.features
y_df: pd.DataFrame = bank_marketing_ds.data.targets
```

Program 1. *Downloading the Bank Marketing dataset*

3.2. Model

We use an XGBoost [10] model to predict whether a customer will subscribe to a term deposit or not. Specifically, we follow the following steps to train the model:

- Convert the categorical features into pandas [14], [15] categorical data type

```
# x_df is the dataframe containing the features
# Convert all columns to categorical type
for col in x_df.columns:
    x_df[col] = pd.Categorical(x_df[col])
```

Program 2. *Converting categorical features to pandas categorical data type*

- Split the dataset into train and test sets
    - 80% for training and 20% for testing with random shuffling using `scikit-learn` [16], [17]

```
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(
    x_df, y_df, test_size=0.2, random_state=42
)
```

Program 3. *Splitting the dataset into train and test sets*

- Train an XGBoost model on the train set
    - Use default hyperparameters
    - Use categorical features using the `enable_categorical` parameter

```
from xgboost import XGBClassifier

xgb_model = XGBClassifier(
    enable_categorical=True,
    objective="binary:logistic",
    seed=42
)
xgb_model.fit(x_train, y_train)
```

Program 4. *Training an XGBoost model*

- Evaluate the model on the test set

Note that we do not perform any feature engineering or hyperparameter tuning in this example, since the goal is to demonstrate the use of SHAP for interpretation rather than building a high-performing model.

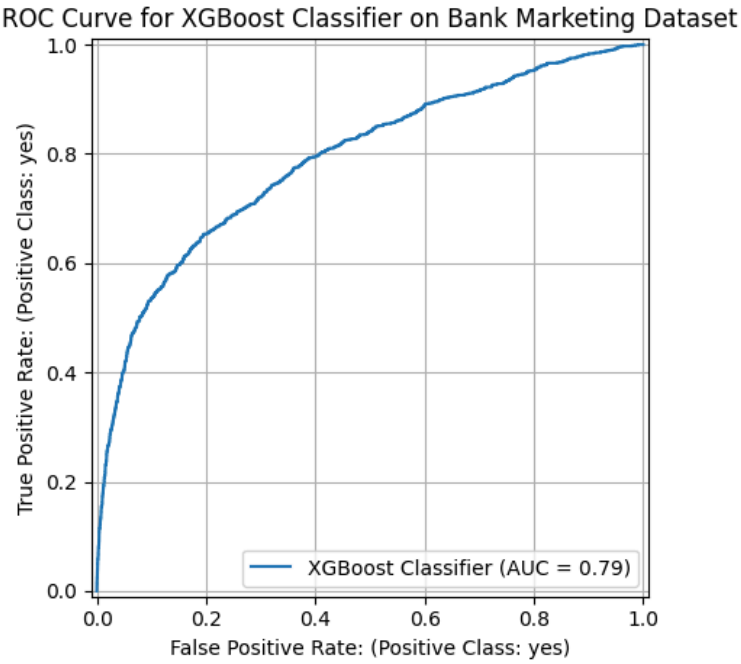


Figure 1. *Performance of the XGBoost model on the test set.*

The performance of the model on the test set is shown in Figure 1.

3.3. SHAP Explanations

In this section, we demonstrate how to apply SHAP to interpret the XGBoost model's predictions on the dataset. We explore both global and local explanations to understand model behavior at different levels of granularity. Global explanations reveal which features are most influential across the entire dataset and how they generally affect predictions, while local explanations show feature contributions for individual predictions. Additionally, we examine dependency plots to understand the relationship between specific feature values and their impact on model predictions.

We use the `shap` Python library [7] to compute the SHAP values for the XGBoost model.

```
import shap

# xgb_model is the trained XGBoost model
# x_test is the test set

explainer = shap.TreeExplainer(xgb_model)
shap_values = explainer.shap_values(x_test)
```

Program 5. *Computing SHAP values for the XGBoost model*

3.3.1. *Global*

The global explanations provide a high-level summary of the model's behavior. They are useful for understanding the overall impact of each feature on the model's predictions.

The shap library provides a number of plotting functions to visualize the SHAP values. The code below uses the `beeswarm` plot to visualize the global SHAP values.

```
import matplotlib.pyplot as plt

ax = plt.subplot()
ax.grid(True)
ax = shap.plots.beeswarm(shap_values, max_display=16, show=False, log_scale=False)
ax.set_title("Global SHAP Values for XGBoost Classifier")
plt.tight_layout()
```

Program 6. *Computing global SHAP values for the XGBoost model*

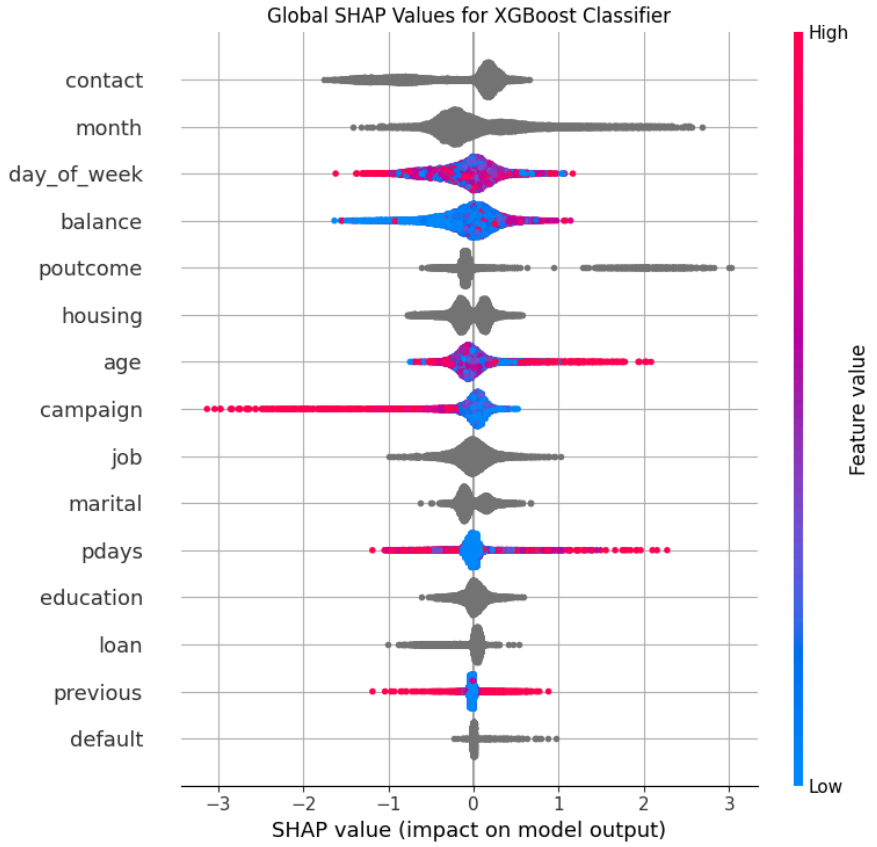


Figure 2. *Global SHAP values for the XGBoost model on the Bank Marketing dataset.*

Note that in Figure 2, the features are ordered by their mean absolute SHAP values in descending order. The x-axis shows the SHAP values, and the y-axis shows the features. Each point represents a sample from the test set.

For numerical features (e.g., age and balance), the dots are color-coded based on the feature values—red signifies high values, and blue denotes low values. This color gradient helps visualize the relationship between the actual feature values and their contribution to predictions. For categorical features (e.g., marital and job), dots are typically displayed in gray because categorical variables do not have a natural ordering. Here, the focus is primarily on their SHAP value distribution rather than the specific category value.

The X-axis of the beeswarm plot represents SHAP values, which quantify the impact each feature has on the model's predictions. A positive SHAP value indicates that the feature pushes the model's prediction higher i.e. towards the positive class (yes), while a negative SHAP value implies the feature reduces the model's predicted value, i.e. towards the nega-

tive class (no). The distance from the center (zero) indicates the magnitude of this impact. The further away a point is from zero, the stronger the feature's influence on that specific prediction.

Thus, features with wide distributions across the X-axis have substantial variation in their impact on different instances, while features tightly clustered near zero exert relatively minor influence on the predictions.

In terms of insights, the beeswarm plot (Figure 2) shows some interesting observations. For example, customers with more balance (`balance` feature) in their account are more likely to subscribe to a term deposit. On the other hand, customers who have been contacted more frequently in the current campaign (`campaign` feature) are less likely to subscribe to a term deposit.

However, the beeswarm plot has limitations when interpreting categorical features. Unlike numerical features where color coding represents the magnitude of feature values (red for high, blue for low), categorical features are displayed in gray since they lack a natural ordering or continuous scale. This makes it difficult to understand which specific categories within a feature drive positive or negative SHAP values. For instance, while we can see that `contact` is at the top of feature importance, the beeswarm plot doesn't reveal whether customers contacted via `cellular` or `telephone` are more likely to subscribe. Dependency plots help address this limitation by explicitly showing the relationship between each category value of the feature and its corresponding SHAP values.

3.3.2. *Dependency*

The dependency plots show the relationship between a feature and the model's predictions. They are useful for understanding how the model uses a particular feature to make predictions.

The code block below defines a function to plot the dependency plot for a categorical feature.

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import shap

def plot_shap_categorical(
  shap_values: shap.Explanation,
  x: pd.DataFrame,
  feature_name: str
 ) -> plt.Axes:

    # Extract SHAP values and feature values
    feature_idx = x.columns.tolist().index(feature_name)
    feature_values = x.iloc[:, feature_idx]
    shap_values_feature = shap_values[:, feature_idx].values

    # Map categories to numeric values for plotting
    categories = feature_values.unique()
    category_to_num = {cat: num for num, cat in enumerate(categories)}
    feature_values_numeric = feature_values.map(category_to_num)

    # Create scatter plot with categories on x-axis
    _, ax = plt.subplots(figsize=(10, 6))
    ax.scatter(
        feature_values_numeric,
        shap_values_feature,
        alpha=0.7,
        s=120,
    )

    # Replace numeric x-ticks with category labels
    ax.set_xticks(
      ticks=np.arange(len(categories)), labels=categories, rotation=45, fontsize=12
    )

    # Reference line at y=0
    ax.axhline(y=0, color="gray", linestyle="--")

    # Labels and title
    ax.grid(True)
    ax.set_xlabel(f"Categorical Feature: {feature_name}")
    ax.set_ylabel("SHAP Value")
    ax.set_title(f"SHAP Dependence Plot for Categorical Feature '{feature_name}'")

    plt.tight_layout()
    return ax
```

Program 7. *Computing categorical dependency plots*

The dependency plot for the categorical feature `contact` can be plotted as:

```python
ax = plot_shap_categorical(shap_values, x_test, "contact")
```

Program 8. *Plotting the dependency plot for the categorical feature `contact`*

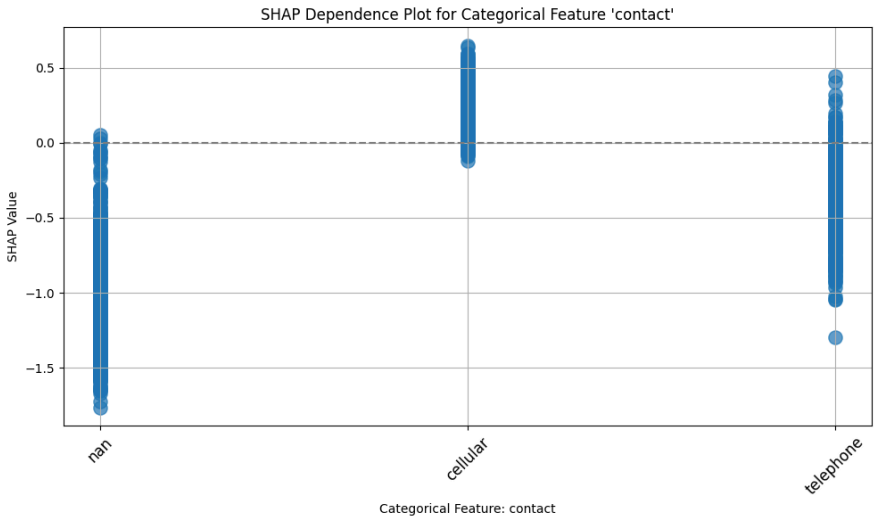


Figure 3. *SHAP dependency plot for the categorical feature* `contact` *on the Bank Marketing dataset.*

The X-axis of the categorical dependency plot represents the categories of the feature, while the y-axis represents the SHAP values. Similar to the beeswarm plot, the positive SHAP values indicate that the feature pushes the model's prediction higher (towards the positive/yes class), while the negative SHAP values imply the feature reduces the model's predicted value (towards the negative/no class).

Figure 3 shows that the `contact` feature has a considerable impact on the model's predictions. Customers who were contacted via cellular (`cellular`) are more likely to subscribe to a term deposit than those who were contacted via telephone (`telephone`).

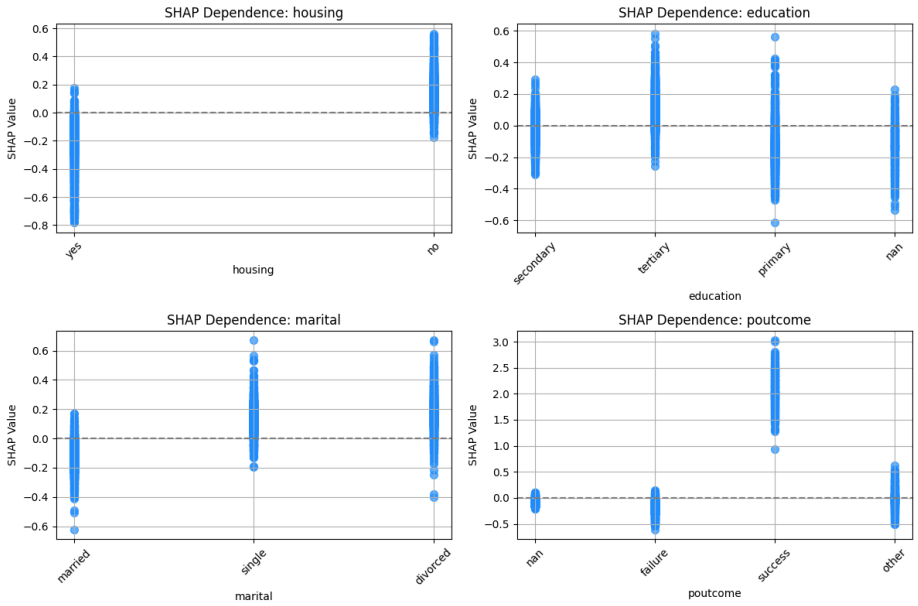


Figure 4. *SHAP dependency plot for some of the other categorical features.*

Figure 4 shows the dependency plots for some of the other categorical features. On the top left, we see the dependency plot for the `housing` feature. Interestingly, customers who have a housing loan (`yes`) are less likely to subscribe to a term deposit than those who do not have a housing loan (`no`). On the bottom right, we see the dependency plot for the feature `poutcome`,

which represents the outcome of the previous marketing campaign. Customers who were previously contacted and the outcome was a `success` are more likely to subscribe to a term deposit than those who were previously contacted and the outcome was a `failure`.

4. SHAP for CNNs

Convolutional Neural Networks (CNNs) are a powerful class of deep learning models particularly well-suited for tasks involving spatial or temporal data, such as image recognition, speech recognition, and time-series classification. CNNs leverage convolutional operations and hierarchical feature extraction to capture intricate patterns within data, making them exceptionally effective for complex datasets [18], [19].

However, despite being so effective, CNNs as with most deep learning models, are often treated as black boxes. In this section, we will demonstrate how SHAP can be used to interpret the predictions of a one-dimensional CNN model.

4.1. Dataset

The Human Activity Recognition Using Smartphones dataset [3] contains sensor data collected from a smartphone. The data has been collected from a group of 30 volunteers within the age group of 19-48 years. The goal is to classify 6 different physical activities from multivariate timeseries sensor data.

The key details of the dataset are:

- **Data type**: Multivariate timeseries
- **Number of training samples**: 7352
- **Number of testing samples**: 2947
- **Number of features in the dataset**: 563
- **Feature types**: Numeric
- **Number of classes**: 6
    - Walking
    - Walking Upstairs
    - Walking Downstairs
    - Sitting
    - Standing
    - Laying
- **Objective**: Multi-class classification

4.2. Model

For this task, we are going to be constructing a 1D convolutional network using PyTorch [20]. Prior to model training, we need to perform some important preprocessing steps.

4.2.1. *Preprocessing*

1. Scale the data so that data is centered around zero mean and a unit variance.

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)

# Zero indexing for the labels
y_train = y_train["label"].apply(lambda x: x - 1)
y_test = y_test["label"].apply(lambda x: x - 1)
```

Program 9. *Data scaling*

1. Create sliding window sequences

```
from torch import Tensor

# Convert to PyTorch tensors
x_train = torch.from_numpy(x_train.to_numpy()).float()
y_train = torch.from_numpy(y_train.to_numpy()).float()
x_test = torch.from_numpy(x_test.to_numpy()).float()
y_test = torch.from_numpy(y_test.to_numpy()).float()


def create_sequences(
    data: Tensor, labels: Tensor, seq_length: int = 64
) -> tuple[Tensor, Tensor]:
    """Create sequences from the data and labels."""

    sequences = []
    labels_seq = []
    for i in range(len(data) - seq_length):
        sequences.append(data[i : i + seq_length])
        labels_seq.append(labels[i + seq_length])
    x_seq, y_seq = torch.stack(sequences), torch.stack(labels_seq)
    return x_seq.permute(0, 2, 1), y_seq


SEQ_LENGTH = 64
x_train_seq, y_train_seq = create_sequences(x_train, y_train, seq_length=SEQ_LENGTH)
x_test_seq, y_test_seq = create_sequences(x_test, y_test, seq_length=SEQ_LENGTH)
```

Program 10. *Creating sliding window sequences*

4.2.2. *Model Definition*

Since we are focused on demonstrating the use of SHAP for interpretation, we are going to skip fine-tuning the model for optimal performance. Instead, we are going to use 2 layer CNN architecture with a full connected layer at the end for classification.

The following code block defines the CNN model class.

```
import torch
import torch.nn as nn

class CNNClassifier(nn.Module):
    def __init__(self, input_dim: int, num_classes: int = 6):
        super().__init__()
        self.layer1 = nn.Sequential(
            nn.Conv1d(input_dim, 32, kernel_size=3, stride=2, padding=1),
            nn.Tanh(),
        )
        self.layer2 = nn.Sequential(
            nn.Conv1d(32, 16, kernel_size=3, stride=2, padding=1),
            nn.Tanh(),
        )
        self.fc = nn.Linear(256, num_classes)

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        x = self.layer1(x)
        x = self.layer2(x)
        x = x.view(x.size(0), -1)
        x = self.fc(x)
        return x

model = CNNClassifier(input_dim=x_train_seq.shape[1], num_classes=6)
```

Program 11. *Defining the CNN model class*

4.2.3. *Training*

1. Define training parameters

```
NUM_EPOCHS = 50
BATCH_SIZE = 32
LEARNING_RATE = 1e-3

criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=LEARNING_RATE)
```

Program 12. *Defining training parameters*

1. Create data loaders

```
from torch.utils.data import DataLoader, TensorDataset

train_ds = TensorDataset(x_train_seq, y_train_seq.long().squeeze())
test_ds = TensorDataset(x_test_seq, y_test_seq.long().squeeze())

# Avoid shuffling since we are using a time series dataset
train_loader = DataLoader(train_ds, batch_size=batch_size, shuffle=False)
test_loader = DataLoader(test_ds, batch_size=batch_size, shuffle=False)
```

Program 13. *Creating data loaders*

1. Train the model

```
for epoch in range(NUM_EPOCHS):
    model.train()
    epoch_loss = 0.0

    for inputs, labels in train_loader:
        optimizer.zero_grad()
        outputs = model(inputs)

        loss = criterion(outputs, labels)
        loss.backward()

        optimizer.step()
        epoch_loss += loss.item()

    avg_loss = epoch_loss / len(train_loader)
    print(f"Epoch [{epoch}/{num_epochs}], Loss: {avg_loss:.5f}")
```

Program 14. *Training the CNN model*

4.2.4. *Evaluate performance*

The performance of the model on the test set is shown below.

Class	Precision	Recall	F1-Score	Support
Walking	0.96	0.94	0.95	496
Walking Upstairs	0.94	0.93	0.93	471
Walking Downstairs	0.92	0.88	0.90	420
Sitting	0.86	0.83	0.85	467
Standing	0.83	0.92	0.87	501
Laying	0.95	0.94	0.94	528
Accuracy			**0.91**	**2883**
Macro Avg	0.91	0.91	0.91	2883
Weighted Avg	0.91	0.91	0.91	2883

4.3. SHAP Explanations

In this section, we will explore how SHAP can be used to interpret the predictions of the CNN model.

The `DeepExplainer` class requires a background dataset for approximating the conditional expectations of the SHAP values. However, since the complexity of the method scales linearly with the number of background data samples passing in a balanced subset of data is needed. Our dataset has 6 target classes, hence it is important to have a good representation for each class.

```
RNG = np.random.default_rng(42)

def create_balanced_background(
    x_data: torch.Tensor, y_data: torch.Tensor, n_per_class: int = 20
) -> tuple[torch.Tensor, torch.Tensor]:
    """
    Create a balanced background dataset with equal representation from each class.

    Parameters:
    -----------
    x_data : torch.Tensor
        Input features
    y_data : torch.Tensor
        Target labels
    n_per_class : int
        Number of samples to include per class

    Returns:
    --------
    torch.Tensor
        Balanced background dataset
    """
    y_np = y_data.detach().cpu().numpy()
    balanced_indices = []

    for cls in np.arange(NUM_CLASSES):
        cls_indices = np.where(y_np == cls)[0]
        if len(cls_indices) >= n_per_class:
            # If we have enough samples, randomly select n_per_class
            selected_indices = RNG.choice(cls_indices, n_per_class, replace=False)
            # selected_indices = np.random.choice(cls_indices, n_per_class, replace=False)
        else:
            # If not enough samples, use all available with replacement
            selected_indices = RNG.choice(cls_indices, n_per_class, replace=True)
            print(
                f"Warning: Class {cls} has only {len(cls_indices)} samples, using with replacement"
            )
        balanced_indices.extend(selected_indices)

    np.random.shuffle(balanced_indices)
    return x_data[balanced_indices], y_data[balanced_indices]


background_data, background_labels = create_balanced_background(
    x_train_seq, y_train_seq, n_per_class=20
)
```

Program 15. *Creating a balanced background dataset*

The next step is to calculate the shap values for the subset of the test set or for the entire test set.

```
import shap

# Limit the number of samples for faster computation
MAX_SAMPLES = 1000

explainer = shap.DeepExplainer(model, background_data)
shap_values = explainer.shap_values(x_test_seq[:MAX_SAMPLES])

# shap_values shape: (MAX_SAMPLES, num_features, SEQ_LENGTH, num_classes)

# get the value for the last time step
feature_importances = shap_values[:, :, -1, :]
```

Program 16. *Computing SHAP values for the CNN model*

4.3.1. *Global*

Since this is a multiclass classification problem, we will look at the global SHAP values for each class separately in a one-vs-all fashion. For the beeswarm plots below, the positive

SHAP values indicate that the feature pushes the model's prediction towards the positive class (the class in question), while the negative SHAP values imply the feature reduces the model's predicted value towards the negative class (other classes).

Walking

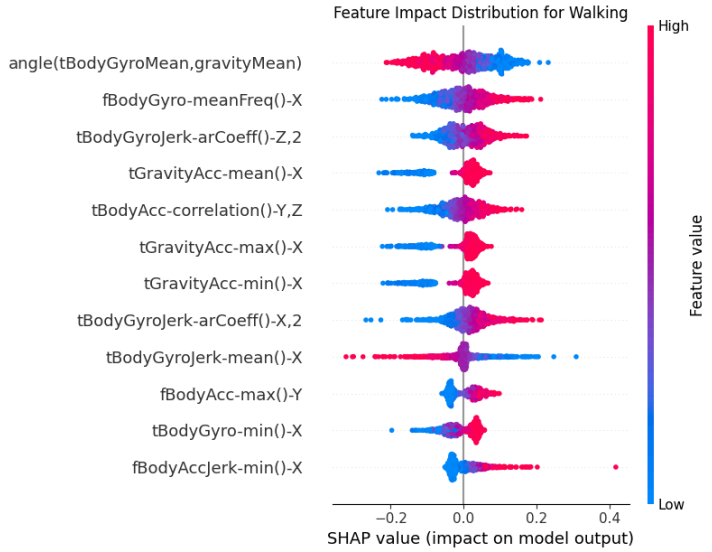


Figure 5. *Global SHAP values for the CNN model for the `Walking` class.*

From Figure 5, we can see that the top feature for the `Walking` class is `angle(tBodyGyroMean, gravityMean)`. The vectors `gravityMean` and `tBodyGyroMean` are obtained by averaging the signals in a signal window sample. The `angle()` function measures the angle between these two vectors. In the context of the `Walking` class, low values of this feature (in blue) are more clustered towards the positive SHAP values. This implies that when a person is walking, the angle between the mean gyroscope signal and the mean gravity signal is smaller. Another important feature is `tBodyGyroJerk-arCoeff()-Z,2`, is the second autoregressive coefficient of the time-domain Z-axis gyroscope jerk signal which is the derivative of angular velocity. Autoregressive coefficients capture how a signal's current value relates to its previous values. Higher values of this coefficient reflect smoother, more predictable rotation changes and are more likely to occur when a person is walking.

Walking Upstairs

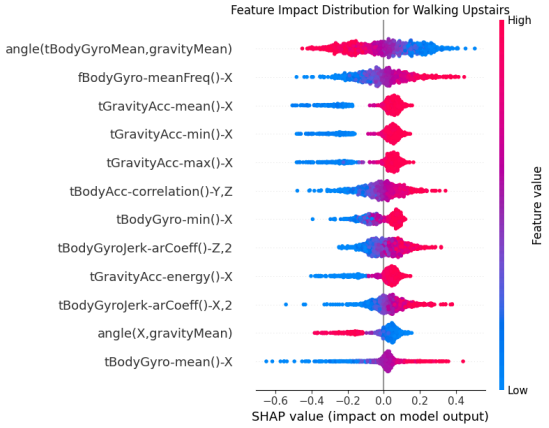


Figure 6. *Global SHAP values for the CNN model for the `Walking Upstairs` class.*

For the `Walking Upstairs` class as shown in Figure 6, the top features include `tGravityAcc-mean()-X`, `tGravityAcc-min()-X`, `tGravityAcc-max()-X` which are all related to the X component of the gravity acceleration. Lower values of these features (in blue) are more clustered towards the negative SHAP values. This implies that when a person is walking upstairs, the X component of the gravity acceleration is higher. Intuitively, when walking upstairs, people naturally lean into the slope, so the gravity vector has a strong component along the phone's X-axis.

Walking Downstairs

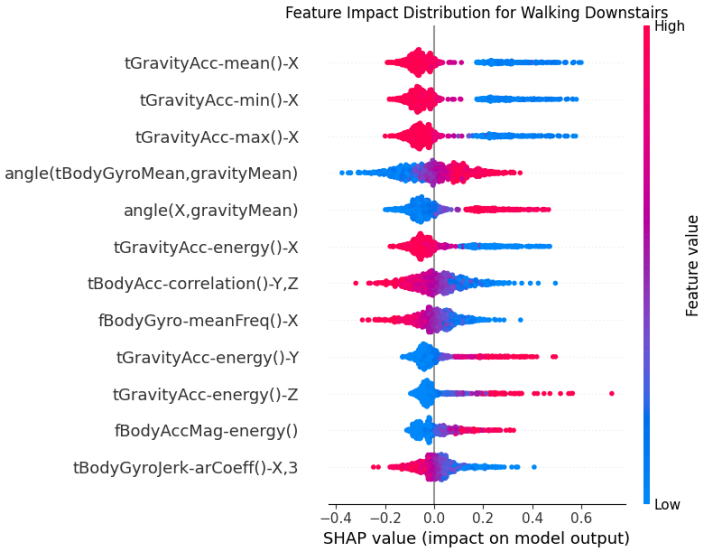


Figure 7. *Global SHAP values for the CNN model for the `Walking Downstairs` class.*

For the `Walking Downstairs` class as shown in Figure 7, we see the opposite behavior for the `tGravityAcc-mean()-X`, `tGravityAcc-min()-X`, `tGravityAcc-max()-X` features. Higher values of these features (in orange) are more clustered towards the negative SHAP values. While descending stairs, the torso actually tilts backward instead of forward, so the gravity vector's projection onto the device's X-axis flips sign.

Sitting

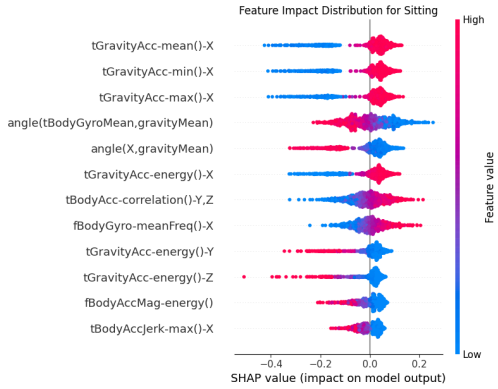


Figure 8. *Global SHAP values for the CNN model for the `Sitting` class.*

For the `Sitting` class, the main features are different from the other `Walking` related classes. As shown in Figure 8, the top feature is `angle(X, gravityMean)` which measures how much

the device's X-axis is tilted relative to the average gravity vector. Low values (in blue) of this feature are more clustered towards the positive SHAP values. The intuition is that when a person is sitting, the attached device on the waist makes a smaller and consistent angle with the vertical compared to other activities, especially compared to when the person is laying down.

Standing

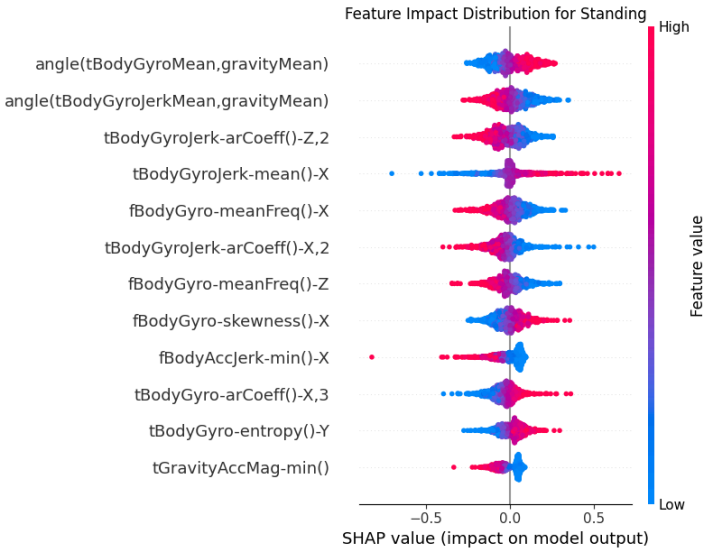


Figure 9. *Global SHAP values for the CNN model for the `Standing` class.*

In Figure 9, we can see that one of the top features for the `Standing` class is `tBodyGyroJerk-arCoeff()-Z,2`. As mentioned earlier, the autoregressive coefficients capture how a signal's current value relates to its previous values. Low values of this feature (in blue) are more clustered towards the positive SHAP values. This implies that for standing, the Z-axis gyroscope jerk signal is essentially flat. The higher-order coefficients (like the second one) collapse toward zero because past values have almost no bearing on the current value. This is in contrast to the `Walking` class (Figure 5) where the gyroscope jerk signal is more dynamic and hence the second autoregressive coefficient is higher.

Laying

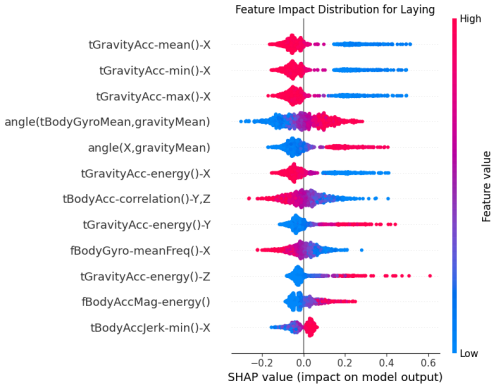


Figure 10. *Global SHAP values for the CNN model for the `Laying` class.*

For the `Laying` class, in Figure 10, the feature behaviors are very different to the other classes. The low values of the top feature `tGravityAcc-mean()-X`, `tGravityAcc-min()-X`, `tGravityAcc-max()-X` is much more distributed towards the positive SHAP values for the class. This is opposite to what we see in Figure 8 for the `Sitting` class. Similarly, the second feature `angle(X, gravityMean)` also shows a reversed behavior compared to the `Sitting` class. This contrast makes sense, since when a person is laying down, the attached device on the waist makes a larger angle with the vertical compared to when the person is sitting.

4.3.2. *Dependency*

For this dataset, all the features are numerical, and hence we can plot the dependency plots directly. We shall look at the top 3 features for the classes `Walking`, `Sitting` and `Laying`.

The dependency plots focus on the relationship between the feature values, the SHAP values and the model output. The x-axis represents the feature values and the y-axis represents the SHAP values. As before, positive SHAP values indicate that the feature pushes the model's prediction towards the positive class (the class in question), while the negative SHAP values imply the feature reduces the model's predicted value towards the negative class (other classes).

For example in Figure 11 for the `Walking` class, we see that the SHAP values for the feature `angle(tBodyGyroMean, gravityMean)` are positive for low values of the feature, meaning that when the angle between the mean gyroscope signal and the mean gravity signal is smaller, the model is more likely to predict the `Walking` class. For the next feature `fBodyGyro-meanFreq()-X`, we see that the SHAP values are positive for high values of the feature, meaning that when the mean frequency of the X-axis gyroscope signal is higher, the model is more likely to predict the `Walking` class.

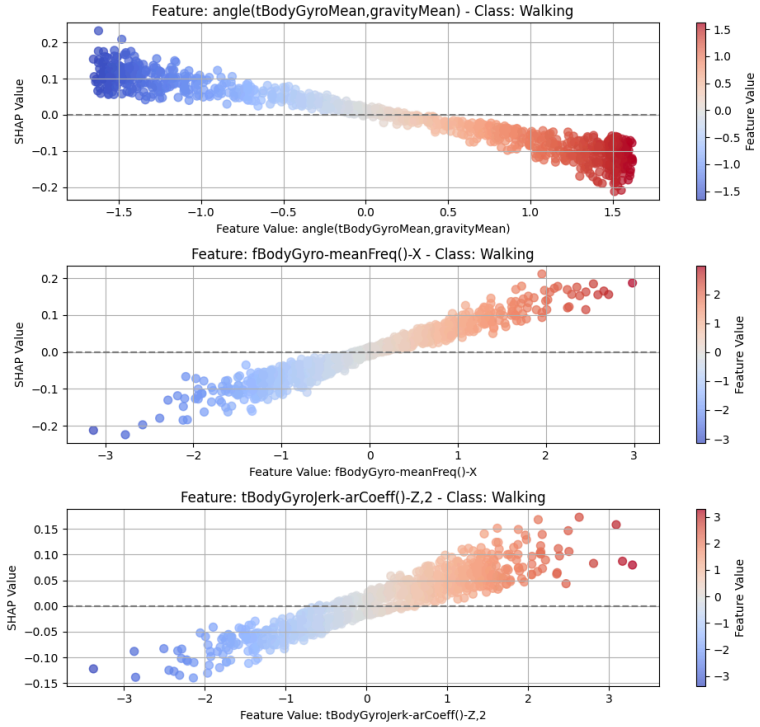


Figure 11. *SHAP dependency plot for the CNN model for the `Walking` class.*

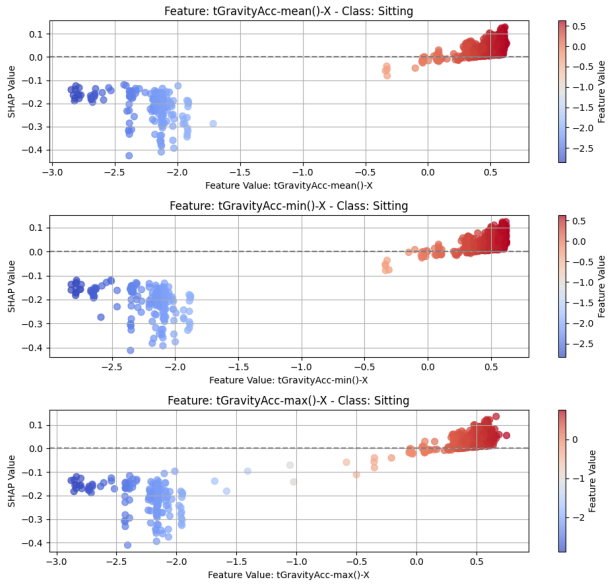


Figure 12. *SHAP dependency plot for the CNN model for the* `Sitting` *class.*

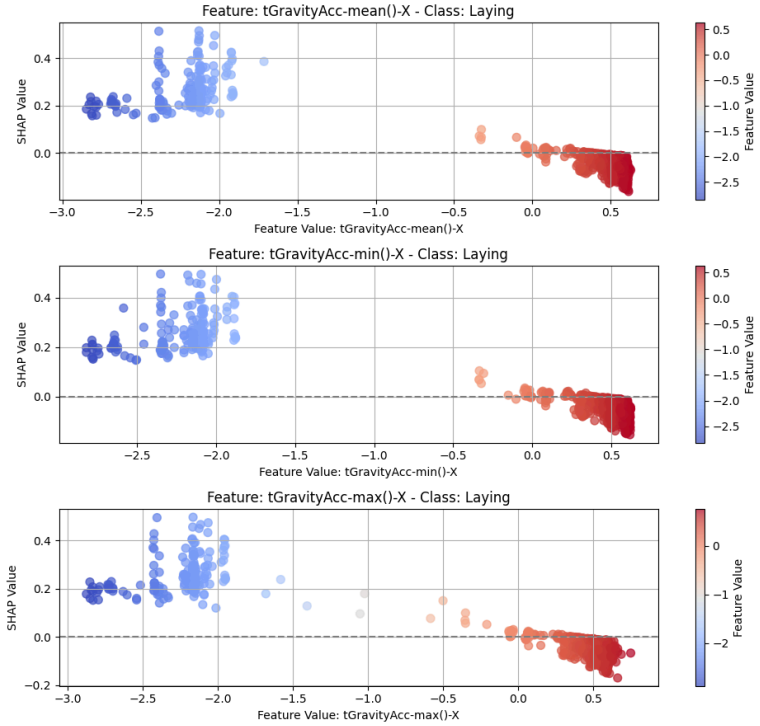


Figure 13. *SHAP dependency plot for the CNN model for the* `Laying` *class.*

4.3.3. *Local*

Local explanations provide insights into the factors influencing individual predictions. They are useful for understanding why the model made a particular prediction for a specific instance. There are many types of local explaination plots that can be used, e.g. force plots, bar plots, waterfall plots, etc. In this section, we shall look at the bar plots.

The red and blue bars represent the positive and negative SHAP values respectively. The length of the bars indicates the magnitude of the SHAP value. The bars are ordered by the magnitude of the SHAP values.

For instance, in Figure 19, we take a look at the local SHAP values for a particular sample that was predicted to be in the `Laying` class. We see that most top features contribute positively to the prediction of the `Laying` class, which are denoted by the red bars. There are a few features that contribute negatively to the prediction, which are denoted by the blue bars, but they are not very significant overall.

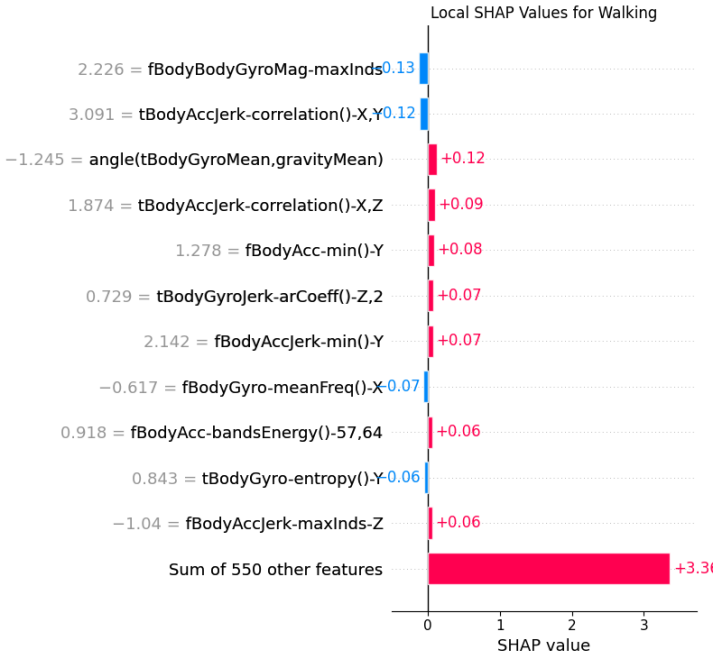


Figure 14. *Local SHAP values for the CNN model for the `Walking` class.*

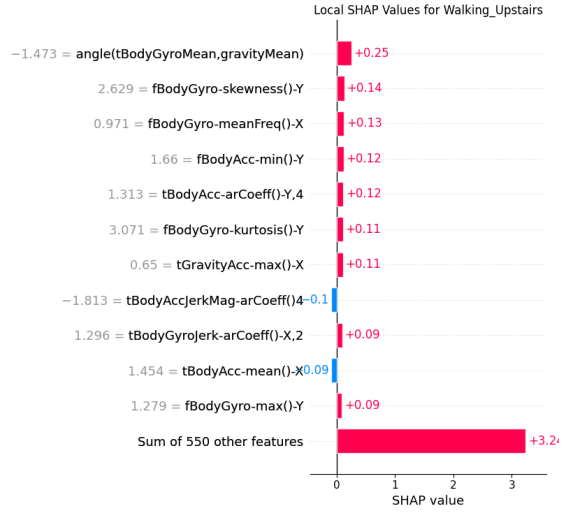


Figure 15. *Local SHAP values for the CNN model for the `Walking Upstairs` class.*

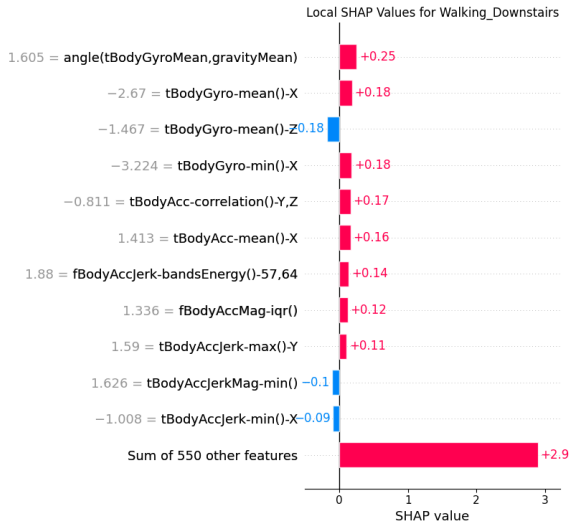


Figure 16. *Local SHAP values for the CNN model for the `Walking Downstairs` class.*

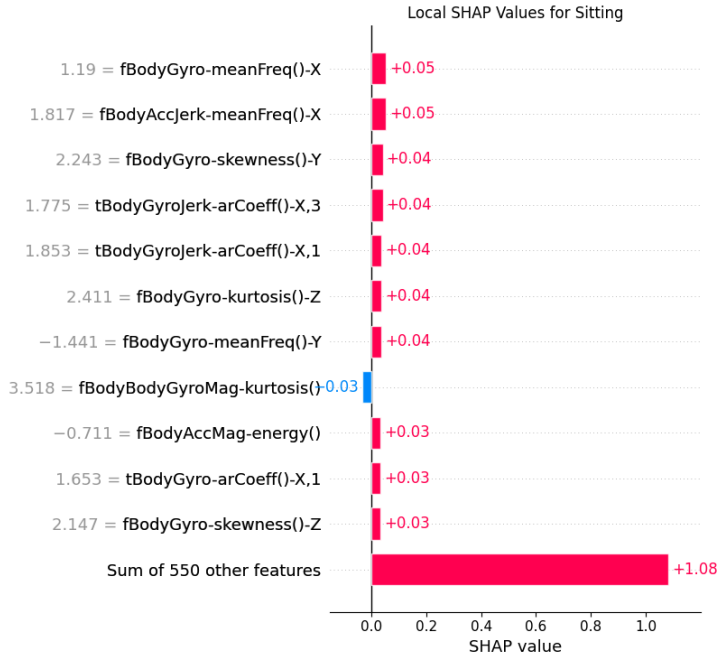


Figure 17. *Local SHAP values for the CNN model for the `Sitting` class.*

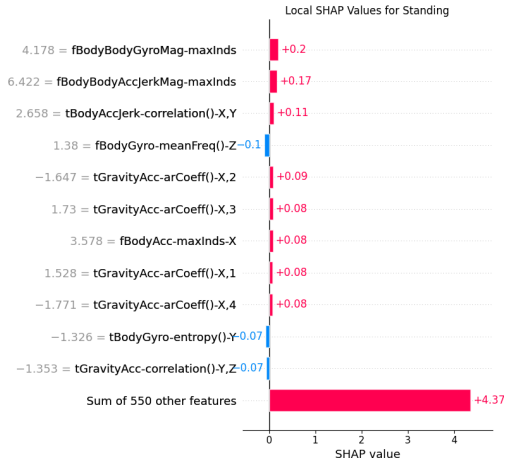


Figure 18. *Local SHAP values for the CNN model for the `Standing` class.*

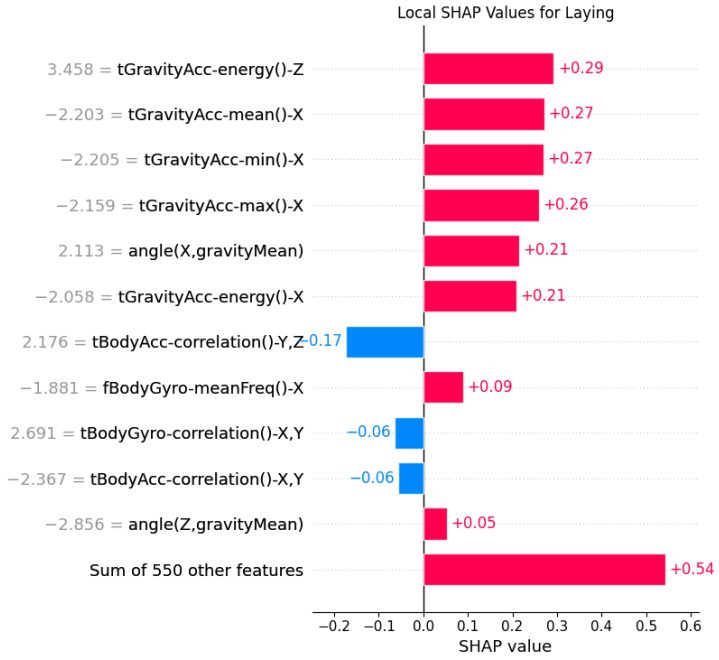


Figure 19. *Local SHAP values for the CNN model for the `Laying` class.*

5. Strengths and Limitations

SHAP is one of the prominent ways to explain ML model predictions. As with any other method, it has its own set of strengths and limitations.

5.1. Strengths

1. SHAP offers a consistent framework for interpreting model predictions, which has its roots from game theory. It is model agnostic and can be applied to any machine learning model, whether it is a simple linear regression model, a decision tree or a deep neural network.
2. SHAP can explain predictions for both global and local interpretability. Global explanations provide a high-level summary of the model's behavior, while local explanations provide insights into the factors influencing individual predictions.

3. The python SHAP library provides a number of plotting functions, e.g. beeswarm, force, waterfall, etc. that enable users to visualize the SHAP values in a number of ways. They communicate the insights effectively to both technical and non-technical stakeholders.

5.2. Limitations

1. Exact calculation of SHAP values becomes exponentially expensive with increasing feature counts. Specialized explainers e.g. TreeExplainer, GPUTreeExplainer [21], and DeepExplainer mitigate this but can still be resource-intensive for complex models or very large datasets.
2. SHAP values can be misleading when features are highly correlated, and may allocate credit arbitrarily among them. In such cases, the correct interpretation can be complicated, and may require domain expertise or additional feature engineering.
3. While SHAP values quantify the impact of features on model predictions, it is important to understand that they do not directly indicate causality. SHAP values measure statistical associations between features and predictions based on patterns learned from training data, but correlation does not imply causation. A feature may have high SHAP values due to confounding variables, spurious correlations, or because it serves as a proxy for the true causal factor. For example, in the bank marketing case, the `contact` method might show strong SHAP values not because the communication channel directly causes subscription decisions, but might be because it correlates with other unmeasured or hidden variables like the timing of the campaigns or other macroeconomic factors. Establishing causality requires additional considerations such as controlled experiments, temporal relationships, and domain expertise beyond what SHAP analysis alone can provide.

6. Conclusion

In this paper, we demonstrated SHAP's practical application to two important classes of models: Gradient Boosted Decision Trees (GBDTs) and Convolutional Neural Networks (CNNs). Our case studies highlighted how SHAP facilitates explanations for both structured and unstructured data on a global and local level. While SHAP effectively addresses many interpretability needs, users should be aware of challenges related to computational efficiency, correlated features, and the potential for over-interpretation.

Future work might focus on addressing these limitations, particularly through improved computational methods and strategies for handling feature correlations. Enhancing SHAP's usability further will contribute significantly to more transparent, reliable, and trusted machine learning applications in industry.

References

[1] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," *CoRR*, 2017, doi: https://doi.org/10.48550/arXiv.1705.07874.

[2] Moro and P. Cortez, "Bank Marketing." UCI Machine Learning Repository, 2014. doi: https://doi.org/10.24432/C5K306.

[3] Reyes-Ortiz and X. Parra, "Human Activity Recognition Using Smartphones." UCI Machine Learning Repository, 2013. doi: https://doi.org/10.24432/C54S4K.

[4] L. S. Shapley and others, "A value for n-person games," 1953, doi: https://doi.org/10.1515/9781400881970-018.

[5] C. Molnar, *Interpretable Machine Learning: A Guide for Making Black Box Models Explainable*, 3rd ed. 2025. [Online]. Available: https://christophm.github.io/interpretable-ml-book

[6] S. M. Lundberg, G. G. Erion, and S.-I. Lee, "Consistent Individualized Feature Attribution for Tree Ensembles," *CoRR*, 2018, doi: https://doi.org/10.48550/arXiv.1802.03888.

[7] S. M. Lundberg *et al.*, "From local explanations to global understanding with explainable AI for trees," *Nature Machine Intelligence*, vol. 2, no. 1, pp. 2522–5839, 2020, doi: https://doi.org/10.1038/s42256-019-0138-9.

[8] A. Shrikumar, P. Greenside, and A. Kundaje, "Learning Important Features Through Propagating Activation Differences," *CoRR*, 2017, doi: https://doi.org/10.48550/arXiv.1704.02685.

[9] J. H. Friedman, "Greedy function approximation: a gradient boosting machine," *Annals of statistics*, pp. 1189–1232, 2001, doi: https://doi.org/10.1214/aos/1013203451.

[10] T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016, pp. 785–794. doi: https://doi.org/10.1145/2939672.2939785.

[11] G. Ke *et al.*, "LightGBM: A Highly Efficient Gradient Boosting Decision Tree," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., 2017. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2017/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf

[12] S. Moro, P. Cortez, and P. Rita, "A Data-Driven Approach to Predict the Success of Bank Telemarketing," *Decision Support Systems*, vol. 62, pp. 22–31, 2014, doi: https://doi.org/10.1016/j.dss.2014.03.001.

[13] M. Kelly, R. Longjohn, and K. Nottingham, "The UCI Machine Learning Repository." [Online]. Available: https://archive.ics.uci.edu/

[14] The Pandas Development Team, "pandas-dev/pandas: Pandas." [Online]. Available: https://doi.org/10.5281/zenodo.3509134

[15] W. McKinney, "Data Structures for Statistical Computing in Python," in *Proceedings of the 9th Python in Science Conference*, Stéfan van der Walt and Jarrod Millman, Eds., 2010, pp. 56–61. doi: https://doi.org/10.25080/Majora-92bf1922-00a.

[16] F. Pedregosa *et al.*, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[17] L. Buitinck *et al.*, "API design for machine learning software: experiences from the scikit-learn project," in *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, 2013, pp. 108–122.

[18] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based Learning Applied to Document Recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998, doi: https://doi.org/10.1109/5.726791.

[19] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017, doi: https://doi.org/10.1145/3065386.

[20] A. Paszke *et al.*, "PyTorch: An Imperative Style, High-Performance Deep Learning Library." [Online]. Available: https://arxiv.org/abs/1912.01703

[21] R. Mitchell, E. Frank, and G. Holmes, "GPUTreeShap: Massively Parallel Exact Calculation of SHAP Scores for Tree Ensembles." [Online]. Available: https://arxiv.org/abs/2010.13972

Open Access | https://doi.org/10.25080/kcht7476

SciPy 2025

July 7 - July 13, 2025

Proceedings of the 24th Python in Science Conference
ISSN: 2575-9752

Explore Solvable and Unsolvable Equations with SymPy

Carl Kadie[1]

[1]FaST-LMM Open Source Genomics Project

Abstract

Why can we solve some equations easily while others seem impossible? This article explores how Python's free computer algebra system, SymPy, helps determine when a closed-form solution exists and when numerical methods are necessary. Instead of relying on advanced mathematics like Galois theory, we take a fun, hacker-style approach—letting SymPy do the work. We'll analyze different equation types, from polynomials to exponentials and trigonometric functions, using real-world examples like Kepler's equation. Whether you're a scientist, engineer, or Python enthusiast, this article will sharpen your intuition about solvability and show how to make the most of symbolic and numerical solutions in SymPy.

Keywords Mathematics, Data Science, Python, Science

```
# Optional: Uncomment this if running interactively
# %pip install numpy matplotlib sympy==1.13.3
```

1. Introduction

> Why can we solve some equations easily, while others seem impossible? And another thing: why is this knowledge hidden from us?

As scientists and engineers, we often create mathematical models. For example, consider the model: $y = x^2$. Given a value for x, we can apply it forward to compute y. For instance, if $x = 3$, then $y = 9$.

We can also apply this model backward. Starting with $y = x^2$, we rearrange to solve for x: $x = \pm\sqrt{y}$. If $y = 9$, then $x = \pm 3$. The expression $x = \pm\sqrt{y}$ is an example of a **closed-form solution** — an expression that uses a finite combination of standard operations and functions.

However, not all models are so straightforward. Sometimes, we encounter equations where we can't simply "solve for x" and get a closed-form expression. In such cases, we might hear, "That's not solvable — you need numerical methods." Numerical methods are powerful. They can provide precise approximations. Still, it frustrates me (and perhaps you) that no one ever seems to explain when closed-form solutions are possible and when they aren't.

The great Johannes Kepler shared our frustration. When studying planetary motion, he created this model:

$$y = x - c\sin(x) \tag{1}$$

This equation converts a body's position along its orbit (x) into its time along the orbit (y). Kepler sought a closed-form solution for x to turn time into a position. However, even 400 years later, the best we have are numerical methods.

In this article, we'll build *intuition* about when to expect a closed-form solution.

Published Jul 10, 2025

Correspondence to
Carl Kadie
carlk@msn.com

Open Access

Copyright © 2025 Kadie. This is an open-access article distributed under the terms of the Creative Commons Attribution 4.0 International license, which enables reusers to distribute, remix, adapt, and build upon the material in any medium or format, so long as attribution is given to the creator.

Rigorous alternative to intuition. The boundary between solvable and unsolvable equations is studied in advanced fields of mathematics. **Galois theory** shows, for example, why the *general* fifth-degree polynomial has no formula in radicals (the Abel–Ruffini theorem). **Transcendental number theory** helps explain why equations mixing algebraic and transcendental functions (like $x = e^x$) rarely yield elementary solutions. **Algebraic geometry** provides even broader tools for reasoning about solvability in higher dimensions. These areas are technically demanding and are not typically part of applied science training — which is exactly why they feel so opaque in practice. For readers curious to explore, the Wikipedia articles on Galois theory, Abel–Ruffini theorem, transcendental number theory, and Lambert (W) function give approachable entry points and references to deeper mathematics.

Instead of diving into these advanced fields, we'll cheat. Using **SymPy**, a Python-based computer algebra system, we'll explore different classes of equations to see which it can solve with a closed-form expression. For completeness, we'll also apply numerical methods.

We'll explore equations that combine polynomials, exponentials, logarithms, and trigonometric functions. Along the way, we'll discover specific combinations that often resist closed-form solutions. We'll see that if you want to create an equation **with** (or **without**) a closed-form solution, you should **avoid** (or **try**) the following:

- Fifth degree and higher polynomials
- Mixing x with $\exp(x)$ or $\log(x)$ — if Lambert's W function is off-limits
- Combining $\exp(x)$ and $\log(x)$ within the same equation
- Some pairs of trigonometric functions with commensurate frequencies
- Many pairs of trigonometric functions with non-commensurate frequencies
- Mixing trigonometric functions with x, $\exp(x)$, or $\log(x)$

Aside 1: I'm not a mathematician, and my SymPy scripts are not higher mathematics. If you find any mistakes or overlooked resources, forgive my oversight. Please share them with me, and I'll gladly add a note.

Aside 2: Welch Lab's recent video, *Kepler's Impossible Equation*, reminded me of my frustration about not knowing when an equation can be solved in a closed form. The video sparked the investigation that follows and provides our first example.

Aside 3: **Limitations:** The equations I explore here were chosen for their instructive qualities, not from any systematic distribution. There isn't really a "natural" way to sample equations at random, so these examples should be seen as exploratory case studies. The patterns I point out are illustrative, not statistical.

= Kepler's Equation

Imagine you are Johannes Kepler's research programmer. He has created the following model of orbital motion:

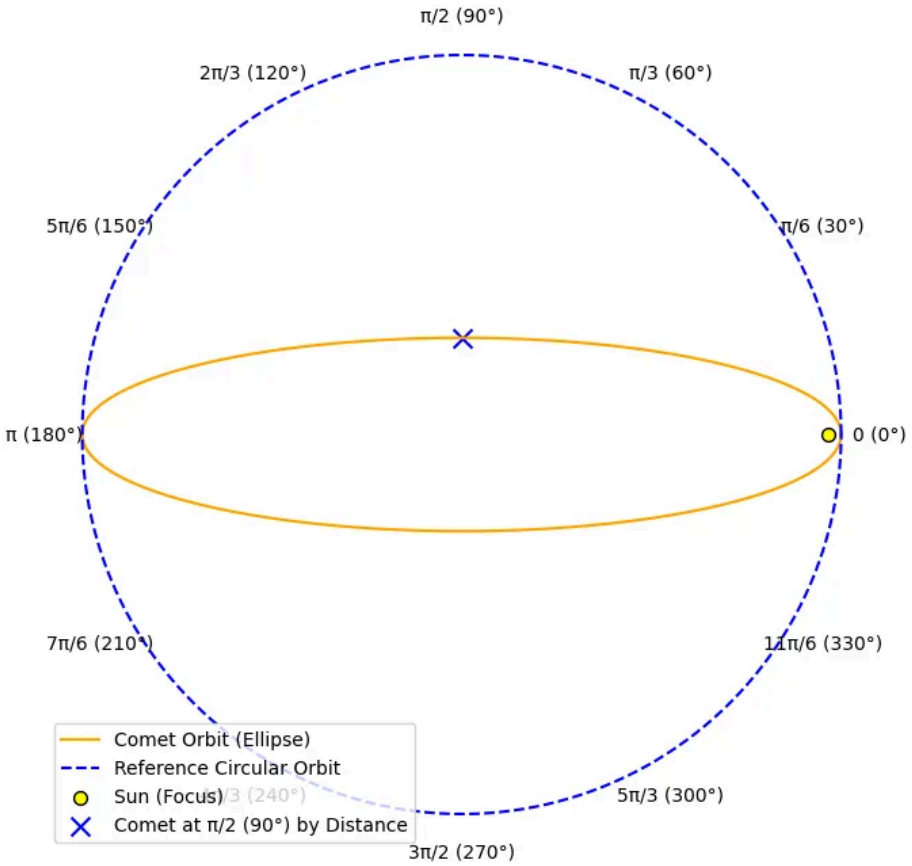
$$y = x - c\sin(x) \tag{2}$$

where:

- x is the body's position along its orbit. We measure this position as an angle (in radians). The angle starts at 0 radians when the body is closest to the Sun. When the body has covered $\frac{1}{4}$ of its orbit's distance, the angle is $\pi/2$ radians (90°). When it has covered half of its orbit's distance, the angle is π (180°), and so on. Recall that radians measure angles from 0 to 2π rather than from 0 to 360°.

- c is the orbit's eccentricity, ranging from 0 (a perfect circle) to just under 1 (a highly elongated ellipse). Suppose Kepler has observed a comet with $c = 0.967$.
- y is the body's time along its orbit. We measure this time as an angle (in radians). For instance, if the comet has an orbital period of 76 Earth years, then $\pi/2$ (90°) corresponds to $\frac{1}{4}$ of 76 years, or 19 years. A time of π (180°) corresponds to $\frac{1}{2}$ of 76 years, or 38 years. A time of 2π (360°) is the full 76-year orbital period.

This diagram shows the comet's position at $\pi/2$ radians (90°), which is $\frac{1}{4}$ of the way along its orbit:



Kepler asks for the time when the comet reaches position $\pi/2$ radians (90°). You create this Python code and look at the output:

```
import numpy as np

def kepler_equation(x):
    return x - c * np.sin(x)

c = 0.967
position_radians = np.pi / 2  # aka 90 degrees
time_radians = kepler_equation(position_radians)
orbital_period_earth_years = 76

t_earth_years = (time_radians / (2 * np.pi)) * orbital_period_earth_years
print(f"It takes approximately {t_earth_years:.2f} Earth years for the comet to move from 0 to π/2
radians.")
```

```
It takes approximately 7.30 Earth years for the comet to move from 0 to π/2 radians.
```

Aside: The comet covers 25% of its orbit distance in under 10% of its orbital period because it speeds up when closer to the Sun.

No good deed goes unpunished. Kepler, fascinated by the result, assigns you a new task: “Can you tell me how far along its orbit the comet is after 20 Earth years? I want to know the position in radians.”

“No problem,” you think. “I’ll just use a bit of high school algebra.”

First, you convert 20 Earth years into radians:

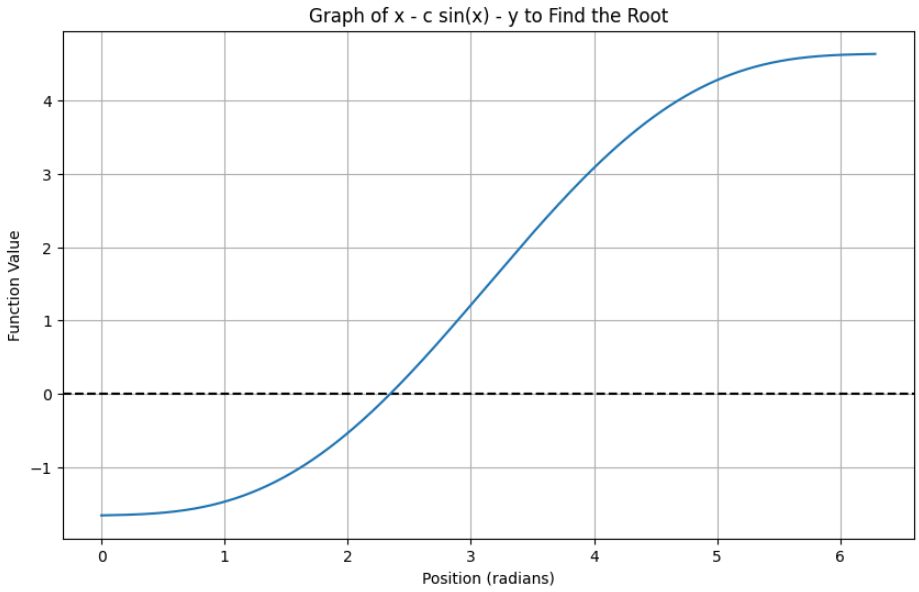
$$\text{time_radians} = \left(\frac{20}{76}\right) \cdot 2\pi = \frac{10}{19}\pi \tag{3}$$

Next, you rearrange Kepler’s equation, setting it equal to zero:

$$x - 0.967\sin(x) - \frac{10}{19}\pi = 0 \tag{4}$$

Now you want to find the value of x that makes this equation true. You decide to graph the equation to see where it crosses zero:

```
import numpy as np
import matplotlib.pyplot as plt

c = 0.967
time_earth_years = 20
orbital_period_earth_years = 76
time_radians = (time_earth_years / orbital_period_earth_years) * 2 * np.pi

def function_to_plot(x):
    return x - c * np.sin(x) - time_radians

x_vals = np.linspace(0, 2 * np.pi, 1000)
function_values = function_to_plot(x_vals)
plt.figure(figsize=(10, 6))
plt.axhline(0, color='black', linestyle='--')  # dashed horizontal line at y=0
plt.xlabel("Position (radians)")
plt.ylabel("Function Value")
plt.title("Graph of x - c sin(x) - y to Find the Root")
plt.grid(True)

plt.plot(x_vals, function_values)
plt.show()
```



So far, so good. The graph shows that a solution for x exists. But when you try to rearrange the equation to solve for x using algebra, you hit a wall. How do you isolate x when you have a combination of x and $\sin(x)$?

"That's okay," you think. "We've got Python, and Python has the SymPy package," a powerful and free computer algebra system.

You pose the problem to SymPy:

```
# Warning: This code will fail.
import sympy as sym
from sympy import pi, sin
from sympy.abc import x

c = 0.967
time_earth_years = 20
orbital_period_earth_years = 76

time_radians = (time_earth_years / orbital_period_earth_years) * 2 * pi
equation = x - c * sin(x) - time_radians

solution = sym.solve(equation, x)
# ^^^^^^^^^^^^^ error ^^^^^^^^^^^^^
print(solution)
```

Unfortunately, it replies with an error:

```
NotImplementedError: multiple generators [x, sin(x)]
No algorithms are implemented to solve equation x - 967*sin(x)/1000 - 10*pi/19
```

SymPy is quite good at solving equations, but not all equations can be solved in what's called *closed form* — a solution expressed in a finite number of elementary functions such as addition, multiplication, roots, exponentials, logarithms, and trigonometric functions. When we combine a term such as x with a trigonometric term like $\sin(x)$, isolating x can become fundamentally impossible. In other words, these types of mixed equations often lack a closed-form solution.

That's okay. From the graph, we know a solution exists. SymPy can get us arbitrarily close to that solution using numerical methods. We use SymPy's `nsolve()`:

```
import sympy as sym
from sympy import pi, sin
from sympy.abc import x

c = 0.967
time_earth_years = 20
orbital_period_earth_years = 76
time_radians = (time_earth_years / orbital_period_earth_years) * 2 * pi
equation = x - c * sin(x) - time_radians

initial_guess = 1.0   # Initial guess for the numerical solver
position_radians = sym.nsolve(equation, x, initial_guess)
print(f"After {time_earth_years} Earth years, the comet will travel {position_radians:.4f} radians
({position_radians * 180 / pi:.2f}°) along its orbit.")
```

```
After 20 Earth years, the comet will travel 2.3449 radians (134.35°) along its orbit.
```

We can summarize the results in a table:

Equation	SymPy Closed-Form (CF) Solution?	CF Solution(s)	A Numeric
$x - 0.967\sin(x) - \frac{10\pi}{19} = 0$	No, SymPy Limitation (NotImplementedError)		2.3449

Here SymPy raised a `NotImplementedError`. We classify this as **No, SymPy limitation (NotImplementedError)**. This means that SymPy did not produce a closed form, but this does not prove that no closed form exists. It may simply reflect a computational gap. We label the last column "A Numeric" to remind ourselves that it represents one numerical solution. There could be more.

In this section, we explored Kepler's equation and discovered the challenge of solving it in closed form. Python's SymPy package confirmed our struggle, and in the end, we had to rely on a numerical solution.

This gives us one example of an equation with no apparent closed-form solution. But is this typical? Are there classes of equations where we can always — or never — find a closed-form solution? Let's dig deeper by exploring another kind of equation: polynomials.

2. Polynomials

Polynomial equations such as $x^2 - x - 1 = 0$ are the reliable hammer of mathematical modeling — simple yet powerful. We all learn how to solve degree-two polynomials (those with x^2, "quadratic") in school.

500 years ago, during the Renaissance in Italy, solving polynomials of higher degrees became a form of public entertainment. Mathematicians like Tartaglia and Cardano competed for glory and recognition in public math duels.

These contests led to solutions for degree-three (cubic) and degree-four (quartic) polynomials. But what about degree five?

Let's use SymPy to investigate selected polynomials:

Equation	SymPy Closed-Form (CF) Solution?	CF Solution(s)	A Numeric
$x^2 - x - 1 = 0$	Yes, Elementary	$\left\{\frac{1}{2} - \frac{\sqrt{5}}{2}, \frac{1}{2} + \frac{\sqrt{5}}{2}\right\}$	-0.6180
$2x^3 + x + 1 = 0$	Yes, Elementary	$\left\{\frac{1}{2\left(-\frac{1}{2} - \frac{\sqrt{3}i}{2}\right)\sqrt[3]{\frac{27}{4} + \frac{3\sqrt{87}}{4}}} - \frac{\left(-\frac{1}{2} - \frac{\sqrt{3}i}{2}\right)\sqrt[3]{\frac{27}{4} + \frac{3\sqrt{87}}{4}}}{3}, \ldots\right\}$	0.2949 + 0.8723i
$x^5 + 1 = 0$	Yes, Elementary	$\left\{-1, \frac{1}{4} + \frac{\sqrt{5}}{4} + i\sqrt{\frac{5}{8} - \frac{\sqrt{5}}{8}}, \ldots\right\}$	-1.0000
$x^5 - x - 1 = 0$	No, SymPy uses CRootOf		1.1673

For polynomials up to degree four, we can always find closed-form elementary solutions. Specifically, these solutions require only a finite expression of basic arithmetic operations and roots (such as square roots or cube roots).

The number of solutions will never exceed the degree of the polynomial. However, some solutions may involve i, the square root of -1, which represents complex numbers. More on that in a moment.

And what about degree-five polynomials and beyond? Can we always find closed-form solutions?

The answer is mixed. Sometimes, we can. When a closed-form solution exists — for example, for $x^5 + 1 = 0$ — SymPy typically finds it.

However, in other cases, such as with $x^5 - x - 1 = 0$, SymPy cannot produce a closed-form elementary solution and instead represents the roots abstractly as `CRootOf`. This is not proof that no closed form exists; it only means that SymPy did not provide one. In fact, Évariste Galois showed that for general degree-five (and higher) polynomials there is no formula in radicals, though some special quintics can be solved. Our label "No, SymPy uses CRootOf" indicates that SymPy could not express the solution in elementary terms, but numerical approximations remain available.

To explore further, let's see exactly what SymPy does when given $x^5 - x - 1 = 0$:

```
import sympy as sym
from sympy.abc import x

equation = x**5 - x - 1
solution = sym.solve(equation, x)
print(solution)
```

```
[CRootOf(x**5 - x - 1, 0), CRootOf(x**5 - x - 1, 1), CRootOf(x**5 - x - 1, 2), CRootOf(x**5 - x - 1,
3), CRootOf(x**5 - x - 1, 4)]
```

Yikes! SymPy is clearly cheating here. It's saying, "Oh, you want a closed form? No problem! I'll just define a new, one-off function called `CRootOf(x**5 - x - 1, 0)` and call that the answer."

This is cheating because it doesn't answer the question of interest. SymPy is essentially giving a new name to an unsolved problem and claiming success.

SymPy, of course, has good reasons for producing its answer this way. For one thing, we can now easily find a numerical solution:

```
from sympy import N, CRootOf

print(N(CRootOf(x**5 - x - 1, 0)))
# Prints: 1.16730397826142
```

```
1.16730397826142
```

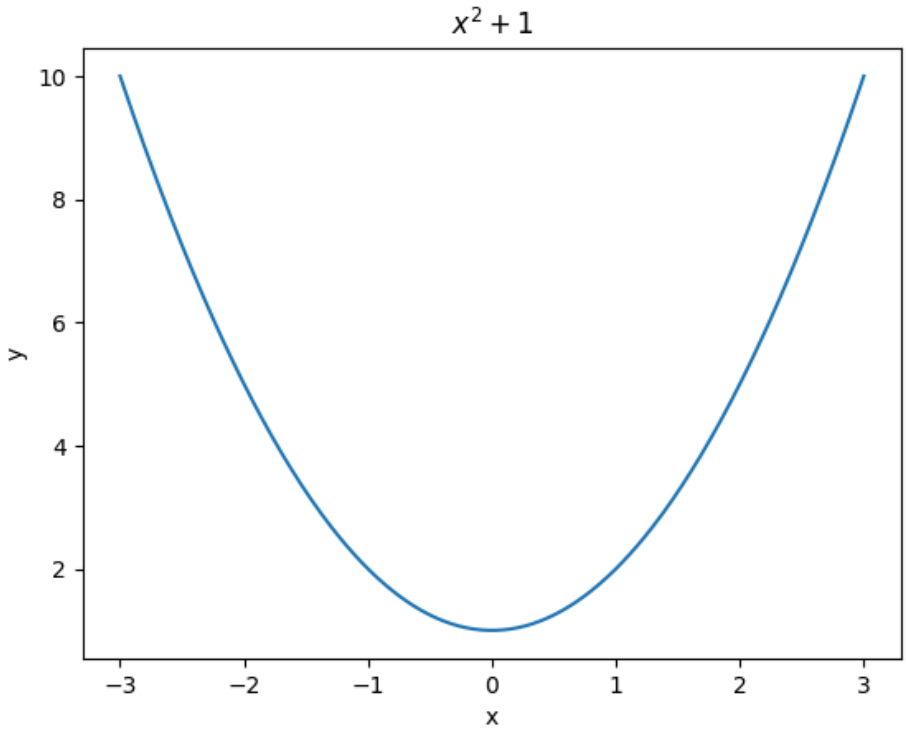
Solutions Even When No Real Solutions Exist: One surprising thing about polynomial equations is that you can always find solutions — at least numerically — even when no real solutions exist!

Consider this simple equation of degree two:

$$x^2 + 1 = 0 \tag{5}$$

If we plot this equation, it never crosses the x-axis, indicating no real solutions.

```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(-3, 3, 300)
plt.plot(x, x**2 + 1)
plt.title(r"$x^2 + 1$")
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```



However, using SymPy, we can find numerical solutions for any polynomial. For example:

```
from sympy import Eq, CRootOf, N, degree
from sympy.abc import x

equation = Eq(x**2 + 1, 0)
numerical_solution = [N(CRootOf(equation, d)) for d in range(degree(equation))]
print(numerical_solution)
# Prints: [-1.0*I, 1.0*I]
```

```
[-1.0*I, 1.0*I]
```

Notice that the solutions use i (the imaginary unit), meaning they are complex numbers. This is an illustration of the Fundamental Theorem of Algebra, which states that every (non-constant) polynomial equation has at least one complex solution — even when no real solutions exist.

The takeaway: unless complex numbers are meaningful in your domain, you should ignore complex solutions.

To summarize polynomials:

- **Degree four and below:** There is always a closed-form solution involving basic arithmetic operations and roots.
- **Degree five and above:** Generally, no closed-form solution exists using elementary operations, though some do exist, and SymPy may occasionally find them.
- **Solutions:** Polynomials will always have solutions — at least numerically — but these solutions may not be real (both mathematically and practically). You should typically ignore them unless complex numbers are meaningful in your domain.

Next, we'll add exponentials and logarithms to our equations. In the solutions, we discover the Lambert W function. Is it a `CRootOf`-like cheat?

3. Exp, Log and x

When we model data mathematically, we often use exponentials and logarithms. Below is a selection of what happens when we try to reverse such models by solving their equations with SymPy:

Equation	SymPy Closed-Form (CF) Solution?	CF Solution(s)	A Numeric
$xe^x = 0$	Yes, Elementary	0	0.0000
$xe^x + \frac{1}{10} = 0$	Yes, with LambertW	$\{W(-\frac{1}{10}), W_{-1}(-\frac{1}{10})\}$	-0.1118
$x + e^x = 0$	Yes, with LambertW	$-W(1)$	-0.5671
$x\log(x) + 1 = 0$	Yes, with LambertW	$e^{W(-1)}$	0.1684 + 0.7078i
$x + \log(x) = 0$	Yes, with LambertW	$W(1)$	0.5671
$x^2 + \log(x) = 0$	Yes, with LambertW	$e^{-\frac{W(2)}{2}}$	0.6529

Observations:

- **Sometimes you get lucky:** The first equation $xe^x = 0$ has an elementary solution: $x = 0$. While this isn't always the case, simple closed-form solutions can sometimes be found, even in equations involving exponentials or logarithms.
- **Every equation in this "family" appears to be solvable**, with two caveats: First, I can't precisely define this family and am unsure if a clear definition is possible. Second, solving these equations requires the Lambert W function, such as $W(1)$ and $W_{-1}(1/10)$. This function arises when x appears both inside and outside of an exponential or logarithmic expression.
- **If you don't accept W, you can't solve these functions in closed form:** Equations in this "family" generally have no closed-form *elementary* solutions without the Lambert W function.
- **We should accept W:** The Lambert W function is a well-defined, easily computable function with applications across math and science. Its late adoption relative to $\exp$, $\log$, $\sin$, and $\cos$ is simply historical.
- **A single W can generate multiple solutions:** Similar to how the square root function can produce two solutions, a W expression can yield zero, one, or two real solutions. When two real solutions exist, SymPy lists them separately, representing one as W (the principal branch) and the other as W_{-1} (the secondary branch). Beyond the real solutions, any W expression also generates an infinite number of complex solutions.

- **Complex solutions will arise:** Some equations, such as $x\log(x)+1=0$, lead only to complex solutions. As with polynomials, you should ignore complex numbers unless they are meaningful in your domain.
- **Degree-five and higher polynomials mixed with $\exp$ or $\log$ remain unsolvable:** Even with special functions like the Lambert W function, these equations cannot be solved in closed form using elementary functions.
- **What happens if we use both an exponential and a logarithm in the same equation?** When both an exponential and a logarithm appear together, SymPy typically cannot find a closed form — not even with the Lambert W function.

Equation	SymPy Closed-Form (CF) Solution?	CF Solution(s)	A Numeric
$e^x\log(x)+1=0$	No, SymPy Limitation (NotImplementedError)		0.5671
$e^x+\log(x)=0$	No, SymPy Limitation (NotImplementedError)		0.2699

To summarize, combining exponentials or logarithms with polynomials typically makes the equation unsolvable in elementary terms. If we allow the Lambert W function, equations with exponentials or logarithms (but not both) often admit closed forms. When both appear together, I do not believe a closed form exists — though that is based on intuition rather than proof, and SymPy's limitations alone do not settle the question.

We should embrace W as a valid tool for handling such cases.

Next, let's generalize Kepler's problem and see what happens when we introduce trigonometric functions into our equations.

4. Trigonometric Equations

Simple Trigonometric Equations: Here is our first batch of selected trigonometric equations:

Equation	SymPy Closed-Form (CF) Solution?	CF Solution(s)	A Numeric
$\sin(x)+\cos(x)=0$	Yes, Elementary	$-\frac{\pi}{4}$	-0.7854
$\sin(x)\cos(x)+1=0$	Yes, Elementary	$\left\{-2\operatorname{atan}\left(-\frac{1}{2}+\frac{\sqrt{2}\sqrt{1-\sqrt{3}i}}{2}+\frac{\sqrt{3}i}{2}\right),\ldots\right\}$	-0.7854 + -0.6585i
$\sin(x)+\sin(x+1)=0$	Yes, Elementary	$\left\{2\operatorname{atan}\left(\frac{1-\sqrt{\tan^2(\frac{1}{2})+1}}{\tan(\frac{1}{2})}\right),2\operatorname{atan}\left(\frac{1+\sqrt{\tan^2(\frac{1}{2})+1}}{\tan(\frac{1}{2})}\right)\right\}$	-0.5000
$\sin(x)\sin(x+1)+1=0$	Yes, Elementary	$\left\{-2\operatorname{atan}\left(\frac{-\tan(\frac{1}{2})+\sqrt{-1+\tan^4(\frac{1}{2})+2\tan^2(\frac{1}{2})-2i\sqrt{\tan^2(\frac{1}{2})+2\tan(\frac{1}{2})}}+i\sqrt{\tan^2(\frac{1}{2})+2}}{\tan^2(\frac{1}{2})+1}\right),.$	-0.5000 + -0.7921i

SymPy successfully finds closed-form elementary solutions for each equation. The solutions involve trigonometric functions, and in some cases, complex numbers appear. (Again, we typically ignore the complex solutions unless they are meaningful for the problem at hand.)

Keep in mind that sine and cosine are periodic, which leads to infinitely many solutions. The closed-form solutions that SymPy provides typically represent a single cycle.

Commensurate Frequency Equations: In the preceding equations, we limited the trigonometric function's input to $x+b$, where b is a constant. What happens if we allow inputs like a_1x+b_1 and a_2x+b_2 where both a_1 and a_2 are rational? This means the two periodic

functions may have different frequencies, but those frequencies can synchronize. We say the trigonometric functions have **commensurate frequencies**.

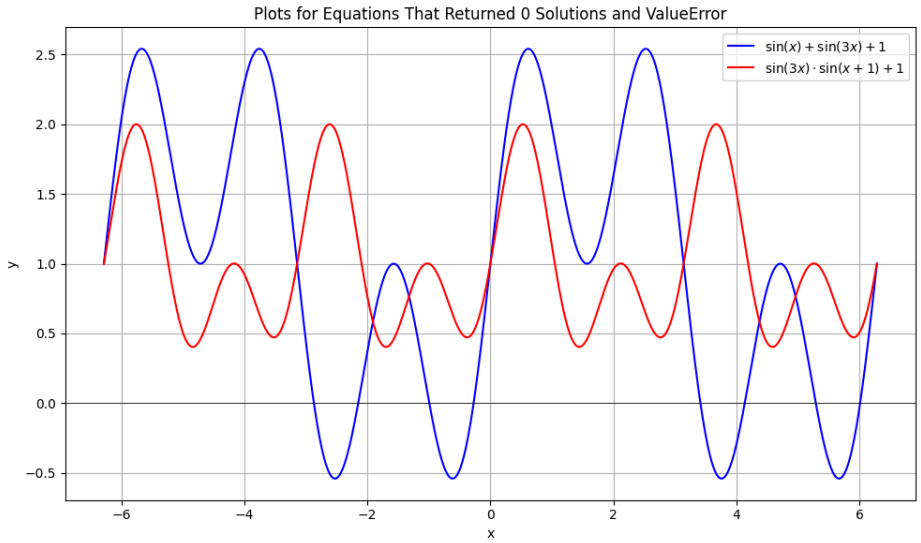
Equation	SymPy Closed-Form (CF) Solution?	CF Solution(s)	A Numeric
$\sin(x) + \sin(3x) = 0$	Yes, Elementary	$\{0, -\frac{\pi}{2}, \frac{\pi}{2}, \pi, ...\}$	0.0000
$\sin(x) + \sin(3x) + 1 = 0$	No, SymPy returns 0 Solutions		-0.2730
$\sin(x)\sin(3x) + 1 = 0$	Yes, Elementary	$\left\{-\frac{\pi}{2}, \frac{\pi}{2}, i\log\left(-\frac{1}{2} + \frac{\sqrt{5}}{2}\right), i\log\left(\frac{1}{2} + \frac{\sqrt{5}}{2}\right), ...\right\}$	-1.5708
$\sin(3x) + \sin(x + 1) = 0$	No, SymPy Limitation (Timeout)		1.3208
$\sin(3x)\sin(x + 1) + 1 = 0$	No, SymPy Limitation (Timeout)		ValueError

Observations:

- We occasionally get a closed-form elementary solution.
- For $\sin(x) + \sin(3x) + 1 = 0$, SymPy returns *zero solutions*. However, plots and numerical methods suggest solutions exist. Moreover, when input into WolframAlpha (an online computer algebra system), it produces hybrid solutions: combinations of elementary functions with `CRootOf` expressions of degree six. As discussed in the polynomial section, such expressions generally lack a closed-form solution.
- SymPy sometimes times out looking for a closed-form solution even when numerical methods succeed.
- In other cases, it times out, and both numerical methods and plots confirm there are no solutions. (Previously, we might have received a complex-number result. WolframAlpha does sometimes return such complex numeric results.)

Let's plot the equation that returned zero closed-form solutions. Let's also plot the one that numerically returned `ValueError`:

```
x = sym.symbols('x')
f1 = sym.lambdify(x, sym.sin(x) + sym.sin(3*x) + 1, "numpy")
f2 = sym.lambdify(x, sym.sin(3*x)*sym.sin(x + 1) + 1, "numpy")
xv = np.linspace(-2*np.pi, 2*np.pi, 1000)
plt.figure(figsize=(10, 6))
plt.plot(xv, f1(xv), label=r'$\sin(x) + \sin(3x) + 1$', color='blue')
plt.plot(xv, f2(xv), label=r'$\sin(3x) \cdot \sin(x + 1) + 1$', color='red')
plt.axhline(0, color='black', lw=0.5)
plt.title("Plots for Equations That Returned 0 Solutions and ValueError")
plt.xlabel("x"), plt.ylabel("y"), plt.legend(), plt.grid(True), plt.tight_layout()
plt.show()
```



Additional Observations:

- From the **blue plot**, SymPy's response of "no solutions" appears to be a **bug**. There are clearly solutions visible, and SymPy should either have found them or thrown an exception.
- On the other hand, in the **red plot**, the numerical result of `ValueError` is accurate — there are no solutions.

For all the trigonometric equations we've encountered so far, SymPy seems to find real-valued closed-form solutions when they exist. When they don't exist, it times out or gives unpredictable errors.

Non-Commensurate Frequency Equations: In the preceding equations, we allowed trigonometric functions with inputs of the form $ax + b$, where a is rational. Now we ask: what if we allow inputs like $a_1x + b_1$ and $a_2x + b_2$ where a_1 is rational but a_2 is irrational? This means the two periodic functions will never synchronize. We say they have **non-commensurate frequencies**.

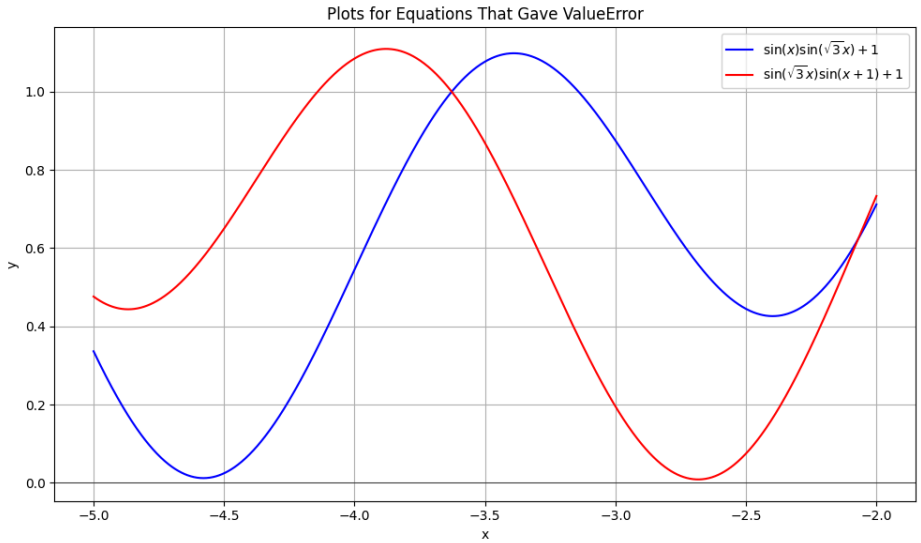
Equation	SymPy Closed-Form (CF) Solution?	CF Solution(s)	A Numeric
$\sin(x) + \sin\left(\sqrt{3}x\right) = 0$	Yes, Elementary	$\left\{0, -\frac{\pi\left(1+\sqrt{3}\right)}{2}\right\}$	0.0000
$\sin(x) + \sin\left(\sqrt{3}x\right) + 1 = 0$	No, SymPy Limitation (NotImplementedError)		-0.3876
$\sin\left(\sqrt{3}x\right) + \sin(x + 1) = 0$	No, SymPy Limitation (PolynomialDivisionFailed)		1.9338

Equation	SymPy Closed-Form (CF) Solution?	CF Solution(s)	A Numeric
$\sin(x)\sin\left(\sqrt{3}x\right)+1=0$	No, SymPy Limitation (NotImplementedError)		ValueError
$\sin\left(\sqrt{3}x\right)\sin(x+1)+1=0$	No, SymPy Limitation (NotImplementedError)		ValueError

Observations:

- Equations with two trigonometric functions having non-commensurate frequencies generally seem **unsolvable** in closed form. When no elementary solution is available, SymPy returns `NotImplementedError`.
- We can still get lucky and occasionally find a closed-form solution. In one such case, SymPy returned `PolynomialDivisionFailed`, but WolframAlpha found a closed-form expression.
- When an equation has **no solutions**, SymPy produces a `ValueError`, which we can confirm via plots. In these cases, we did not see complex-number results.

```
x = sym.symbols('x')
f1 = sym.lambdify(x, sym.sin(x)*sym.sin(sym.sqrt(3)*x) + 1, "numpy")
f2 = sym.lambdify(x, sym.sin(sym.sqrt(3)*x)*sym.sin(x + 1) + 1, "numpy")
xv = np.linspace(-5, -2, 1000)
plt.figure(figsize=(10, 6))
plt.plot(xv, f1(xv), label=r'$\sin(x)\sin(\sqrt{3}x)+1$', color='blue')
plt.plot(xv, f2(xv), label=r'$\sin(\sqrt{3}x)\sin(x+1)+1$', color='red')
plt.axhline(0, color='black', lw=0.5)
plt.title("Plots for Equations That Gave ValueError"), plt.xlabel("x"), plt.ylabel("y")
plt.legend(), plt.grid(True), plt.tight_layout()
plt.show()
```



The equations do not quite touch zero, so no solutions

Our conclusion regarding trigonometric equations is that we can often find elementary closed-form solutions.The main exception seems to be when the **frequencies are non-commensurate** — for example, in an equation containing $\sin(x)$ and $\sin\left(\sqrt{3}\,x\right)$.

The final question we'll explore is what happens when we mix trigonometric functions with exponentials and logarithms.

5. Trigonometric and x, exp, log

Our final set of selections will require only a short discussion. What if we run selected of equations through SymPy, each equation containing one trigonometric function combined with either x, $\exp(x)$, or $\log(x)$?

Equation	SymPy Closed-Form (CF) Solution?	CF Solution(s)	A Numeric
$x + \sin(x) = 0$	No, SymPy Limitation (NotImplementedError)		0.0000
$x \sin(x) + 1 = 0$	No, SymPy Limitation (NotImplementedError)		3.4368
$\log(x) + \sin(x) = 0$	No, SymPy Limitation (NotImplementedError)		0.5787
$e^x + \sin(x) = 0$	No, SymPy Limitation (NotImplementedError)		-0.5885
$x^3 + \cos(x) = 0$	No, SymPy Limitation (NotImplementedError)		-0.8655

The results are unanimous: SymPy is unable to produce closed-form solutions for any of these combinations. However, it seems that SymPy should have produced $x = 0$ as the closed-form solution for the first equation, as indeed WolframAlpha does.

6. Conclusion

So, there you have it — an exploration of which equations tend to lack closed-form solutions. If you're interested in experimenting with the examples in this article, you can find my Python code on GitHub.

Here is a summary of SymPy behaviors:

Behavior	Meaning & Next Step
`CRootOf` returned	No closed form; use as-is or switch to numeric.
`NotImplementedError`	Solver not coded; try another computer algebra system or numeric.
`PolynomialDivisionFailed`	Algebraic step broke; try another system or numeric.
Timeout	Solver stuck; abandon analytic, use numeric or another system.
0 solutions (empty set)	Either true or a solver miss; check elsewhere before concluding.
Numeric `ValueError`	Numeric solver failed; try new numeric guess.

Rule of thumb:
Only `CRootOf` is a final answer. All others are failures — switch to numeric or try another computer algebra system.

As I worked through these selected equations, here is what surprised me:

- **Kepler's Equation** is wonderfully simple. I didn't know one could model an ellipse — a geometric shape I find complicated — with such elegance.
- **Lambert's W function** proved to be invaluable for handling equations that mix terms like x and $\exp(x)$. We should consider it an elementary function.
- **SymPy** is an excellent, free tool that handles symbolic algebra and trigonometric equations far better than many of us could handle manually. While it may not match WolframAlpha in some cases, it's incredibly versatile and accessible.
- **Mixing trigonometric functions** with other terms frequently prevents closed-form solutions, especially when frequencies are non-commensurate.
- **When closed-form solutions remain out of reach**, plotting and numerical methods step in, delivering practical results.

Thank you for joining me on this journey. I hope you now have a clearer understanding of when you can use equation-solving techniques to reverse models and how much SymPy can assist. Also, when an equation resists a closed-form solution, you can now understand why and when to rely on numerical methods.

Aside: If you enjoyed exploring mathematics with Python and SymPy, you may also enjoy using them to explore **Newtonian physics**. Please see this Towards Data Science article and the related, popular PyData conference talk.

Open Access | https://doi.org/10.25080/vxkf4244

SciPy 2025

July 7 - July 13, 2025

Proceedings of the 24th Python in Science Conference
ISSN: 2575-9752

Ocetrac: An Object-based Framework for Tracking and Quantifying Climate Events in Gridded Datasets

Cassia Cai[1], **Jacob T. Cohen**[1], **Hillary Scannell**[2], **Valentina Staneva**[3], and **LuAnne Thompson**[1]

[1]School of Oceanography, University of Washington, [2]Jupiter Intelligence, [3]eScience Institute, University of Washington

Abstract

Climate extremes such as marine heatwaves and ocean acidification events appear as dynamic, irregular structures in gridded datasets. Software tools are available for detecting these events. However, there is a need for standardized methods for tracking their spatiotemporal evolution and evolving characteristics. Ocetrac is an open-source Python package that bridges this gap by integrating morphological image processing (using `scikit-image`) with `xarray`-based data workflows to track and quantify amorphous two-dimensional features in space and time. The tracking algorithm identifies and tracks objects that split and merge. The Measures submodule computes shape-, motion-, temporal-, and intensity-based measures. These measures enable comparison between similar events that occur at different times (e.g., via clustering), machine learning integration such as feature extraction for classification/regression (e.g., `scikit-learn`), and process-oriented analysis such as linking event behavior to physical drivers (e.g., surface heat fluxes). Designed for the Scientific Python ecosystem, Ocetrac simplifies workflows from detection to downstream analysis, supporting applications in climate science, oceanography, and atmospheric research.

Keywords spatiotemporal tracking, morphological operations, image processing, scientific Python, climate extremes

1. Introduction

Gridded climate datasets are important for studying phenomena like marine heatwaves (MHWs), deoxygenation zones, and atmospheric blocking events among others. These events exhibit complex spatiotemporal behavior including splitting, merging, and complex evolution. The ecological impact of these events is closely tied to their spatial and temporal characteristics. For example, in the case of MHWs, long-lived and spatially extensive events can have significant ecological impacts [1]. These impacts include habitat destruction [2] and shifts in marine species distributions [3]. Understanding these events requires not only detecting their occurrence but also tracking their evolution and quantifying their characteristics over time.

Conventional analysis approaches, including gridpoint-based statistics, fixed-region summaries, and empirical orthogonal functions (EOFs), offer valuable insight in explaining spatial patterns but fall short in characterizing events as coherent and evolving structures. Gridpoint statistical analysis obscures spatial connectivity while EOFs, which is a dimensionality reduction technique that decomposes spatiotemporal fields into orthogonal spatial patterns and their time-varying amplitudes, lack object-level granularity. Object tracking is a different approach and has been applied to both atmospheric and oceanic phenomena.

Published Jul 10, 2025

Correspondence to
Cassia Cai
fmc2855@uw.edu

Open Access

Copyright © 2025 Cai *et al.*. This is an open-access article distributed under the terms of the Creative Commons Attribution 4.0 International license, which enables reusers to distribute, remix, adapt, and build upon the material in any medium or format, so long as attribution is given to the creator.

Object-based approaches treat climate features as coherent objects in space (x, y) and time (t), thus enabling direct evaluation of properties like shape, connectivity, and persistence. For example, D. J. Gagne, A. McGovern, S. E. Haupt, R. A. Sobash, J. K. Williams, and M. Xue [4] used an enhanced watershed method to identify hailstorm objects in radar data. Similarly, `TempestExtremes` described in P. A. Ullrich, C. M. Zarzycki, E. E. McClenny, M. C. Pinheiro, A. M. Stansfield, and K. A. Reed [5] applies a MapReduce-style algorithm to detect and track compact, threshold-defined features such as tropical and extratropical cyclones. While effective for well-defined structures, such threshold-based methods can struggle to capture the extent and temporal evolution of more diffuse or amorphous events.

Object-based detection and tracking of spatiotemporal climate features is a growing area of research. Other climate features, such as tropical cyclones and atmospheric rivers, have benefitted from tracking algorithms [5]. Recent studies have used a variety of techniques to identify MHW events, each with its own benefits and limitations. All techniques track objects using a form of connected component analysis: an MHW grid cell is connected with adjacent MHW grid cells in space and time. Algorithms differ in how they assign connectivity, though. Some studies, such as G. Bonino, S. Masina, G. Galimberti, and M. Moretti [6] and R. I. Woolway, E. J. Anderson, and C. Albergel [7], connect grid cells only if they are directly adjacent to each other. D. Sun, Z. Jing, F. Li, and L. Wu [8] introduced a more flexible approach, using a Nearest Neighbor algorithm to connect nearby grid cells that are not necessarily adjacent to one another. Additional constraints exist in deciding what constitutes overlap for temporal tracking. D. Sun, F. Li, Z. Jing, and et al. [9] extended their tracking methodology to analyze MHW volumes. The TOOCAN algorithm [10], designed for organized deep convective systems, also uses a connected components approach with adjustable spatiotemporal constraints. Ocetrac, like D. Sun, Z. Jing, F. Li, and L. Wu [8], addresses spatial coherence through smoothing (via morphological operations rather than K-Nearest Neighbor filtering). D. Sun, Z. Jing, F. Li, and L. Wu [8] uses a temporal overlap parameter to determine whether two objects are the same event while Ocetrac connects objects that overlap by at least one grid cell in space and time.

Generally, existing tools tend to fall into two categories: specialized trackers designed for specific phenomena (e.g., tropical cyclones) that use rigid assumptions, or pixel-level anomaly detectors that lack spatial integration. Specialized trackers are often tailored to particular variables or regions, while pixel-level anomaly detectors can produce disjointed results that require manual post processing to reconstruct event trajectories. Moreover, many studies rely on custom scripts that are often not well-documented, making comparisons and reproducibility challenging. Despite their advantages, object-based techniques remain underutilized in many geophysical applications due to gaps in software accessibility and generality.

This paper describes Ocetrac, an open-source Python package designed for detecting, tracking, and quantifying irregular, evolving features in gridded climate data. Ocetrac combines morphological image processing using `scikit-image` [11] with labelled array workflows using `xarray` [12] to characterize irregular structures. Although Ocetrac uses an initial threshold to identify candidate regions, Ocetrac moves beyond a simple binary definition by using morphological operations to filter and refine these regions based on their spatial coherence and connectivity. The Measures submodule provides functionality to quantify spatial and temporal properties of the tracked events, and thus facilitates further statistical analysis, such as aggregation analysis of multiple events within large datasets. Its modular design supports:

1. Detection: Ocetrac smooths spatially connected regions (features) in a binary field, such as sea surface temperature (SST) anomalies, to identify objects that exceed a user-defined threshold (e.g., the 90th percentile of anomalies)

2. Tracking: Ocetrac labels objects across time steps, even when they split or merge
3. Quantification: Ocetrac computes shape, motion, and intensity measures
4. Integration: Ocetrac is compatible with the Scientific Python ecosystem, including `dask` [13], `xarray` [12] and `scikit-learn` [14]

Although originally developed for MHW analysis, Ocetrac can be used for other applications, from atmospheric rivers to hypoxic zones and bloom patches in the ocean. Its modular design allows users to customize detection thresholds, tracking logic, and measures of tracked objects. This paper details its design, demonstrates several use cases, and outlines future directions.

2. Goals and Motivations

Climate phenomena like MHWs are dynamic events that can propagate across basins, changing their size, shape, and intensity as they evolve over time. Traditional gridpoint or static regional analyses that evaluate variables at individual locations or within a fixed region may overlook the structure and the evolution of such events. Object-based methods address this limitation by treating features as one event. This approach shifts the focus from isolated grid cells to the geometry, movement, and intensity evolution of entire events.

Building on the need for accessible and general object-based tracking, Ocetrac, unlike phenomenon-specific tools, operates on intensity data that is thresholded (e.g., SST anomalies exceeding the 90th percentile), applying morphological operations to group connected regions into labeled objects before tracking their evolution. The design prioritizes interoperability with standard climate data formats (CF-compliant files following Climate and Forecast metadata conventions) and is compatible with observational products, reanalyses, and model output. For MHW-specific applications, Ocetrac is compatible with output from packages such as marineHeatWaves and xmhw, which allows for existing anomaly detection workflows to be combined with Ocetrac's object-based tracking. Because the tracking logic is decoupled from domain-specific assumptions, Ocetrac is a practical and adaptable tool for tracking climate features across datasets and can be used to answer diverse research questions. The Measures submodule processes tracked objects to generate quantitative diagnostics, which allows for standardized comparison across events and datasets.

3. Ocetrac: Tracking and Measures Submodules

Ocetrac is a flexible tracking framework designed for geophysical feature analysis. While we demonstrate its application using SST anomalies to identify MHWs, the algorithm itself is variable-agnostic. Ocetrac can be applied to any two-dimensional spatiotemporal field that can be thresholded into spatially coherent features and has sufficient temporal resolution to resolve event evolution. The tracking methodology is described briefly below and in detail in H. A. Scannell, C. Cai, L. Thompson, D. B. Whitt, D. J. Gagne, and R. P. Abernathey [15] where it was used to examine MHWs throughout the globe. To support event characterization beyond simple tracking, we introduce the Measures submodule, which enables the calculation of a suite of geometric, intensity-based, and motion-related diagnostics for each tracked event. The Ocetrac workflow is demonstrated below in Figure 1. The tutorial associated with this workflow is available here.

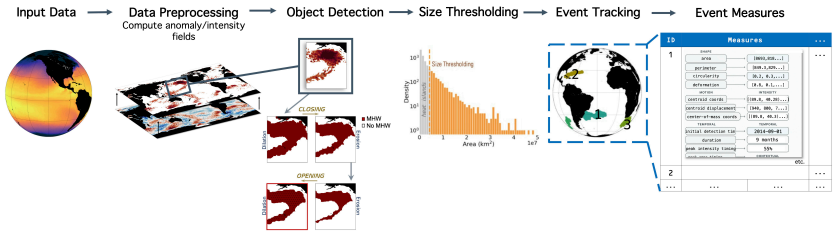


Figure 1. *Ocetrac workflow diagram*

3.1. Input Specifications and Preprocessing

Ocetrac requires input data as a 3D `xarray.DataArray` with dimensions ordered as (x, y, t), where (x, y) correspond to regular spatial coordinates, typically latitude and longitude, and t corresponds to time. The time dimension should be uniformly spaced to ensure optimal tracking performance of features through the temporal sequence. The time dimension should also have high enough spatial and temporal resolution to capture the temporal evolution of the target feature. Temporal gaps in the data can be filled using linear temporal interpolation to maintain continuity in the time series.

An optional binary land mask (1 for valid grid cells, 0 for excluded regions such as land or sea ice) can be provided to omit specific areas from detection and tracking. All preprocessing, such as detrending and anomaly calculation, thresholding, must be performed before using Ocetrac. Common thresholding approaches include: percentile-based (e.g., values exceeding the 90th percentile of anomalies), absolute value thresholds (e.g., values exceeding 28°C), or statistical significance thresholds (e.g., values exceeding two standard deviations from the mean). Ocetrac is agnostic to the thresholding method, as long as the input is a binary spatiotemporal field.

3.2. Tracking Submodule

The tracking process begins with converting anomalies into binary maps using a threshold (typically the 90th percentile for MHWs). Grid cells exceeding the threshold are marked as features where active grid cells are assigned the value of 1 and all others are inactive and assigned the value of 0.

To refine features, Ocetrac applies sequential morphological operations:

1. Closing (dilation followed by erosion): Fills small holes within features and connects nearby regions belonging to the same object. This operation helps maintain spatial coherence by ensuring that nearby features that should logically be part of the same event are connected.
2. Opening (erosion followed by dilation): Removes isolated pixels, smooths feature boundaries, and cleans residual artifacts introduced by closing. This operation helps eliminate noise and small-scale features that may not be physically meaningful.

The choice of these morphological operations follows image processing practices for feature enhancement and noise reduction. The closing operation is applied first to connect nearby features, followed by opening to remove small artifacts and refine the boundaries of the features. This sequence is important for maintaining the integrity of the detected features while minimizing fragmentation. The morphological operations are implemented using `scikit-image` [11] and are designed to be computationally efficient, allowing for processing large datasets in parallel using `dask` [13]. While other sequences of operations (e.g., opening followed by closing) can be implemented, empirically, this sequence (closing followed by opening) optimizes feature integrity while minimizing fragmentation. The intermediate

results of morphological operations are shown and described in greater detail in H. A. Scannell, C. Cai, L. Thompson, D. B. Whitt, D. J. Gagne, and R. P. Abernathey [15].

The operations are performed using a structuring element, which defines the neighborhood around each pixel used to determine its new value. Ocetrac uses a circular structuring element, which is a disk-shaped kernel that treats all directions equally. The size of this disk is defined by its radius R (in number of grid cells). The radius is a parameter that controls the spatial scale of the filtering: a larger radius results in more aggressive smoothing and merging of features, while a smaller radius preserves finer details but may retain more noise.

For 0.25° resolution data:

- R = 4 to 6 grid cells (1 - 1.5°): Preserves smaller-scale features while removing noise
- R = 6 to 8 grid cells (1.5 to 2°): Emphasizes larger, more coherent structures
- R > 8 grid cells: May merge distinct features or even fail to identify features as valid objects altogether

For data at a different resolution, the radius must be scaled to maintain a consistent physical filtering scale. For example, for higher-resolution 0.125° data, the grid cell size is halved. To achieve an equivalent physical filter size, R should be doubled.

The choice of R should be informed by the typical spatial scale of the features of interest and the resolution of the input data and represents a trade-off between feature completeness (higher R retains more connected areas) and spatial precision (lower R preserves finer-scale details) (Figure 2). The optimal radius depends on the spatial resolution of the input data and the typical size of the features being tracked. Users should experiment with different radius values to find the best balance between feature retention and noise reduction for their specific application. Users should also validate this parameter against known feature scales in their domain. Features, after undergoing morphological operations, are now referred to as objects.

Ocetrac then applies size-based filtering to focus on the most spatially coherent objects. It calculates the area of each detected object in grid cells, then removes objects smaller than a user-defined area threshold P. By default, it keeps objects larger than 75% of all detected features (the 75th percentile) for the full dataset, meaning the smallest 25% of all objects are filtered out. This filtering step helps eliminate noise and isolate the most physically meaningful objects in the dataset.

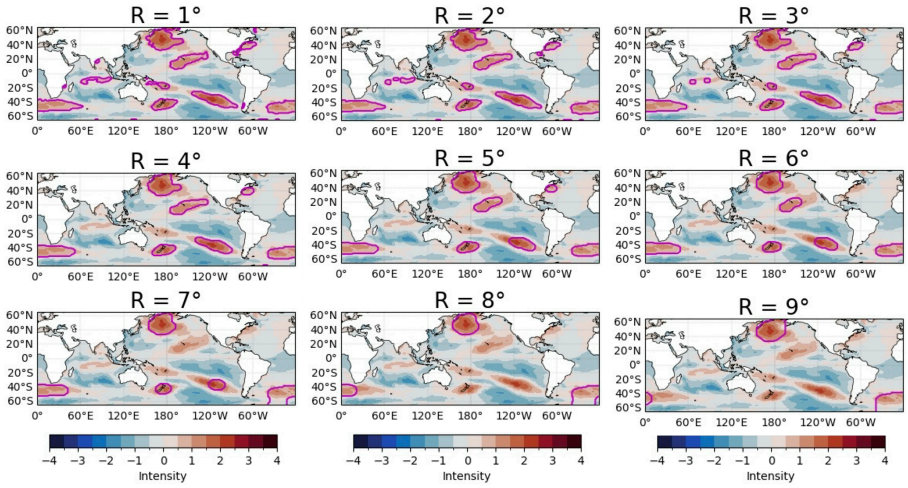


Figure 2. *Objects outlined (purple outlines) detected using varying R values (R ranging from 1° to 7°). The background shows SST anomalies (°C) from February 2007 of an ensemble member of the CESM2-LENS dataset. The horizontal resolution is 1°.*

Ocetrac identifies objects in each timestep by grouping together adjacent pixels, including diagonal connections (8-connectivity). This is a permissive approach for the initial detection phase to ensure that complex, irregularly shaped features are identified as single, coherent objects rather than being artifically fragmented into multiple parts. Then, to link objects across consecutive timesteps, Ocetrac uses a more conservative 6-connectivity approach where it matches objects that appear in either the exact same grid cell or directly adjacent cells (up/down/left/right) in the next timestep. This conservative approach allows objects to move gradually and prevents discontinuous jumps. The system maintains consistent ID numbers for objects as they evolve and handles global datasets where the map wraps around the globe at 0°/360° longitude, and tracks when objects merge together or split apart. The only requirement is that objects must overlap by at least one grid cell from one time step to the next to be in the same event.

The current implementation of Ocetrac is designed for rectilinear (latitude-longitude) grids. A limitation of this framework is its handling of high-latitude regions. At high latitudes, the convergence of meridians leads to distortions in area representation, which can affect the accuracy of object detection and tracking. Future versions of Ocetrac will address this limitation by incorporating grid-aware algorithms that adjust for varying grid cell sizes and shapes at different latitudes.

Implementation example

```
# Initialize tracker with user-defined parameter
obj_Tracker = ocetrac.Tracker(
    binary_out_afterlandmask,   # Thresholded binary field
    newmask,                    # Land/ice mask (optional)
    radius=3,                   # R in grid cells
    min_size_quartile= 0.75,    # P (determines minimum object size)
    timedim = 'time',           # Time dimension name
    xdim = 'lon',               # Longitude dimension name
    ydim='lat',                 # Latitude dimension name
    positive=True)              # Track positive anomalies

# Execute tracking and morphological operations
blobs = obj_Tracker.track()

mo = obj_Tracker._morphological_operations()
```

Here Ocetrac provides basic diagnostics (e.g., initial/final object counts, area retention percentage) to evaluate tracking performance and feature characteristics.

3.3. Measures Submodule

The output of the tracker is a `xarray.Dataset` with many labelled events. The Measures submodule allows users to process large numbers of tracked events by automating repetitive calculations. Its standardized outputs allow for direct comparison across different events and datasets, which streamlines analysis workflows. The submodule supports both individual case studies and ensemble analyses. In brief, the Measures submodule processes the output of Ocetrac's tracking algorithm and provides quantitative metrics to characterize detected events. These measures fall into five categories: shape, motion, intensity, and temporal, and contextual and are summarized in Table 1.

Shape measures characterize geometric properties including area, perimeter, and circularity that are relevant for quantifying structural evolution. Shape measures can be used for understanding ecological impacts, for example, the area/extent can be used for assessing po-

tential habitat affected, while the deformation quantifies shape changes between timesteps. These measures can help answer questions such as (1) How does event shape affect local ecosystems? and (2) Do certain shapes persist or evolve predictably? Motion measures track movement and displacement patterns via centroids and intensity-weighted centers of mass, while also handling longitudinal wrapping. Motion measures can help identify propagation directions and whether events remain stationary. Intensity measures capture magnitude variations through spatial statistics (mean, maximum, percentile extremes) of the underlying anomaly field. Intensity measures can be used for assessing thermal stress on ecosystems. Temporal measures record lifecycle characteristics including duration, initiation timing, and timing peaks of intensity and area. Temporal measures can determine event development stages, seasonal dependencies, and potential climate connections. Some of the measures can also be used to create contextual measures (e.g., object counts per timestep) which capture splitting and merging objects. Together, these measures enable analysis of event dynamics.

Table 1. *Event measures in the Measures submodule*

Category	Measure	Definition	Interpretation
SHAPE	Area (Extent)	Spatial coverage in km² or grid cells. Calculated as the sum of cell areas within bounds.	Larger values = greater spatial influence
	Perimeter	Boundary length (km), computed via geodesic contours. Calculated as Haversine sum of contour points.	Complex shapes yield high perimeters.
	Circularity	Deviations from a perfect circle. Calculated as $4\pi \frac{Area}{Perimeter^2}$.	1 = perfect circle; 0 = highly irregular
	Deformation	Shape stability between timesteps. Calculated as $1 - \frac{Shared\ Area}{Total\ Area}$.	0 = no change; 1 = complete deformation
	Convex Hull Area	Object area relative to its convex hull area. Calculated as $\frac{Area}{Convex\ Hull\ Area}$.	Lower values = more concave/irregular
MOTION	Centroids per timestep	Geometric centers of all objects at each timestep, handling longitudinal wrapping (0°-360°).	Used to identify object locations.
	Centroid displacement	Distance (km) between centroids across timesteps. Calculated using Haversine distance.	Tracks movement paths; large values = faster movement
	Center-of-mass coordinates	Intensity-weighted mean position per timestep.	Reflects mass distribution shifts
	Center-of-mass displacement	Distance between center-of-mass positions across timesteps.	Quantifies intensity-weighted movement

INTENSITY	Cumulative	Sum of intensity values across space per timestep	Total event magnitude
	Mean	Spatial average intensity	Baseline event strength
	Max.	Spatial maximum intensity	Peak local magnitude
	Std. Dev.	Spatial variability	Higher values = more heterogeneous
	Percentile	Threshold-exceeding intensity (e.g., 90th)	Robust extreme value detection
TEMPORAL	Initial Detection Time	First timestep of event occurrence	Event onset
	Duration	Total timesteps the event persists	Distinguishes transient vs. persistent
	Peak Intensity Timing	When max intensity occurs (% duration)	Early peak = rapid intensification
	Peak Area Timing	When maximum extent occurs	Early peak = rapid growth
CONTEXTUAL	Object counts per timestep	Number of distinct objects detected	Higher counts indicate fragmentation

The computed measures are designed for application across both individual events and multiple events. Each measure is stored in a structured Python dictionary keyed by event ID, supporting single-event analysis and multi-event comparison, where ensemble statistics can be calculated when aggregating across multiple event IDs. The submodule includes built-in visualization tools that generate plots of trajectory maps (centroid/center-of-mass paths with arrow markers indicating directionality).

Implementation example for a few measures

```
event_binary, event_intensity = get_object_masks(blobs, var_notrend, object_id=226.) # specific event
ID

shape = ShapeMeasures(lat_resolution=111.320,
                      lon_resolution=111.320,
                      use_decorators=False)
spatial_extent_data = shape.calc_spatial_extents(event_binary) # Area-related measures

motion = MotionMeasures(use_decorators=False)
centroid_path, centroid_displacements = motion.calculate_centroid_displacement(
  event_binary) # Calculate centroid displacements

# Intensity
intensity_metrics = calculate_intensity_metrics(event_intensity)
```

The nested dictionary output structure accommodates mixed dimensionality, where scalars (e.g., maximum intensity) coexist with vectors (e.g., centroid path) and expansion, where new measures can be added without restructuring existing outputs. This approach is particularly valuable for comparative analyses, where aggregating specific measures across events needs only simple dictionary comprehensions. The Measures submodule allows for user-defined extension, which is an important feature for domain-specific applications.

Implementation example

```
object_ids = [8., 11., 16.]           # Example IDs from Ocetrac tracking output

# Toggle which measures to compute for the objects (True/False flags)
run_shape_flag = True
run_motion_flag = False               # Example: turn off motion measures
run_temporal_flag = True
run_intensity_flag = True


results_for_objects = process_objects_and_calculate_measures(
   object_ids,                        # List of object IDs to process
   blob_data = labelled_field,        # Ocetrac's labeled output (tracked objects)
   intensity_data = intensity_field,  # Intensity data (e.g., SST anomalies)
   run_shape=run_shape_flag,
   run_motion=run_motion_flag,
   run_temporal=run_temporal_flag,
   run_intensity=run_intensity_flag,
   lon_resolution_value=lon_resolution_value,
   lat_resolution_value=lat_resolution_value
)
```

3.4. Flexibility and Extensibility

Ocetrac's modularity supports diverse applications beyond MHWs and can be used to analyze atmospheric systems (with geopotential height data), precipitation events, ocean eddies (using sea surface height anomalies), and freshwater plumes (via salinity fields), amongst others. Ocetrac integrates with `xarray` for labeled data handling, `dask` for parallel processing, and common geospatial toolkits, making the package suitable for studying different geophysical phenomena.

4. Example use cases

To demonstrate Ocetrac's broad applicability, we present several example use cases focused on MHWs. These examples illustrate Ocetrac's capabilities in analyzing global patterns, investigating regional dynamics, placing events within a historical context, and evaluating forecast skill.

4.1. Application to Observational Data

We apply Ocetrac to the NOAA 1/4° Optimum Interpolation Sea Surface Temperature (OISST) dataset [16], [17] to demonstrate its functionality for global MHW analysis. This methodology for anomaly calculation and results is described in greater detail in H. A. Scannell, C. Cai, L. Thompson, D. B. Whitt, D. J. Gagne, and R. P. Abernathey [15]. This study demonstrates Ocetrac successfully identifying known MHW events and their spatial structure, temporal evolution, and intensity distributions from the observational record. This shows how Ocetrac captures both large-scale and regional MHW events in observational products. The workflow is shown in Figure 3.

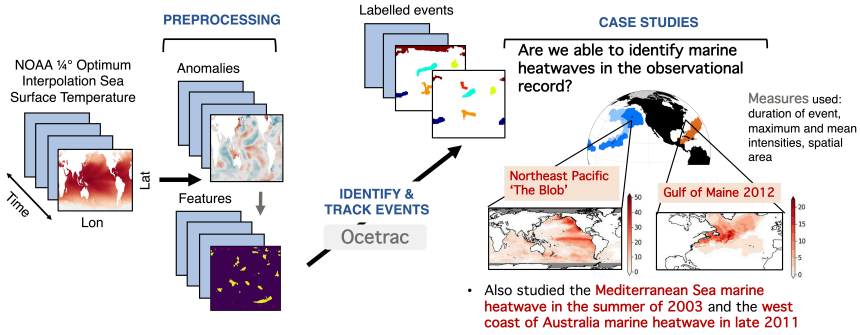


Figure 3. *Ocetrac workflow for MHW analysis in an observational data product (in this example, the NOAA 1/4° Optimum Interpolation Sea Surface Temperature dataset). The diagram illustrates the processing pipeline from model output through event detection, tracking, and characterization of observational MHWs.*

4.2. Characterizing Events in a Large Ensemble

This use case demonstrates that Ocetrac allows for comprehensive analysis of MHWs across a large ensemble of climate simulations. By processing many ensemble members, we generate an extensive event catalog that is orders of magnitude larger than the observational record. This provides robust statistics for MHW characteristics calculated using the Measures submodule, This expanded dataset supports clustering analyses to identify recurrent event types and their associated physical drivers.

The large number of simulated events allows us to address key questions about MHW diversity: What canonical patterns emerge? How do their dynamical drivers differ? Such analyses would not be possible using the observational record alone. The approach is particularly valuable for characterizing rare but high-impact events, where large ensembles provide sufficient samples to assess their physical rarity and driving mechanisms. The workflow is shown in Figure 4.

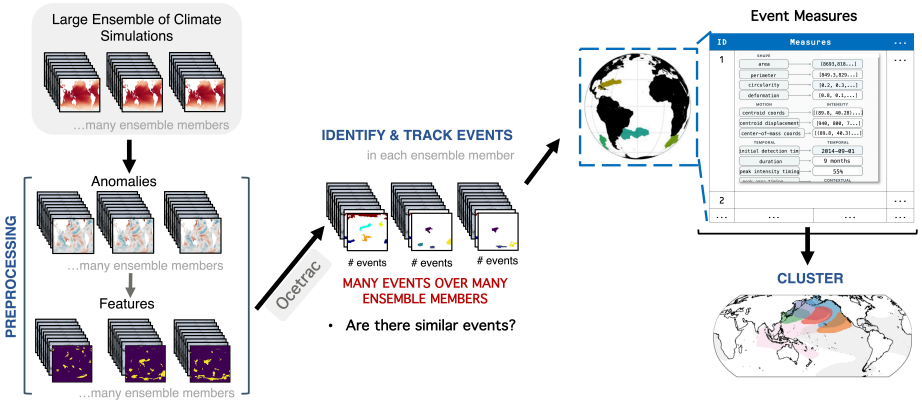


Figure 4. *Ocetrac workflow for MHW analysis in climate ensembles. This workflow focuses on events in the North Pacific Ocean. This approach allows for statistical analysis of event properties.*

4.3. Comparing Observed Marine Heatwaves Against Many Simulated Events

In this scenario, we show the use of Ocetrac to compare MHWs in the observational record (i.e., the spatially large and long-lasting Northeast Pacific MHW 'The Blob' in 2014-2016) with events simulated in a large ensemble of climate models, such as the Community Earth System Model version 2 Large Ensemble Community Project (CESM2-LENS) [18]. The approach facilitates a statistical framework for assessing their rarity, spatial structure, and dynamical drivers. By applying consistent object-based detection and tracking to both observations and model output, Ocetrac allows direct comparisons that address questions such as (1) How unusual was the observed event in the context of internal climate variability?, (2) What is the modeled likelihood of similar or more intense MHWs?, and (3) What types of MHWs does the model suggest are possible but have not yet been observed? An analysis workflow is illustrated in Figure 5.

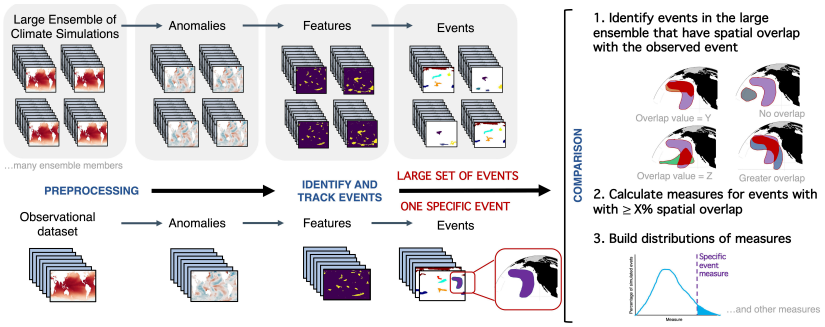


Figure 5. *Ocetrac workflow for comparing observed and simulated MHWs.*

4.4. Evaluating Marine Heatwave Predictions Using Object-Based Metrics

Object-based analysis of MHWs can be applied not only to observations and model output, but also to seasonal forecasts. In recent work, J. T. Cohen, L. Thompson, E. Maroon, A.-L. Deppenmeier, and C. Cai [19] combine Ocetrac with the Method for Object-based Diagnostic Evaluation (MODE) to evaluate MHW forecasts. More information about MODE can be found in C. A. Davis, B. G. Brown, and R. G. Bullock [20], C. A. Davis, B. G. Brown, and R. G. Bullock [21], and R. Bullock, T. Fowler, and B. Brown [22]. Events were identified with `ocetrac.Tracker`, then forecast and observed events were compared and matched based on the similarity of their attributes. Event matching and attribute calculations were performed using MODE. The Measures submodule, however, will allow users to perform this analysis within the Python ecosystem (Figure 6).

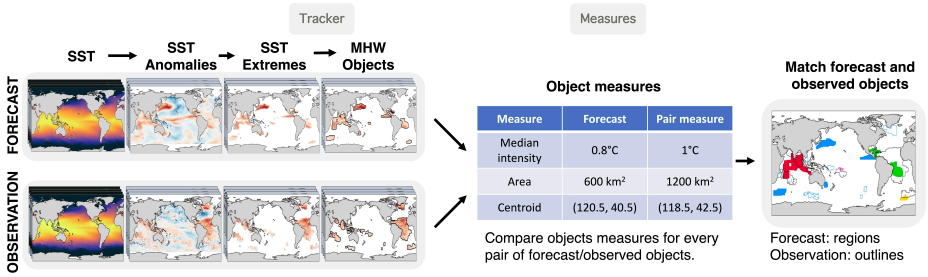


Figure 6. *Ocetrac workflow for comparing forecast and observed MHW events. Objects are identified and tracked with* `ocetrac.Tracker` *and compared between the forecast and observation datasets with ocetrac.measures*

5. Discussion

Ocetrac is an open-source software suite designed to identify, track, and quantify irregular, evolving features in gridded climate data, such as MHWs. It is actively being developed with extensibility in mind. Future plans include: (1) expanding grid compatibility (e.g., Parallel Ocean Program (POP) grid, (2) enhancing tracking to incorporate vertical structure (e.g., MHWs can have subsurface signatures), and (3) customizing analysis (e.g., different kinds of user-defined thresholding and intensity field extraction). Beyond MHWs, Ocetrac is being adapted for other spatiotemporal phenomena, including atmospheric blocking, ocean eddies, and biogeochemical extremes, with practical examples to be demonstrated in interactive tutorials. The Measures submodule will also be extended to include new measures (e.g., co-occurring event counts) and visualization features. Although Ocetrac was originally developed for MHW research, this package is a versatile tool for ocean, atmospheric, and climate sciences. As a Python-based tool, Ocetrac is designed to evolve with both community needs and computational Earth science advancements. We invite collaboration through conferences, workshops, and open development on GitHub, particularly from researchers

working on event detection and feature tracking. These efforts will advance Ocetrac's capacity to analyze the structure of ocean and atmospheric events in observational and model data.

Acknowledgments

We gratefully acknowledge the software and methods contributions to Ocetrac from Ryan Abernathey, Julius Busecke, David John Gagne, and Daniel Whitt (listed alphabetically). The work was supported by funding from NSF, NSF NCAR, the Leonardo DiCaprio Foundation Foundation, Microsoft, the Gordon and Betty Moore Foundation, and the University of Washington eScience Institute.

References

[1] D. Smale *et al.*, "Marine heatwaves threaten global biodiversity and the provision of ecosystem services," *Nat. Clim. Change*, vol. 9, pp. 306–312, 2019, doi: 10.1038/s41558-019-0412-1.

[2] T. P. Hughes *et al.*, "Global warming and recurrent mass bleaching of corals," *Nature*, vol. 543, pp. 373–377, 2017, doi: 10.1038/nature21707.

[3] K. E. Mills *et al.*, "Fisheries management in a changing climate: Lessons from the 2012 ocean heat wave in the Northwest Atlantic," *Oceanography*, vol. 26, no. 2, pp. 191–195, 2013, doi: 10.5670/oceanog.2013.27.

[4] D. J. Gagne, A. McGovern, S. E. Haupt, R. A. Sobash, J. K. Williams, and M. Xue, "Storm-based probabilistic hail forecasting with machine learning applied to convection-allowing ensembles," *Weather and Forecasting*, vol. 32, pp. 1819–1840, 2017, doi: 10.1175/WAF-D-17-0010.1.

[5] P. A. Ullrich, C. M. Zarzycki, E. E. McClenny, M. C. Pinheiro, A. M. Stansfield, and K. A. Reed, "TempestExtremes v2.1: a community framework for feature detection, tracking, and analysis in large datasets," *Geoscientific Model Development*, vol. 14, no. 8, pp. 5023–5048, 2021, doi: 10.5194/gmd-14-5023-2021.

[6] G. Bonino, S. Masina, G. Galimberti, and M. Moretti, "Southern Europe and western Asian marine heatwaves (SEWA-MHWs): a dataset based on macroevents," *Earth System Science Data*, vol. 15, pp. 1269–1285, 2023, doi: 10.5194/essd-15-1269-2023.

[7] R. I. Woolway, E. J. Anderson, and C. Albergel, "Rapidly expanding lake heatwaves under climate change," *Environmental Research Letters*, vol. 16, no. 9, p. 94013, 2021, doi: 10.1088/1748-9326/ac1a3a.

[8] D. Sun, Z. Jing, F. Li, and L. Wu, "Characterizing global marine heatwaves under a spatio-temporal framework," *Progress in Oceanography*, vol. 211, p. 102947, 2023, doi: 10.1016/j.pocean.2022.102947.

[9] D. Sun, F. Li, Z. Jing, and et al., "Correction to: Frequent marine heatwaves hidden below the surface of the global ocean," *Nature Geoscience*, vol. 17, p. 171, 2024, doi: 10.1038/s41561-023-01359-0.

[10] T. Fiolleau and R. Roca, "A database of deep convective systems derived from the intercalibrated meteorological geostationary satellite fleet and the TOOCAN algorithm (2012–2020)," *Earth System Science Data*, vol. 16, no. 9, pp. 4021–4050, 2024, doi: 10.5194/essd-16-4021-2024.

[11] S. van der Walt *et al.*, "scikit-image: Image processing in Python," *PeerJ*, vol. 2, p. e453, 2014, doi: 10.7717/peerj.453.

[12] S. Hoyer and J. Hamman, "xarray: N-D labeled arrays and datasets in Python," *Journal of Open Research Software*, vol. 5, no. 1, p. 10, 2017, doi: 10.5334/jors.148.

[13] M. Rocklin, "Dask: Parallel computation with task scheduling," *Proceedings of the 14th Python in Science Conference*, pp. 130–136, 2015, doi: 10.25080/majora-629e541a-00e.

[14] F. Pedregosa *et al.*, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011, doi: 10.48550/arXiv.1201.0490.

[15] H. A. Scannell, C. Cai, L. Thompson, D. B. Whitt, D. J. Gagne, and R. P. Abernathey, "Spatiotemporal Evolution of Marine Heatwaves Globally," *Journal of Atmospheric and Oceanic Technology*, vol. 41, no. 12, pp. 1247–1263, 2024, doi: 10.1175/JTECH-D-23-0126.1.

[16] R. W. Reynolds, N. A. Rayner, T. M. Smith, D. C. Stokes, and W. Wang, "An improved in situ and satellite SST analysis for climate," *J. Climate*, vol. 15, pp. 1609–1625, 2002, doi: 10.1175/1520-0442(2002)015<1609:AIISAS>2.0.CO;2.

[17] R. W. Reynolds, T. M. Smith, C. Liu, D. B. Chelton, K. S. Casey, and M. G. Schlax, "Daily high-resolution-blended analyses for sea surface temperature," *J. Climate*, vol. 20, pp. 5473–5496, 2007, doi: 10.1175/2007JCLI1824.1.

[18] K. B. Rodgers *et al.*, "Ubiquity of human-induced changes in climate variability," *Earth Syst. Dynam.*, vol. 12, pp. 1393–1411, 2021, doi: 10.5194/esd-12-1393-2021.

[19] J. T. Cohen, L. Thompson, E. Maroon, A.-L. Deppenmeier, and C. Cai, "Object-based evaluation of seasonal-to-multiyear marine heatwave predictions," *ESS Open Archive*, 2025, doi: 10.22541/essoar.173870867.74048627/v1.

[20] C. A. Davis, B. G. Brown, and R. G. Bullock, "Object-based verification of precipitation forecasts, Part I: Methodology and application to mesoscale rain areas," *Mon. Wea. Rev.*, vol. 134, pp. 1772–1784, 2006, doi: 10.1175/mwr3145.1.

[21] C. A. Davis, B. G. Brown, and R. G. Bullock, "Object-based verification of precipitation forecasts, Part II: Application to convective rain systems," *Mon. Wea. Rev.*, vol. 134, pp. 1785–1795, 2006, doi: 10.1175/MWR3146.1.

[22] R. Bullock, T. Fowler, and B. Brown, "Method for Object-Based Diagnostic Evaluation," 2016. doi: 10.5065/D61V5CBS.

Open Access | https://doi.org/10.25080/uaha7744

SciPy 2025

July 7 - July 13, 2025

Proceedings of the 24th Python in Science Conference
ISSN: 2575-9752

OptiMask: Efficiently Finding the Largest NaN-Free Submatrix

Cyril Joly[1]

[1]Airparif

Abstract

When working with tabular data, many processes cannot handle missing values. While imputation strategies are commonly used, another approach is to extract a NaN-free submatrix. Simply dropping every row or column containing NaN values can drastically reduce the dataset, potentially leaving no usable data. OptiMask is a heuristic designed to solve the following optimization problem: identifying the submatrix of maximum size (in terms of the number of elements) without missing values. This submatrix is composed of rows and columns that are not necessarily contiguous in the original data. OptiMask determines the sets of rows and columns to remove from the input matrix, providing either an exact or near-optimal submatrix.

Keywords Optimization, Missing data, Machine Learning

1. Introduction

Missing data is a common challenge in data analysis, often represented as NaN (Not a Number) values in matrices or DataFrames. Many algorithms and statistical methods require complete datasets, necessitating effective handling of missing values [1], [2]. Traditional approaches include imputation (replacing missing values with estimates) and complete-case analysis (discarding rows/columns with any NaN) [3], [4]. However, imputation can introduce bias [5], while complete-case analysis may discard excessive data, especially when missing values are evenly distributed [6].

An alternative strategy is to identify the largest possible NaN-free submatrix, while preserving as much of the original, unaltered data as possible. This reduces to an optimization task: remove the minimal set of rows and columns to yield a NaN-free submatrix of maximum size.

This problem is computationally challenging, as the search space grows exponentially with the number of rows and columns containing NaN. Exact solutions (e.g., linear programming) guarantee optimality but are prohibitively expensive for large matrices.

Heuristic methods like OptiMask provide near-optimal solutions efficiently. OptiMask iteratively permutes rows and columns to isolate NaN values along a frontier, simplifying the search for the largest contiguous NaN-free submatrix. By combining randomization with multiple restarts, it reliably finds high-quality solutions. This paper explores the OptiMask algorithm, theoretical foundations, and practical performance across diverse datasets, including large and structured matrices. It also discusses the `optimask` Python package (https://pypi.org/project/optimask/), which enables applying the algorithm to matrix-like data structures popular with Python programmers.

Published Jul 10, 2025

Correspondence to
Cyril Joly
cyril.joly@airparif.fr

Open Access

Copyright © 2025 Joly. This is an open-access article distributed under the terms of the Creative Commons Attribution 4.0 International license, which enables reusers to distribute, remix, adapt, and build upon the material in any medium or format, so long as attribution is given to the creator.

2. Problem Formalization and Challenges

2.1. Mathematical Definition

Given an $m \times n$ matrix A with missing values (NaN), the goal is to find:

- A subset of rows $\mathcal{R} \subseteq \{1, ..., m\}$ and columns $\mathcal{C} \subseteq \{1, ..., n\}$ such that the submatrix $A[\mathcal{R}, \mathcal{C}]$ contains no NaN values.
- The solution maximizing the size $| \mathcal{R} | \times | \mathcal{C} |$ of the submatrix.

2.2. The Fundamental Trade-off

When handling missing values in a matrix, we must decide whether to remove affected rows, columns, or a combination of both. The optimal choice depends on the matrix's dimensions and the distribution of NaN values:

1. **Toy Example**:
    - In **tall matrices** (rows > columns), removing a problematic **row** typically preserves more data.
    - In **wide matrices** (columns > rows), removing an affected **column** is usually preferable.

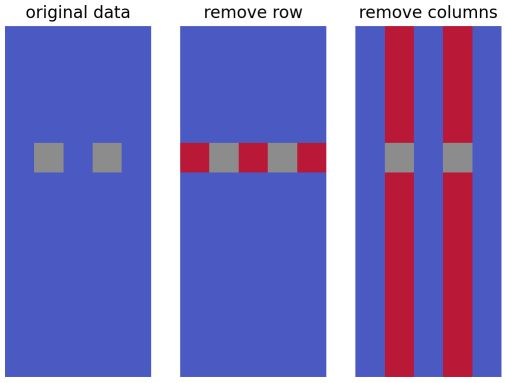


Figure 1. *In this toy example, removing the row with the two NaNs yields a larger submatrix (11x5) than removing the two columns, which results in a smaller submatrix (12x3) from the original 12x5 matrix.*

1. **Generalizing the Intuition**: The toy example suggests a greedy heuristic:
    - If a row contains many NaNs, removing that single row may be better than removing all corresponding columns (which could discard more data).
    - Conversely, if a column contains many NaNs, removing it might be better than eliminating all affected rows.
2. **Complex Cases Require a Systematic Approach**:
    - When NaNs are spread across both rows and columns, the optimal solution isn't obvious. Sometimes, the best solution involves a mix of row and column removals.
    - This necessitates an algorithmic strategy to maximize the preserved submatrix.

2.3. Linear Programming Formulation

The problem can be formulated using integer linear programming [7], defining decision variables for removing rows, columns, and individual cells, subject to constraints ensuring

all NaN values are handled. The objective is to minimize the total number of effectively removed cells, equivalent to maximizing the area of the remaining NaN-free submatrix.

Given:

- A matrix A of shape $m \times n$ with elements $a_{i,j}$.
- Decision variables for $i \in \{1, ..., m\}$ and $j \in \{1, ..., n\}$:
  - $r_i \in \{0, 1\}$: 1 if row i is removed, 0 otherwise.
  - $c_j \in \{0, 1\}$: 1 if column j is removed, 0 otherwise.
  - $e_{i,j} \in \{0, 1\}$: 1 if cell (i, j) is effectively removed (i.e., part of a removed row or column), 0 otherwise.

Constraints:

For all $i \in \{1, ..., m\}$ and $j \in \{1, ..., n\}$:

- If $a_{i,j}$ is NaN, then $e_{i,j} = 1$.
- $r_i + c_j \geq e_{i,j}$.
- $e_{i,j} \geq r_i$.
- $e_{i,j} \geq c_j$.

Objective Function:

Minimize the total number of effectively removed cells:

$$\min \sum_{i=1}^{m} \sum_{j=1}^{n} e_{i,j} \tag{1}$$

This formulation can be solved using integer linear programming solvers (e.g., GLPK [8], Gurobi [9], CPLEX [10]), often interfaced via modeling languages like Pyomo [11] or PuLP [12] in Python for data science practitioners. However, its primary disadvantage is computational cost: for an $m \times n$ matrix, the formulation uses $m \times n + m + n$ binary variables, which becomes prohibitive for large matrices.

3. Algorithm

OptiMask is a heuristic designed to provide near-approximations of the optimal solutions to the problem. The core idea is to compute row and column permutations such that the search for the largest non-contiguous NaN-free submatrix reduces to finding a contiguous one.

3.1. Core Approach

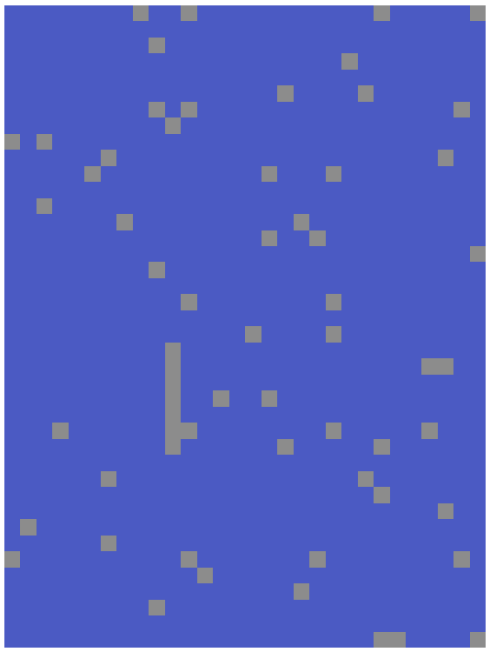


Figure 2. *An example 40x30 matrix, containing 63 missing values, illustrating the algorithm's steps. Grey cells represent missing values.*

OptiMask employs an iterative permutation-based algorithm to identify the largest NaN-free submatrix through these key steps:

1. **Problem Reduction**:
    - Isolate rows and columns containing at least one NaN value (rows or columns without NaNs are preserved, as there is no reason to remove them).
    - Create a boolean mask matrix where True represents NaN positions.
2. **Frontier Detection**:
    - Compute `hx`: column-wise, NaN cell with highest row index (i.e., furthest bottom).
    - Compute `hy`: row-wise rightmost NaN index (from the left).
    - These define the current "NaN frontier" of the matrix.

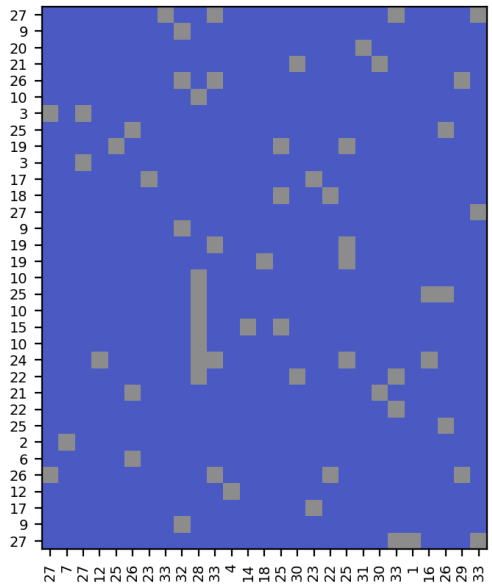


Figure 3. *Steps #1 and #2: isolating the 33 rows and 27 columns with NaNs and computing* `hx` *and* `hy`*.*

1. **Permutation Phase**:
    - Alternately sort rows and columns to push NaN values toward a Pareto frontier.
    - Even iterations: Sort columns by descending `hx`.
    - Odd iterations: Sort rows by descending `hy`.
    - Track all permutations applied during this process.
    - Repeat until both `hx` and `hy` form non-increasing sequences, indicating an optimal NaN frontier.

Figure 4. *Three alternate permutations lead to a Pareto frontier of NaNs.*

1. **Submatrix Extraction**:
    - Identify the largest contiguous NaN-free rectangle in the permuted space.

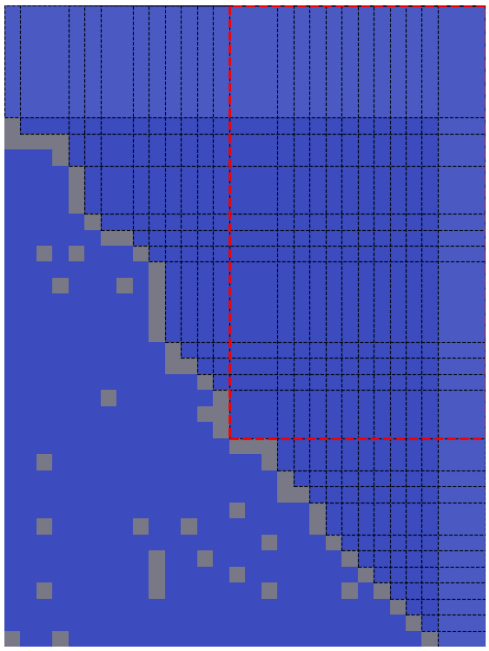


Figure 5. *OptiMask result in permuted space: the black-dotted rectangles are candidates for the largest contiguous NaN-free submatrix, with the red-dotted one selected for its maximal area.*

- Apply inverse permutations to map back to original row/column indices.

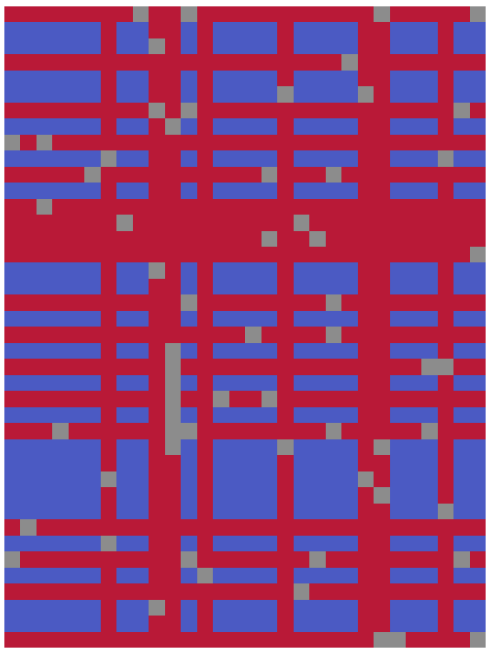


Figure 6. *OptiMask result: grey indicates missing values, red indicates removed rows and columns, blue marks the computed NaN-free submatrix. The algorithm computes a 27x17 submatrix, whereas discarding every row with NaN would result in a 7x30 submatrix, and discarding every column with NaN would yield a 40x3 submatrix.*

4. Python Package

A Python implementation of the algorithm is available on PyPI (https://pypi.org/project/optimask/) and conda-forge (https://anaconda.org/conda-forge/optimask). The library uses Numba [13] for speed and supports common input formats, including NumPy arrays [14], pandas DataFrames [15], and Polars DataFrames [16].

4.1. Basic Usage

```
import numpy as np
from optimask import OptiMask
from optimask.utils import generate_mar

# Generate a Missing At Random matrix with 2% NaN values
x = generate_mar(m=100_000, n=1_000, ratio=0.02, rng=0)
rows, cols = OptiMask(random_state=0).solve(x)
np.isnan(x[np.ix_(rows, cols)]).any()  # False
len(rows), len(cols)  # (37386, 49)
```

This computation takes approximately 200ms on an average personal computer. The implementation provides the sorted indices of the rows and columns to retain, ensuring that the relative order of the elements is preserved.

4.2. Handling Missing Data for Machine Learning

OptiMask removes missing values (NaN) from datasets while maximizing usable data. It does this by finding an optimal subset of samples (rows) and features (columns) to discard, ensuring the remaining data contains no missing values. This makes the dataset directly usable for machine learning models that require complete data, such as scikit-learn's linear models:

```
import numpy as np
from optimask import OptiMask
from sklearn.datasets import make_spd_matrix
from sklearn.linear_model import LinearRegression

def load_data_with_nan(m, n, nan_ratio):
    mean = np.random.randn(n + 1)
    cov = make_spd_matrix(n + 1)
    data = np.random.multivariate_normal(mean=mean, cov=cov, size=m)
    mask = np.random.rand(*data.shape) < nan_ratio
    data[mask] = np.nan
    return data[:, :-1], data[:, -1]

# Simulate a dataset with missing values
X, y = load_data_with_nan(m=10_000, n=100, nan_ratio=0.02)

# Drop samples where the target (y) is missing
valid_samples = np.isfinite(y)
X, y = X[valid_samples], y[valid_samples]

# Apply OptiMask to remove remaining NaNs
rows, cols = OptiMask().solve(X)
X_clean, y_clean = X[np.ix_(rows, cols)], y[rows]
print(X_clean.shape, y_clean.shape) # (3581, 50) (3581,)

# Train a model on the NaN-free data
model = LinearRegression().fit(X=X_clean, y=y_clean)
```

By strategically selecting which rows and columns to keep, OptiMask ensures the dataset remains meaningful while becoming fully trainable.

5. Conclusion

OptiMask provides a heuristic for finding the largest NaN-free submatrix in large datasets where exact methods like linear programming become computationally impractical. By strategically permuting rows and columns to isolate missing values, it offers a practical solution that preserves maximal data without imputation. The Python implementation supports common data structures (numpy, pandas, polars) and delivers results efficiently even for large matrices.

Future work will explore theoretical guarantees on the approximation quality and extensions to weighted optimization problems. The implementation will continue to be optimized for speed in subsequent versions. Finally, an OptiMask-based algorithm for tabular imputation will be developed and benchmarked against MICE (Multiple Imputation by Chained Equations, [5]) to evaluate whether it can achieve better or faster results.

Acknowledgements

This work was funded by Airparif. I'd like to thank Paul Catala (Université de Lorraine) and Alexis Lebeau (RTE) for their assistance and review.

References

[1] R. J. Little and D. B. Rubin, *Statistical analysis with missing data*, vol. 793. John Wiley & Sons, 2019. doi: 10.1002/9781119482260.

[2] D. B. Rubin, *Multiple imputation for nonresponse in surveys*, vol. 81. John Wiley & Sons, 2004. doi: 10.1002/9780470316696.

[3] J. L. Schafer, *Analysis of incomplete multivariate data*. CRC press, 1997. doi: 10.1201/9781439821862.

[4] S. Van Buuren, *Flexible imputation of missing data*. CRC press, 2018. doi: 10.1201/9780429492259.

[5] I. R. White, P. Royston, and A. M. Wood, "Multiple imputation using chained equations: issues and guidance for practice," *Statistics in medicine*, vol. 30, no. 4, pp. 377–399, 2011, doi: 10.1002/sim.4067.

[6] C. K. Enders, *Applied missing data analysis*. Guilford press, 2010. doi: 10.1111/j.1467-842X.2012.00656.x.

[7] L. A. Wolsey, *Integer programming*. John Wiley & Sons, 2020. doi: 10.1002/9781119606475.

[8] A. Makhorin, "GLPK (GNU linear programming kit)," 2012. [Online]. Available: http://www.gnu.org/software/glpk/

[9] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual." [Online]. Available: https://www.gurobi.com/

[10] IBM ILOG CPLEX, "V12. 1: User's Manual for CPLEX," 2009.

[11] W. E. Hart, J.-P. Watson, and D. L. Woodruff, "Pyomo: modeling and solving mathematical programs in Python," *Mathematical Programming Computation*, vol. 3, no. 3, pp. 219–260, 2011, doi: 10.1007/s12532-011-0026-8.

[12] S. Mitchell, M. OSullivan, and I. Dunning, "PuLP: a linear programming toolkit for python." [Online]. Available: https://pythonhosted.org/PuLP/

[13] S. K. Lam, A. Pitrou, and S. Seibert, "Numba: A llvm-based python jit compiler," in *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, 2015, pp. 1–6. doi: 10.1145/2833157.2833162.

[14] C. R. Harris *et al.*, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, 2020, doi: 10.1038/s41586-020-2649-2.

[15] W. McKinney and others, "Data structures for statistical computing in python," in *Proceedings of the 9th Python in Science Conference*, 2010, pp. 51–56. doi: 10.25080/Majora-92bf1922-00a.

[16] R. Vink, "Polars: Lightning-fast DataFrame library for Rust and Python." [Online]. Available: https://pola.rs/

Open Access | https://doi.org/10.25080/wjjm7869

Python is all you need: an overview of the composable, Python-native data stack

Deepyaman Datta[1]

[1]Kedro

SciPy 2025

July 7 - July 13, 2025

Proceedings of the 24th Python in Science Conference
ISSN: 2575-9752

Abstract

For the past decade, SQL has reigned king of the data transformation world, and tools like dbt have formed a cornerstone of the modern data stack. Until recently, Python-first alternatives couldn't compete with the scale and performance of modern SQL. Now Ibis can provide the same benefits of SQL execution with a flexible Python dataframe API.

Through new integrations, Ibis supercharges existing open-source libraries like Kedro and Pandera. Combined, these technologies enable building and orchestrating scalable data engineering pipelines without sacrificing the comfort and other advantages of Python.

Keywords data engineering, modern data stack, composable data stack, data ingestion, data transformation, data validation

1. Introduction

By 2016, the modern data stack was a well-defined idea [1] that had begun to take hold of the data engineering community. Specifically, it represented a set of products that enabled end-to-end data transformation *in* the cloud using SQL. This mindset shift was driven by the advent of cloud-based analytics databases like Amazon Redshift, Google BigQuery, and Snowflake [2] and the subsequent rise of the extract, load, transform (ELT) paradigm. Compared to the extract, transform, load (ETL) workflows that had become ubiquitous since their invention in the 1970s [3], ELT leveraged the power and efficiency of cloud data warehouses in performing transformations [4], combined with the simplicity of managed compute and processes. The differences between ETL and ELT can be seen in Figure 1 below.

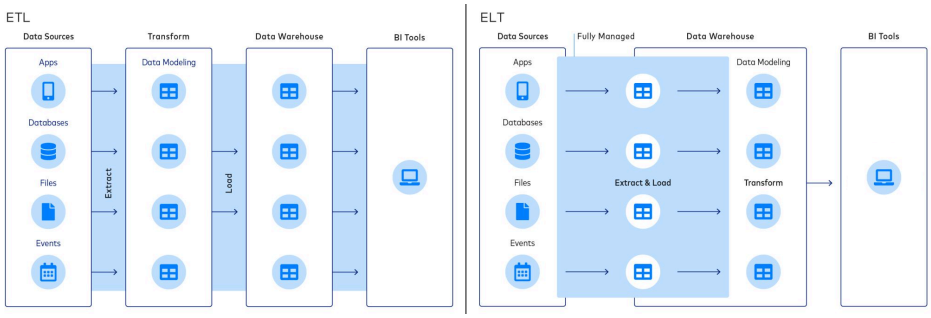


Figure 1. *In ETL workflows (left), data is modeled and transformed before loading it into the data warehouse. In contrast, in ELT workflows (right), data sources are extracted and loaded into the data warehouse before building transformations to create data models for analytics [3].*

Over the next few years, the tooling surrounding the modern data stack matured. Products like Fivetran and Airbyte solidified their position as data ingestion tools, and various data quality and observability vendors offered solutions for ensuring data correctness and completeness as part of the end-to-end process. Business intelligence (BI) tools that offered

Published Jul 10, 2025

Correspondence to
Deepyaman Datta
deepyaman.datta@utexas.edu

Open Access

Copyright © 2025 Datta. This is an open-access article distributed under the terms of the Creative Commons Attribution 4.0 International license, which enables reusers to distribute, remix, adapt, and build upon the material in any medium or format, so long as attribution is given to the creator.

cloud-native analytics workflows, such as Looker, rose in popularity, while established players like Tableau evolved (out of necessity) to become more cloud-native. Last but not least, dbt became the de facto standard for SQL-based transformation in the cloud. By 2024, the tenets of the modern data stack were so widely accepted and adopted that Tristan Handy, the individual who initially coined the term, wrote that the idea of distinguishing the "modern" data stack had outlived it's usefulness:

> When I was a consultant, helping small companies build analytics capabilities, I would only work with MDS tooling. It was so much better that I simply wouldn't take on a project if the client wanted to use pre-cloud tools. I, and others like me, needed a way to talk about this preference, to differentiate between these new products and all of the other ones that we didn't want to use. The term actually conveyed important information.
>
> Today, most data products are built for the cloud. Either they have been built in the past ten years and therefore baked in cloud-first assumptions, or they have been re-architected to do so.
>
> — T. Handy [1]

However, when data and analytics engineers talk about the modern data stack, they invariably mean SQL-based workflows. Most *SQL-oriented* data products are built for the cloud. Despite Python's explosive growth—by some measures, overtaking SQL in popularity among developers [5]—the Python data ecosystem has lagged in supporting data engineering best practices. Python is often relegated to the fallback option for when a use case can't be solved be solved with SQL, and support can be limited [6] (if available at all).

In this paper, we present a set of new software integrations that form the basis for the composable Python analytics stack[1]. First, we will introduce the concept of composable data management systems and Ibis [7], the data processing workhorse of the Python analytics stack. Then, we will present two key components of the emerging stack: Kedro as the core transformation framework and Pandera for data validation. In both cases, we will highlight how Ibis extends the capabilities of the existing, established tool. Finally, we will step back and look at the remaining pieces of the composable analytics stack. We will fill out the picture with Python-native recommendations for ingestion and orchestration. We will also be transparent about some of the current gaps compared to the more established SQL-first approach and ongoing work to address them.

2. Ibis and the rise of composable data management systems

In 2021, engineering leaders from Meta, Voltron Data, Databricks, and Sundeck published a joint paper outlining a vision for a modular data stack of reusable components built on open standards [8], as seen in Figure 2. In short, they posited that modern data systems all share a common set of logical components, and developers of modern data management systems should take advantage of these similarities instead of reinventing the wheel and further fragmenting the ecosystem. They argued that a paradigm shift towards composable data systems would not only benefit those building such products but also user experience by improving consistency and reducing the learning curve across libraries.

[1] Given Handy suggests "analytics stack" as a more appropriate term for the ecosystem of cloud-first data tools [1], and to avoid confusion with the existing set of technologies closely associated with the phrase "modern data stack," we prefer to use the name "Python analytics stack" or "Python data stack."

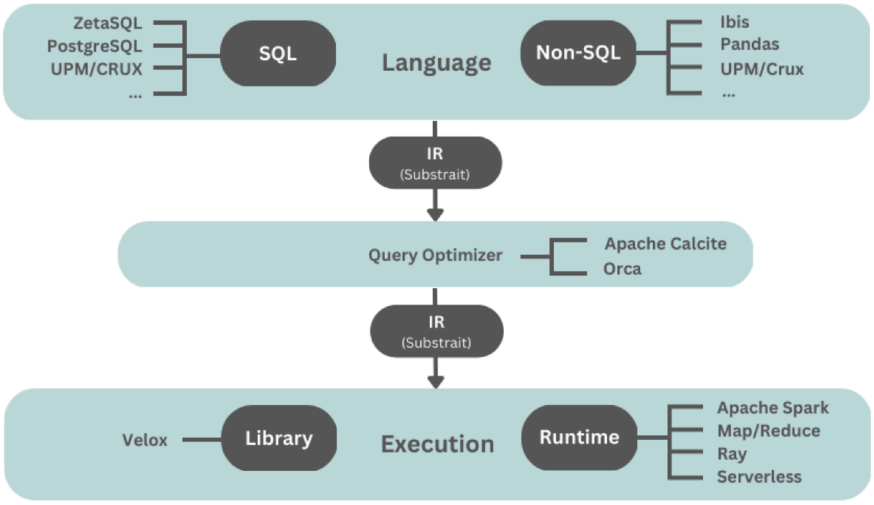


Figure 2. *The emerging modular data stack can be distilled into a general model comprised of five layers. A language component generates an intermediate representation (IR) that serves as input to a query optimizer. The optimizer produces query framents that are passed to an execution engine.*

One of the key results of this architecture is that users should be empowered to use the language frontend of their choice with the execution engine that they need. Whereas SQL provides one standardized[2] interface for modern data systems, many use cases demand non-SQL APIs. Pushed by the popularity of Python, coinciding with the increased prevalence of data processing for machine learning and artificial intelligence, more and more data products developed Python dataframe APIs: PySpark and the pandas API on Spark, BigQuery DataFrames, Snowpark and the Snowpark pandas API. However, many database system developers don't have the resources or expertise to build dataframe-like APIs on top of execution engines. Furthermore, P. Pedreira *et al.* [8] point out the merits of the model popularized by Apache Spark [10], where SQL (Spark SQL) and non-SQL (DataFrames, Datasets, PySpark, and the pandas API on Spark) APIs all generate the same underlying intermediate representation (IR), meaning that the following queries should be indistinguishable from an execution standpoint:

```sql
SELECT a, b, c FROM t WHERE a > 1
```

Program 1. *Select a subset of rows and columns using SQL.*

```python
t.filter(t.a > 1).select("a", "b", "c")
```

Program 2. *Select the same subset of rows and columns using a dataframe API.*

This need prompted the creation of open-source libraries like Ibis, the portable Python dataframe API. Ibis provides a lightweight, universal interface for data wrangling that supports 20+ query engines, from local backends like Polars, DuckDB, and DataFusion to remote databases and distributed computation frameworks like BigQuery, Snowflake, and Spark [7]. Ibis constructs a query plan, or IR, that it evaluates lazily (i.e. as and when needed) on the execution engine [11]. Because Ibis code is functionally equivalent to SQL—in fact,

[2]The SQL standard isn't a well-adhered-to standard in that each database supports a slightly different version of the SQL standard, often referred to as a SQL dialect. G. Forsyth [9] shows a few common examples of queries that need to be expressed differently based on the database or execution engine being used.

Ibis produces and executes SQL under the hood for all but the Polars backend—it is well suited for ELT workflows as part of a Python-first data stack.

3. Building the composable, Python-native data stack

In order to propose a valid Python-first alternative to the SQL-based analytics stack, we break the existing stack into a set of primary functions and address each in turn. As per T. Krantz and A. Jonker [12], the modern data (or analytics) stack consists of several core components including:

- Data storage
- Data ingestion
- Data transformation
- BI and analytics
- Data observability

Some definitions of the analytics stack include specific callouts for reverse ETL, orchestration, and other aspects, but we will focus on the above five layers.

3.1. Data storage

The storage layer of the analytics stack provides a centralized, most often cloud-based, medium for managing data. Traditionally, online analytical processing (OLAP) databases like Redshift, BigQuery, and Snowflake were the primary storages associated with the modern data stack. Data lakes that supported managing large data volumes in cloud object storage have also been popular options; distributed query engines like Spark and Trino are well-suited for workloads accessing such file systems. Finally, recent interest in data lakehouse architectures has resulted in increased support for working with data stored in Iceberg, Delta, and other related table formats.

Ibis supports a wide range of file formats, including Parquet files, CSV files, and newline-delimited JSON data. It also allows reading from and writing to Delta Lake tables. If some Ibis execution backend natively implements a more efficient path for working with a particular format, Ibis will leverage it; otherwise, Ibis falls back to a unified solution for handling data in PyArrow record batches. For data warehouses, the storage engine doubles as the execution engine, so Ibis can create, read, update, and delete tables by running the appropriate SQL statements under the hood. Ultimately, data storage layer support comes down to the underlying engine, and Ibis attempts to keep up to date with new developments in its supported backends.

3.2. Data ingestion

Data ingestion is the process of extracting data from various sources and loading it into a data storage system (like those discussed in the previous section) for processing and analysis [12]. Because of the activities performed, this process is is also often referred to as the extract and load (EL) stage.

dlt is an open-source, production-ready Python framework that has become a popular choice as part of the analytics stack [13]. Since many data engineers are already opting for dlt over established like Fivetran and Airbyte, we don't find it necessary to justify dlt's space in the Python analytics stack. That said, it is worth mentioning that dlt also provides Pythonic reverse ETL [14] (one of the categories that we chose not to cover explicitly, given the broader industry trend favoring general data movement tools over dedicated reverse ETL frameworks). dlt also supports basic transformation capabilities, including built-in Ibis support. However, for all but the simplest use cases, we recommend a dedicated transfor-

mation framework, much in the way dlt is frequently used alongside dbt in the existing analytics stack.

```python
from pathlib import Path

import dlt
from dlt.sources.filesystem import filesystem, read_csv_duckdb

# Create a list of dlt resources corresponding to each of the raw files.
readers = []
for name in ["raw_customers", "raw_orders", "raw_payments"]:
    files = filesystem(
        bucket_url=(Path(__file__).parent / "data").as_uri(),
        file_glob=f"01_raw/{name}.csv",
    )
    reader = (files | read_csv_duckdb()).with_name(name)
    readers.append(reader)

# Create a new dlt pipeline configured to use DuckDB as the destination.
pipeline = dlt.pipeline(
    pipeline_name="jaffle_shop", dataset_name="main", destination="duckdb"
)

# Run the pipeline to load data into DuckDB, and output run information.
info = pipeline.run(readers)
print(info)
```

Program 3. *A dlt pipeline to load data from local CSV files into a DuckDB database.*

3.3. Data transformation

After data lands in the centralized data storage, it needs to be cleaned, processed, and combined before it's useful for analytics, reporting, or machine learning. Before dbt became the standard for SQL data transformation, the SQL data transformation layer often consisted of an unstructured mess of scripts, stored procedures, or—even worse—GUI-based data pipelines. dbt changed the game by providing a "SQL-first transformation workflow that lets teams quickly and collaboratively deploy analytics code following software engineering best practices" [15].

Kedro is an open-source Python framework that empowers data teams to write reproducible, maintainable, and modular data pipelines [16]. Kedro joined the Linux Foundation AI & Data Foundation in 2021, graduating from Incubation in 2024 [17]. Like dbt, Kedro helps apply software engineering best practices to data transformation code. Both projects also model transformation workflows as directed acyclic graphs (DAGs). A former dbt Labs product manager remarked on the similarities between the two frameworks:

> When I learned about Kedro (while at dbt Labs), I commented that it was like dbt if it were created by Python data scientists instead of SQL data analysts (including both being created out of consulting companies).
>
> — Cody Peterson, Technical Product Manager @ Voltron Data [11]

Our work integrating Kedro and Ibis opens the door to Python transformation workflows on par with SQL-based transformation with tools like dbt. To illustrate, we reimplement the canonical Jaffle Shop dbt demo project [18] using this stack.

3.3.1. *Configuring the Data Catalog*

One of the core features of Kedro is the Data Catalog, a per-Kedro-project repository of data sources and sinks configured for use by the project [16]. Kedro also supports a concept of data connectors called datasets. To enable reading and writing Ibis tables in Kedro pipelines, we contributed Ibis dataset implementations for handling data in tables and in

files to Kedro-Datasets, a Python package containing commonly-used dataset implementations maintained by the Kedro team.

The standard approach for using a dataset is to add it to the `catalog.yml` configuration file:

```yaml
_duckdb:
  backend: duckdb
  # `database` and `threads` are parameters for `ibis.duckdb.connect()`.
  database: jaffle_shop.duckdb
  threads: 24

seed_customers:
  type: ibis.FileDataset
  filepath: data/01_raw/raw_customers.csv
  file_format: csv
  connection: ${_duckdb}

raw_customers:
  type: ibis.TableDataset
  table_name: raw_customers
  connection: ${_duckdb}
  save_args:
    materialized: table

...
```

Program 4. *A subset of the dataset definitions used to implement a port of the Jaffle Shop dbt demo project [18] using Kedro and Ibis.*

With the Kedro-Ibis integration, data is represented in memory as Ibis tables, analogous to other lazy dataframe implementations like Polars LazyFrames and Spark DataFrames. The main difference is that Ibis can work with a much wider array of backend data systems; for example, the above example connects to a DuckDB database. The implementations of the Ibis datasets also abstract other functionalities that data engineers require, including support for different materialization strategies (`seed_customers` stores loaded data in a view, whereas `raw_customers` overrides the default behavior and persists data as a table) [11].

Other optimizations include caching connections for reuse. Similar to how dbt configures connection details in `profiles.yml` [19], we leverage variable interpolation supported by the Kedro configuration loader to define and reuse connection information [11].

3.3.2. *Building pipelines*

Kedro pipelines are composed of nodes, which themselves are simply Python functions. Given the datasets described in the above section load and save data as Ibis tables, the function needs to take as input and return as output any number of Ibis tables.

```python
from __future__ import annotations

import ibis


def process_orders(
    orders: ibis.Table, payments: ibis.Table, payment_methods: list[str]
) -> ibis.Table:
    total_amount_by_payment_method = {}
    for payment_method in payment_methods:
        total_amount_by_payment_method[f"{payment_method}_amount"] = ibis.coalesce(
            payments.amount.sum(where=payments.payment_method == payment_method), 0
        )

    order_payments = payments.group_by("order_id").aggregate(
        **total_amount_by_payment_method, total_amount=payments.amount.sum()
    )

    final = orders.left_join(order_payments, "order_id")[
        [
            orders.order_id,
            orders.customer_id,
            orders.order_date,
            orders.status,
            *[
                order_payments[f"{payment_method}_amount"]
                for payment_method in payment_methods
            ],
            order_payments.total_amount.name("amount"),
        ]
    ]
    return final
```

Program 5. *The same logic as the `orders.sql` model from dbt Labs, Inc. [18] expressed using Ibis in a Python function.*

One of the advantages of native Python is cleaner parametrization. In the above example, we iterate over `payment_methods` using a for loop. By comparison, a templating language like Jinja is necessary to extend the capabilities of SQL to support control structures, variable interpolation, and more in dbt [20]. Another benefit of Python as a data transformation language is that it supports unit tests. While dbt added support for unit tests in 2024, it has many limitations [21].

A complete port of the Jaffle Shop project using Kedro and Ibis is available on GitHub [22].

3.4. BI and analytics

The BI and analytics layer includes everything from dashboards (with tools like PowerBI, Tableau, Looker, and Streamlit) to machine learning and data science [12]. While SQL is well supported choice by most BI tools, Python holds a significant edge in AI and ML workflows. In fact, we believe that data scientists, machine learning engineers, and data engineers who work alongside data science teams are likely to be the biggest beneficiaries of the emergence of the Python-first analytics stack. Kedro is already established as a popular framework for authoring data science pipelines, and these developments help enable cross-functional teams to write end-to-end data transformation pipelines using a single technology.

That said, Ibis does integrate with a number of data visualization and dashboarding tools, including Streamlit and Plotly [7]. Furthermore, most BI tools support Python to some extent, and Ibis can produce data in a supported dataframe format.

3.5. Data observability

Data observability is a broad category that includes data quality, freshness, and other metrics [12]. In our work, we focused on providing a solution for data validation as part of

the composable Python analytics stack. Pandera originated as a lightweight-yet-expressive API for validating pandas dataframes [23]. As it evolved, Pandera added support for other pandas-compatible APIs, including GeoPandas, Dask, Modin, and the pandas API on Spark. Later, Pandera loosened its tight coupling with the pandas API in order to support other execution engines like Polars [24].

We take advantage of this prior work to support validating Ibis tables using Pandera—and, by extension, enable data validation across the full suite of Ibis-supported backends. For built-in checks (i.e. the wide range of common data quality checks that Pandera supports out of the box), the fact that *Ibis* enables validation on a previously-unsupported data processing framework or database can be almost transparent to the user. For more bespoke checks, the Ibis backend (similar to all of the previously-existing Pandera backends) supports defining custom checks, and these do need to be written using Ibis syntax.

```
import ibis
import pandera.ibis as pa
from ibis import _
from pandera.ibis import IbisData


def total_amount_positive(data: IbisData) -> ibis.Table:
    w = ibis.window(group_by="order_id")
    with_total_amount = data.table.mutate(total_amount=data.table.amount.sum().over(w))
    return with_total_amount.order_by("order_id").select(_.total_amount >= 0)


schema = pa.DataFrameSchema(
    columns={
        "order_id": pa.Column(int),
        "amount": pa.Column(float),
        "status": pa.Column(
            str,
            pa.Check.isin(
                ["placed", "shipped", "completed", "returned", "return_pending"]
            ),
        ),
    },
    checks=[pa.Check(total_amount_positive)],
)

con = ibis.duckdb.connect("jaffle_shop.duckdb")
orders = con.table("orders")

schema.validate(orders)
```

Program 6. *An example of validating data in DuckDB using the Ibis backend for Pandera, including both built-in and custom checks.*

4. The future of the composable, Python-native data stack

In this paper, we have laid out the foundations for the composable, Python-native data stack. As with any new development in the open-source data ecosystem, getting to this point would not have been possible without the contributions of many people and projects over the past decade—maintainers, committers, contributors, and users across dlt, Kedro, Pandera, and, of course, Ibis.

At the same time, the Python data stack is not (yet) a product category that the data engineering community has coalesced around in the way that the modern data stack went from an idea to the accepted way to do data engineering using SQL and the cloud years prior. This means that there are gaps waiting to be addressed and hardening that needs to happen. dlt has quickly gained traction among data engineers. Kedro is a mature data transformation framework that benefits from an active community and usage across academia, research institutions, startups, and Fortune 500 companies. The Ibis integration has

been widely adopted by users, despite being relatively new; this usage has also highlighted functionalities that need adding, such as support for insert and upsert operations, both of which are in progress. Pandera itself is a very popular choice for data validation, but the Ibis integration is new. Other integrations, usually driven by the community (between Kedro and Pandera, as well as between Kedro and workflow orchestrators like Airflow and Dagster), show promise as part of a future iteration of the stack.

The future is bright for the Python analytics stack. For one, the pieces fell together not because we wanted to create a Python-based alternative to the already-successful SQL data stack, but rather because users were performing data engineering tasks using Python and didn't have the tooling to do it the "right" way. Furthermore, the growth in popularity of Python means that more users will face the same challenges we already do. While not a focus of this paper, some of the frameworks presented also have technical advantages that come from the composable nature of the proposed stack. Because Ibis provides a unified API, data transformation and data validation using Ibis is portable, meaning you can develop using local engines and run in production using other engines. By extension, the Python-first analytics stack is well-positioned with increasing interest in multi-engine stacks.

References

[1] T. Handy, "Is the "Modern Data Stack" Still a Useful Idea?." [Online]. Available: https://roundup.getdbt.com/p/is-the-modern-data-stack-still-a

[2] T. Handy, "What are the steps / tools in setting up a modern, SaaS-based BI infrastructure?." [Online]. Available: https://www.linkedin.com/pulse/what-steps-tools-setting-up-modern-saas-based-bi-tristan-handy/

[3] C. Wang, "ETL vs. ELT: Choose the right approach for data integration." [Online]. Available: https://www.fivetran.com/blog/etl-vs-elt

[4] I. a. i. a. Amazon Web Services, "Data Warehousing on AWS: AWS Whitepaper," 2021. [Online]. Available: https://docs.aws.amazon.com/pdfs/whitepapers/latest/data-warehousing-on-aws/data-warehousing-on-aws.pdf

[5] Stack Exchange, Inc., "Stack Overflow Developer Survey 2023." [Online]. Available: https://survey.stackoverflow.co/2023/#section-most-popular-technologies-programming-scripting-and-markup-languages

[6] dbt Labs, Inc., "Python models." [Online]. Available: https://docs.getdbt.com/docs/build/python-models

[7] Ibis maintainers, "Ibis: the portable Python dataframe library." [Online]. Available: https://ibis-project.org/

[8] P. Pedreira *et al.*, "The Composable Data Management System Manifesto," *Proceedings of the VLDB Endowment*, vol. 16, no. 10, pp. 2679–2685, 2023, doi: 10.14778/3603581.3603604.

[9] G. Forsyth, "Designing Interfaces is hard." [Online]. Available: https://ibis-project.org/presentations/datafusion-meetup-nyc-2024/talk#/sql-questions-with-no-single-answer

[10] M. Zaharia *et al.*, "Apache Spark: a unified engine for big data processing," *Communications of the ACM*, vol. 59, no. 11, pp. 56–65, 2016, doi: 10.1145/2934664.

[11] D. Datta, "Building scalable data pipelines with Kedro and Ibis." [Online]. Available: https://kedro.org/blog/building-scalable-data-pipelines-with-kedro-and-ibis

[12] T. Krantz and A. Jonker, "What is the modern data stack?." [Online]. Available: https://www.ibm.com/think/topics/modern-data-stack

[13] dltHub, "dltHub: ELT as Python Code." [Online]. Available: https://dlthub.com/

[14] A. Brudaru, "dlt adds Reverse ETL - build a custom destination in minutes." [Online]. Available: https://dlthub.com/blog/reverse-etl-dlt

[15] dbt Labs, Inc., "What is dbt?." [Online]. Available: https://www.getdbt.com/product/what-is-dbt

[16] S. Alam *et al.*, "Kedro." [Online]. Available: https://github.com/kedro-org/kedro

[17] T. L. Foundation, "Kedro." [Online]. Available: https://lfaidata.foundation/projects/kedro/

[18] dbt Labs, Inc., "The Jaffle Shop." [Online]. Available: https://github.com/dbt-labs/jaffle_shop_duckdb

[19] dbt Labs, Inc., "About profiles.yml." [Online]. Available: https://docs.getdbt.com/docs/core/connect-data-platform/profiles.yml

[20] dbt Labs, Inc., "Jinja and macros." [Online]. Available: https://docs.getdbt.com/docs/build/jinja-macros

[21] dbt Labs, Inc., "Unit tests." [Online]. Available: https://docs.getdbt.com/docs/build/unit-tests

[22] D. Datta, “Jaffle Shop.” [Online]. Available: https://github.com/deepyaman/jaffle-shop

[23] N. Bantilan, “pandera: Statistical Data Validation of Pandas Dataframes,” in *Proceedings of the 19th Python in Science Conference*, in SciPy. 2020, pp. 116–124. doi: 10.25080/majora-342d178e-010.

[24] N. Bantilan, “Pandera: Going Beyond Pandas Data Validation,” in *Proceedings of the 22nd Python in Science Conference*, in SciPy. 2023, pp. 124–129. doi: 10.25080/gerudo-f2bc6f59-010.

Open Access | https://doi.org/10.25080/crcw9835

SciPy 2025

July 7 - July 13, 2025

Proceedings of the 24th Python in Science Conference
ISSN: 2575-9752

Zamba: Computer vision for wildlife conservation

Emily Dorne[1], **Jay Qi**[1], **Peter Bull**[1], **Colleen Stephens**[2], **Mattia Bessone**[2,3,4], **Benjamin Debetencourt**[5,2], **Barbara Fruth**[6,7], **David Morgan**[8], **Meredith S. Palmer**[9], **Crickette Sanz**[10,11], **Janika Wendefeuer**[12,13], **Catherine Crockford**[14,15], **Tobias Deschner**[16,2], **Kevin E. Langergraber**[17,18], **Alex K. Piel**[19,20,21], **Martha Robbins**[22], **Volker Sommer**[19,23], **Fiona A. Stewart**[19,20,21], **Roman M. Wittig**[14,15], **Klaus Zuberbuehler**[24,25], **Hjalmar S. Kühl**[26,27], and **Mimi Arandjelovic**[22,28]

[1]DrivenData, [2]Max Planck Institute for Evolutionary Anthropology, [3]Centre for the Advanced Study of Collective Behaviour, University of Konstanz, [4]Department of Animal Societies, Max Planck Institute of Animal Behavior, [5]Wild Chimpanzee Foundation, [6]Department for the Ecology of Animal Societies, Max-Planck Institute of Animal Behavior, [7]Centre for Research and Conservation/KMDA, [8]Lester E. Fisher Center for the Study and Conservation of Apes, Lincoln Park Zoo, [9]Center for Biodiversity and Global Change, Yale University, [10]Department of Anthropology, Washington University in Saint Louis, [11]Congo Program, Wildlife Conservation Society, [12]Universität Hamburg, [13]WWF Central African Republic, [14]Ape Social Mind Lab, Institute for Cognitive Sciences Marc Jeannerod (UMR 5229), CNRS and University of Lyon 1, [15]Taï Chimpanzee Project, Centre Suisse de Recherches Scientifiques, [16]Institute of Cognitive Science, University of Osnabrück, [17]School of Human Evolution and Social Change, Arizona State University, [18]Institute of Human Origins, [19]Department of Anthropology, University College London, [20]Department of Human Origins, Max Planck Institute for Evolutionary Anthropology, [21]GMERC, LTD. (Greater Mahale Ecosystem Research and Conservation Project), [22]Department of Primate Behavior and Evolution, Max Planck Institute for Evolutionary Anthropology, [23]Gashaka Primate Project, [24]Institut de Biologie, Université de Neuchâtel, [25]School of Psychology and Neuroscience, University of St Andrews, [26]Senckenberg Museum of Natural History Görlitz, [27]International Institute Zittau, Technische Universität Dresden, [28]German Centre for Integrative Biodiversity Research (iDiv) Halle-Jena-Leipzig

Abstract

Camera traps are indispensable tools for wildlife conservation, but the vast volume of data they generate creates a significant bottleneck in manual processing. We present Zamba, an open-source Python package that leverages machine learning to streamline camera trap data analysis. Unlike most existing tools that support only image data, Zamba enables species classification for both still images and videos, as well as depth estimation from videos. It also supports custom model training on user-provided data, allowing adaptation to new species and geographic regions not covered by pretrained models. Zamba powers Zamba Cloud, a no-code web application that makes these capabilities accessible to non-programmers. By enabling conservationists to efficiently analyze large datasets and train models tailored to their specific ecological contexts, Zamba advances wildlife monitoring, research, and evidence-based conservation decision-making.

Keywords camera traps, computer vision, machine learning, wildlife conservation, species classification, depth estimation

Published Jul 10, 2025

Correspondence to
Emily Dorne
emily@drivendata.org

Open Access

Copyright © 2025 Dorne *et al.*. This is an open-access article distributed under the terms of the Creative Commons Attribution 4.0 International license, which enables reusers to distribute, remix, adapt, and build upon the material in any medium or format, so long as attribution is given to the creator.

1. Introduction

Camera traps are automated camera systems typically triggered by motion, heat, or timers [1]. They are a widely used tool for wildlife monitoring in both research and conservation. However, they generate an immense volume of image and video data due to the nature of the data collection and scale of their deployment. It takes the valuable time of teams of experts, or thousands of citizen scientists, to manually process this data and identify videos and images of interest [2].

Automated computer vision systems offer a natural solution to assist with this data processing. Yet, two key gaps remain in the ecosystem for wildlife camera trap data. First, available domain-specific tools only support processing image data, not video [3], and it is not computationally feasible to predict on all frames in a video as if they were images. Moreover, models that process video natively can leverage inter-frame information—such as motion—that is otherwise lost. Second, pretrained models fail to capture the wide range of environments and use cases for which camera traps are deployed. Wildlife monitoring projects vary widely in habitat, focal species, deployment context, and taxonomic resolution, necessitating adaptable models that can be fine-tuned to specific contexts.

We present Zamba [4], an open source Python package for using machine learning to process camera trap data. Zamba supports species classification for both videos and still images. Zamba also supports custom model training on user-provided camera trap data containing new species and new geographies not in the included pretrained models. In addition, Zamba includes inference-only video processing pipelines for depth estimation. Zamba also underlies Zamba Cloud, which is an accessible no-code web application that runs the python package on managed cloud resources. Zamba Cloud bridges a critical gap between the technical feasibility of training models and the practical ability for conservationists — who are often not programmers — to use them.

By enabling conservationists to efficiently process large datasets and train custom models suited to their unique ecological contexts and research questions, Zamba accelerates wildlife monitoring, research, and evidence-based conservation decision-making.

This paper provides an overview of Zamba's contributions to the conservation community and details its underlying methodology. The motivation section illustrates the value of camera trap data for conservation efforts as well as the current bottlenecks. The development section reviews Zamba's history. The following three sections detail the methods and performance for the video species classification, image species classification, and depth estimation models. We then discuss how Zamba Cloud provides an accessible no-code way to access Zamba's classification capabilities. The discussion section concludes with guidance on common workflows.

2. Motivation

Camera traps hold immense potential for conservationists as they are a noninvasive way of monitoring wildlife [5]. They enable the measurement of species occupancy [6], relative [7] or absolute abundance [8], population trends over time [9], patterns in animal behavior [3], [10], and more. They are also one of the few tools for monitoring multiple species simultaneously [11].

As simple sensors, camera traps do not automatically label or filter the species they observe. Manual review and labeling represent a critical bottleneck, consuming vast amounts of resources that could otherwise support direct conservation actions [2], [12].

One of the most common needs is filtering out "blank" images and videos, i.e., those that do not contain an animal but rather were a false trigger, potentially caused by wind, rain,

or changes in light. These "blanks" account for up to 70% of captured images [13] in some datasets. Estimating animal distances in camera trap footage currently entails an extremely manual, time-intensive process. Automated distance extraction is estimated to be 21 times faster than manual labelling [14].

While more and more conservation organizations and researchers are collecting and storing camera trap data, many are not yet able to effectively use that data for decision-making. The people-power needed to view and annotate camera trap data is prohibitively expensive in terms of time and money, which results in organizations not using camera traps; data accumulating on hard drives; or long delays in turn-around while teams sift through thousands of incoming images and videos every month.

Machine learning (ML) models have shown significant promise in automating species identification from camera trap imagery [3]. Several software tools that leverage ML have gained widespread adoption in recent years, including MegaDetector [13], Wildlife Insights [15], and Addax AI (formerly EcoAssist) [16]. These tools have advanced the application of pretrained computer vision models for animal detection and species classification particularly for camera trap image data.

Accurate, accessible, and automated species detection enables improved monitoring of animal populations and evaluation of conservation impacts on species abundance. Faster processing of camera trap data allows conservationists to conduct these assessments in weeks instead of years. This not only supports timely evaluations and interventions—allowing for adaptive management if efforts are not proving effective—but also enables the collection of more data from a broader range of locations. It also allows conservationists to direct their time toward more complex secondary analyses and making evidence-based conservation decisions.

3. Development history

Zamba's development has its origins in online machine learning challenges hosted by DrivenData in partnership with wildlife conservation experts—in particular from the Max Planck Institute for Evolutionary Anthropology (MPI-EVA). Machine learning competitions have proven effective at harnessing community-driven innovation and enabling the rapid testing of thousands of approaches to solving a problem [17]. Zamba, named after the word for "forest" in Lingala[1], was created to make these advances more accessible to ecologists and conservationists.

2017: Pri-matrix Factorization Challenge and Zamba v1

In the Pri-matrix Factorization Challenge [18], hosted by DrivenData and MPI-EVA, over 320 participants competed to develop accurate species detection models for camera trap video data. The competition used a unique dataset created through the Chimp&See Zooniverse project [2], where thousands of citizen scientists manually labeled camera trap videos. Their efforts produced nearly 2,000 hours of annotated footage from the Chimp&See database.[2] The winning algorithm achieved 96% accuracy in detecting wildlife presence and 99% average accuracy in classifying species across 23 label classes [19] and formed the foundation of the initial version[3] of Zamba, available through a command-line interface (CLI).

[1] Lingala is a Bantu language spoken widely across Central Africa, particularly in the Democratic Republic of the Congo and the Republic of the Congo.

[2] Since this data was non-expert labeled, certain thresholds on how many user annotations were required to accept a label as well as thresholds related to percentages of user agreement were applied to go from raw annotations to a well-labeled dataset.

[3] Since Zamba's origins in 2017, computer vision techniques have evolved significantly. The original algorithm implemented in Zamba v1 used an ensemble of five models with architectures based on convolutional neural networks (CNNs)—ImageNet—ResNet50, ResNet152 [20], InceptionV3 [21], Xception

2018: Zamba Cloud

Zamba was further developed into a web application—Zamba Cloud—to promote wider accessibility and adoption. Zamba Cloud enables users to run Zamba on high-performance, cloud-based infrastructure through an easy-to-use web interface.

2021: Zamba v2

Zamba underwent a major re-architecture to leverage recent advances in computer vision, released as Zamba v2.0.0. Its models were also retrained on an expanded dataset of 250,000 labeled camera trap videos.[4] This update also introduced support for fine-tuning custom models on videos and incorporated a smarter frame selection strategy using a newly trained machine learning model called MegadetectorLite.

2021: Deep Chimpact Challenge and depth estimation

DrivenData partnered again with the Max Planck Institute for Evolutionary Anthropology and the Wild Chimpanzee Foundation to host the Deep Chimpact: Depth Estimation for Wildlife Conservation challenge [25]. More than 300 participants competed to develop models for monocular depth estimation—inferring the distance between camera traps and wildlife appearing at various moments in video footage. In 2022, the second-place model from this challenge was integrated into Zamba as an inference module.

2024–2025: Species classification for images

In March 2025, species classification for images was added in Zamba v2.6.0 and Zamba Cloud. This major update, supported by the WILDLABS Awards 2024, focused on enabling non-programmer users to train custom image classification models through Zamba Cloud's no-code interface. The work involved aggregating diverse image datasets, experimenting with model architectures, and training a robust base model spanning 178 taxonomic label classes.

4. Classifying species in videos

Camera trap videos are especially valuable for researchers studying animal behavior, as they capture rich information including sounds, movements, tool use, and interactions between individuals [26], [27], [28]. Videos also provide multiple views of the same individual, which can aid in identifying characteristics such as sex, size, and age [2], [29], [30]. It can also enable individual identification, which is useful for capture-recapture studies [31].

However, video classification is more challenging than image classification. It demands more complex model architectures, increased computational resources such as graphics processing units (GPUs), and the ability to manage and process datasets often measured in terabytes. Zamba initially focused on videos in order to address the gap in the tooling ecosystem, as other automated species classification tools only supported still images.

In the sections that follow, we describe Zamba's approach to species classification in videos. This includes the core methodologies, the available pretrained models and features, and performance metrics based on evaluation on a holdout set.

---

[22], and Inception-ResNetV2 [23]. Base models trained on ImageNet were fine-tuned. Then, class probabilities for 32 frames per clip were aggregated into summary statistics and passed to a second-level XGBoost [24] model, which generated the final species prediction. [19] This model was replaced in 2021 with the release of Zamba v2 so is not discussed in more depth here.

[4] This retraining update also addressed artifacts from the original competition. For example, the competition's lack of site-aware splitting contributed to inflated performance metrics by allowing models to learn location-specific features from backgrounds. Additionally, labeling errors stemming from crowd-sourced annotations were corrected using a curated set of 125,000 expert-labeled videos, which significantly improved performance for certain species.

4.1. Methods

4.1.1. *Training data*

A large and diverse dataset is one of the most important factors influencing model accuracy for camera trap species classification. Zamba's video models were trained on a dataset of 250,000 labeled camera trap videos. Half of this data was collected and annotated by trained ecologists, while the other half consists of high-confidence labels generated by citizen scientists through the Chimp&See platform. The training data comes from camera trap deployments in the following countries: Cameroon, Central African Republic, Democratic Republic of the Congo, Gabon, Guinea, Liberia, Mozambique, Nigeria, Republic of the Congo, Senegal, Tanzania, and Uganda.

Table 1. *Locations included in the training data for the video species classification models*

Country	Location
Cameroon	Campo Ma'an National Park
	Korup National Park
Central African Republic	Dzanga-Sangha Protected Area
Côte d'Ivoire	GEPRENAF–Comoé
	Djouroutou
	Taï National Park
Democratic Republic of the Congo	Bili-Uele Protected Area
	Salonga National Park
Gabon	Loango National Park
	Lopé National Park
Guinea	Bakoun Classified Forest
	Moyen-Bafing National Park
Liberia	East Nimba Nature Reserve
	Grebo-Krahn National Park
	Sapo National Park
Mozambique	Gorongosa National Park
Nigeria	Gashaka-Gumti National Park
Republic of the Congo	Conkouati-Douli National Park
	Nouabalé-Ndoki National Park
Senegal	Kayan
Tanzania	Grumeti Game Reserve
	Ugalla River National Park
Uganda	Budongo Forest Reserve
	Bwindi Forest National Park
	Ngogo, Kibale National Park

A carefully considered train–test split strategy is critical for camera trap data. Because each camera remains fixed in place, many videos share identical backgrounds. Without careful splitting, models risk overfitting to site-specific visual features rather than generalizing to animal appearance. To address this, Zamba uses a site-aware split strategy, where all videos from a single camera location are placed entirely in either the training or test set.

Species labels in the dataset were standardized into 32 output classes[5] by manually grouping vernacular species names used in expert-labeled videos into a fixed set of target classes. This aggregation provided enough training examples per class while preserving a level of taxonomic granularity useful to most users out of the box. Although the 32 classes cover only a small portion of species studied in conservation research worldwide, Zamba also supports training custom models to cover additional species, habitats, and taxonomic ranks. See "Custom model training" for further discussion.

Table 2. *Example vernacular names that were all mapped to the target class* `antelope_duiker`

antilope spp.	bongo	bushbuck	dikdik	duiker
eland	gazelle	grantsgazelle	hartebeest	impala
jentink's duiker	kudu	maxwell's duiker	nyala	ogilby's duiker
oribi	peter's duiker	red duiker	reedbuck	royal antelope
sable	sitatunga	thompsonsgazelle	topi	water chevrotain
waterbuck	wildebeest	white bellied duiker	yellow backed duiker	zebra duiker

An additional training dataset of European wildlife was used to fine-tune a model specialized for European species. This dataset includes camera trap videos from Hintenteiche bei Biesenbrow, Germany [32].

4.1.2. *Classification model architecture*

Zamba includes four pretrained species classification video models that implement one of the two architectures: `time_distributed` or `slowfast`.[6]

The `time_distributed` model architecture is based on EfficientNetV2 [38]. EfficientNetV2 models are convolutional neural networks designed to jointly optimize model size and training speed. EfficientNetV2 is image-native, meaning it operates on each frame individually. The model is wrapped in a `TimeDistributed` layer [39], which aggregates frame-level predictions into a single video-level prediction.

The `slowfast` model architecture uses the SlowFast [40] backbone for video classification. This model is named for its two parallel pathways: a slow pathway that processes low frame-rate inputs to capture spatial semantics, and a fast pathway that processes high frame-rate inputs to capture motion dynamics. As a video-native architecture, SlowFast models consider temporal relationships between frames, which can be especially valuable for detecting occluded animals that may only become apparent through movement.

Table 3 provides an overview of the models. The `time_distributed` and `slowfast` models are trained to classify 32 species or species groups common to Central and West Africa.[7] The `european` model is trained to classify 11 common species in Western Europe.[8] Each model has distinct relative strengths. For example, the `slowfast` model trained on African species may perform better for small-bodied animals like rodents than the `time_distributed` African

[5] These output classes were selected based on the research needs of the data providers from the Max Planck Institute for Evolutionary Anthropology.

[6] Other architectures evaluated during development in 2021 included image-based and video-based models such as ResNet [33], R2Plus1D [34], TimeSFormer [35], X3D [36], and I3D [37].

[7] The label classes for the `time_distributed` and `slowfast` African species models are: aardvark, `antelope_duiker`, `badger`, `bat`, `bird`, `blank`, `cattle`, `cheetah`, `chimpanzee_bonobo`, `civet_genet`, `elephant`, `equid`, `forest_buffalo`, `fox`, `giraffe`, `gorilla`, `hare_rabbit`, `hippopotamus`, `hog`, `human`, `hyena`, `large_flightless_bird`, `leopard`, `lion`, `mongoose`, `monkey_prosimian`, `pangolin`, `porcupine`, `reptile`, `rodent`, `small_cat`, `wild_dog_jackal`

[8] The label classes for the `european` model are: `bird`, `blank`, `domestic_cat`, `european_badger`, `european_beaver`, `european_hare`, `european_roe_deer`, `north_american_raccoon`, `red_fox`, `weasel`, and `wild_boar`

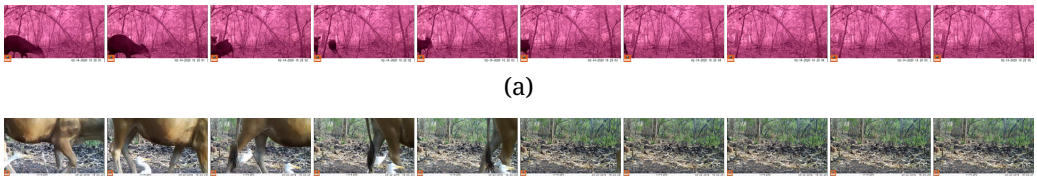
species model. There is also a `european` model, fine-tuned from the `time_distributed` African model, that is trained on species from non-jungle ecologies in Western Europe. All models include a `blank` class to identify videos with no animals, but the dedicated `blank_nonblank` model may provide higher accuracy for blank detection alone.

Table 3. *Summary of Zamba's pretrained video models*

Model	Geography	Relative strengths	Architecture	Number of training videos
`blank_nonblank`	Central Africa, West Africa, and Western Europe	Just blank detection, without species classification	Image-based `TimeDistributedEfficientNet`	~263,000
`time_distributed`	Central and West Africa	Recommended species classification model for jungle ecologies	Image-based `TimeDistributedEfficientNet`	~250,000
`slowfast`	Central and West Africa	Potentially better than `time_distributed` at small species detection	Video-native `SlowFast`	~15,000
`european`	Western Europe	Trained on non-jungle ecologies	Finetuned `time_distributed` model	~13,000

4.1.3. *Frame selection approach*

One of the key technical challenges in working with camera trap videos—rather than still images—is frame selection. For datasets that motivated the development of Zamba, videos had a frame rate of 30 frames per second and were typically about 60 seconds long. Processing every frame as an image for such videos is computationally infeasible. Furthermore, animals may only be present in a minority of recorded frames, and selecting frames without the animal degrades the performance of downstream tasks like species classification and depth estimation.



(a)

(b)

Figure 1. *Excerpts from two 60-second example videos in which animals appear only briefly. Each excerpt shows the first 5 seconds; the remaining 55 seconds contain an unoccupied field of view.*

The default frame selection method currently used by Zamba is an efficient object detection model we developed called MegadetectorLite. This model evaluates each frame for the likelihood that it contains an animal. By default, the top 16 frames[9] with the highest detection probabilities are selected and passed to either the species classification or blank detection models.

[9]This parameter was found empirically via crowd-sourced experimentation by participants in the Primatrix Factorization Challenge to be sufficient for accurately classifying videos while balancing the overall computational requirements of the pipeline. Users may choose to tune this parameter for their particular dataset and use case. See the "Configuration and customization" section.

Figure 2. *Flow diagram of the video species classification pipeline. Gray nodes indicate data and blue nodes indicate a processing step.*

Because frame selection is an upstream task, it must be fast without sacrificing too much accuracy. Larger models tend to be more accurate but are slower to run. MegadetectorLite is based on the YOLOX architecture [41] and was trained using a knowledge distillation approach [42] (also known as teacher–student training). It uses the predictions of the original MegaDetector [13] as supervisory labels. MegaDetector itself is a highly accurate, but computationally intensive, object detection model trained specifically on camera trap imagery. While MegaDetector is used directly in Zamba's image workflows, it is too slow for frame-by-frame inference in video processing. MegadetectorLite offers a more efficient alternative with a lightweight architecture.

We experimented with two YOLOX model variants (yolox-nano and yolox-tiny) and two input image sizes (416 and 640 pixels). We ultimately selected yolox-tiny with an input size of 640 pixels as the best tradeoff between speed and accuracy.

MegadetectorLite was trained on over 1 million frames. To improve detection of rare species —which have historically underperformed in automated systems—the training dataset was enriched with extra frames from videos featuring less frequently observed animals, including hyenas, leopards, aardvarks, reptiles, bats, giraffes, lions, cheetahs, and badgers.

4.2. Results

4.2.1. *Pretrained model performance*

Metrics for camera trap tasks are highly dataset-dependent and can vary widely depending on deployment context and class balance. Here, we report some illustrative evaluation metrics for Zamba's pretrained models using holdout validation.

The holdout set comprises a random sample of labeled videos, selected on a transect-by-transect basis. That is, all videos from a given transect (camera location) are assigned entirely to either the training or holdout set. While not all use cases require this site-aware split, this is a stricter evaluation to performance on transects the model has never seen.

The holdout set includes data from all 14 countries in the training dataset, with country-level proportions roughly matching those of the complete set. Figure 3 shows the distribution of videos across 30 animal species, as well as a substantial number of blank videos and some containing humans.

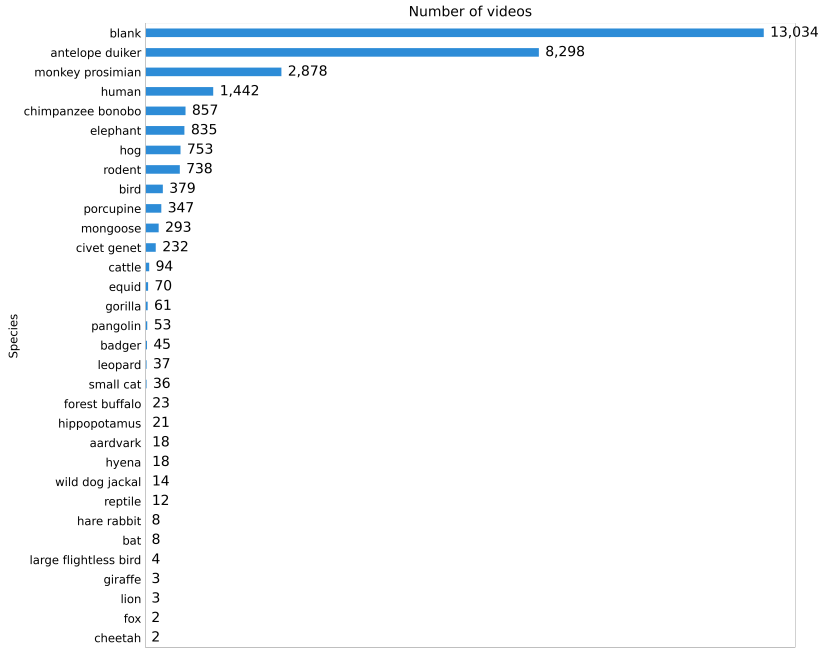


Figure 3. *Distribution of label classes in video model holdout set.*

Species classification

On the holdout set, African species `time_distributed` model achieves a top-1 accuracy[10] of 82% and a top-3 accuracy[11] of 94%. Performance varies across regions depending on species diversity. For example, in the limited sample of videos from Equatorial Guinea, only three species are represented. In this case, top-1 accuracy is 97%, significantly above the global average. In contrast, Ivory Coast videos contain 21 species, making classification more challenging; here, top-1 accuracy drops to 80%.

Figure 4 shows precision and recall by class for the holdout set. Eleven species with too few examples were excluded from this analysis. In general, recall tends to be higher for larger animals. However, even for species with lower recall, it is still feasible to use predicted probabilities to flag likely candidates for further review. For example, rodents have a recall of only 22%, but sorting videos by the model's predicted probability for the "rodent" class remains a useful heuristic—even if other classes have higher probabilities predicted.

[10] Top-1 accuracy is defined as the percentage of videos where the class with the highest predicted probability is actually present.

[11] Top-3 accuracy is the percentage where the true class appears among the top three predicted classes.

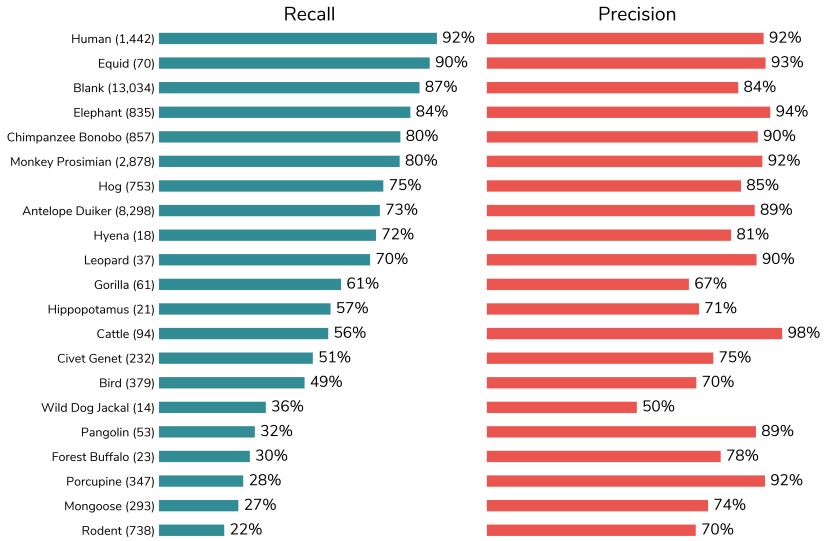


Figure 4. *Precision and recall by class for the holdout set.*

Model performance tends to be lower for:

- Rare species with limited training data
- Small-bodied animals, especially since the video models classify the entire frame without cropping around the subject
- Heavily occluded or distant animals, or those only partially visible or present for just a few frames
- Low-quality media, such as videos that are washed out or too dark

Because of a lack of comparable wildlife camera trap tools that perform species classificaiton for videos, it is difficult to provide a direct comparison to other results. Users are always encouraged to evaluate models on their own data. Additionally, Zamba supports fine-tuning custom models which can often be the best choice for achieving strong performance.

Blank detection

Blank detection is an important feature in camera trap workflows due to the high prevalence of blank videos. In our holdout set, 42% of videos contained no animal activity. The `blank_nonblank` model achieved 87% recall (correctly identifies 11,288 of 13,034 blank videos) and 84% precision (13,507 videos predicted as blank, of which 11,288 were correct).

4.2.2. *Applications to out-of-domain data*

While Zamba's pretrained models are trained to predict specific classes for species from Africa and Western Europe, it can be possible to effectively apply them to out-of-domain data that includes geographies or species that the model has never seen. In DrivenData [43], we show a case study using a dataset of 3,056 videos from New Zealand. Using the pretrained African species `time_distributed` model, the `bird` label correctly captures 77% of the takahē videos (recall) at 82% precision, which is particularly notable since the takahē is a New-Zealand-native flightless bird that is not present in the African training data. However, other New Zealand-native birds like the weka are less successfully captured by the `bird` category (44% recall). Training custom models is recommended to achieve more accurate classification on new geographies and species.

4.2.3. *Configuration and customization*

All Zamba workflows—inference, fine-tuning, and training—support an extensive range of configuration options. For instance, processed videos are resized to configurable size, by default 240x426 pixels. Higher resolution videos will lead to superior accuracy in prediction, but will use more memory and take longer to train or predict.

Frame selection is another key configurable option. In addition to the default MegadetectorLite method, users can choose from:

- Evenly distributed frames
- Early-biased frame sampling
- Key frame extraction
- Scene-change-based selection

The number of frames selected is also customizable (default: 16).

Configurations can be provided via command-line flags or through YAML-formatted configuration files.

4.2.4. *Custom model training*

One of Zamba's signature features is its support for training custom models. Custom models are primarily built by fine-tuning general-purpose species classifiers and can enhance performance in specific ecological contexts—even when classifying the same species. Custom models are also essential for extending classification to new species, new habitats, or label classes at different taxonomic ranks.

Zamba supports two fine-tuning scenarios based on label compatibility:

- If the new training data uses a subset of the model's existing classes, Zamba continues training with the existing classification head.
- If the training data includes new or different label classes, Zamba automatically replaces the model's classification head (i.e., the final neural network layer) before continuing training.

5. Classifying species in images

While Zamba began with a focus on video data, the majority of camera trap users today collect still images. Images have several practical advantages: they require less storage space, use less battery life, and are faster to transfer, review, and process. Images are often sufficient for determining presence or absence of a particular species and can be used for distance sampling [14] as well to get to abundance estimates.

There are a number of machine learning tools for processing camera trap images [3]. However, static pretrained models fail to capture the diversity in camera trap images and the range of environments in which camera traps are used around the globe.

Zamba's extension into image support was motivated by the need to bring custom model training workflows to image-based data, building on the similar capabilities already developed for videos. The ability to train accurate models with a relatively small amount of labeled data is central to Zamba's goal of enabling camera trap users to harness automated species classification for their unique datasets.

In the sections that follow, we outline the core methods Zamba employs for species classification in images, and present performance metrics for the general image classification model that forms the foundation for training custom models.

5.1. Methods

5.1.1. *Cropping*

Following the approach of T. Gadot *et al.* [44], Zamba's image classification pipeline first applies a species-agnostic object detection model to extract bounding box crops of detected animals. The image classification model then operates solely on these cropped regions, rather than processing the full-frame image.

For object detection, Zamba uses MegaDetector [13], a widely adopted open-source model designed specifically for wildlife camera trap data. MegaDetector identifies animals, humans, and vehicles in camera trap imagery and is a standard component in many ecological processing pipelines [3]. For additional technical details and performance evaluations of MegaDetector, see J. Vélez *et al.* [3].

Figure 5. *Flow diagram of the image species classification pipeline. Gray nodes indicate data and blue nodes indicate a processing step.*

5.1.2. *Training data*

Zamba's pretrained image classification model was trained on over 15 million annotations from over 7 million images from the Labeled Information Library of Alexandria: Biology and Conservation (LILA BC) data repository [45]. Each annotation corresponds to a cropped image of an animal, extracted using bounding boxes generated by MegaDetector [13].

To maximize geographic and ecological diversity, the training dataset aggregated 20 terrestrial camera trap datasets from LILA BC, representing a wide range of global habitats. (See Table 4 for a full list of datasets.) This broad coverage was designed to produce a versatile base model well-suited for fine-tuning to a wide variety of ecological contexts.

Table 4. *Datasets from LILA BC included in training data*

Dataset	Geography	Count of original images	Count of cropped annotations
Caltech Camera Traps [46]	Southwestern United States	59,205	96,724
Channel Islands Camera Traps [47]	California, United States	125,369	239,472
Desert Lion Camera Traps [48]	Namibia	61,910	185,475
ENA24-detection [49]	Eastern North America	8,652	11,092
Idaho Camera Traps [50]	Idaho, United States	338,706	1,072,912
Island Conservation Camera Traps [51]	7 islands around the world	44,007	79,660
Missouri Camera Traps [52]	Missouri, United States	946	955

North American Camera Trap Images [53]	United States	2,705,394	7,426,839
New Zealand Trailcams [54]	New Zealand	2,109,592	2,794,859
Orinoquia Camera Traps [3]	Colombia	80,307	103,856
Snapshot Safari 2024 Expansion [55]	Africa (multiple countries)	836,522	1,949,366
Snapshot Safari Camdeboo [55]	South Africa	15,299	26,379
Snapshot Safari Enonkishu [55]	Kenya	9,049	37,252
Snapshot Safari Karoo [55]	South Africa	5,764	8,426
Snapshot Safari Kgalagadi [55]	South Africa and Botswana	2,060	2,938
Snapshot Safari Kruger [55]	South Africa	3,112	6,343
Snapshot Safari Mountain Zebra [55]	South Africa	5,535	9,333
SWG Camera Traps [56]	Vietnam and Laos	87,309	100,677
WCS Camera Traps [57]	12 countries	523,897	920,471
Wellington Camera Traps [58]	New Zealand	203,038	269,146

A key challenge in using data from these disparate sources was the lack of a unified taxonomy across datasets. Each dataset was collected independently, often for different research purposes, and the taxonomic granularity varies significantly. For example, some datasets label all birds under a generic “bird” class, while others distinguish specific bird species.

To address this, we reviewed 1,231 label classes across the source datasets and curated a unified taxonomy with 178 label classes. These classes span a variety of Linnaean taxonomic ranks, with each class having a minimum of 100 annotations to ensure sufficient training examples. As with the video models, these label classes represent a narrow subset of the wildlife of interest to conservationists. This model is intended to primarily serve as a base model for fine-tuning custom models, rather than serving as a general-purpose model that directly covers conservation use cases.

As with the video classification models, a split-aware strategy was used to create training, validation, and test sets. All data from a given camera location were assigned to the same

split to avoid overfitting to background features and ensure generalizable model performance.

5.1.3. *Classification model architecture*

In order to select a backbone architecture, we evaluated several modern neural network architectures that post-date the commonly used backbones in most existing camera trap models. The candidate architectures included ConvNeXt, BEiT, EfficientNet, LeViT, and ViT.

Figure 6 shows run curves for the different backbone architectures we evaluated. Runs for ConvNeXt variants outperformed others, achieving greater accuracy with fewer training epochs. Based on these results, we chose ConvNeXt V2 base as the backbone.

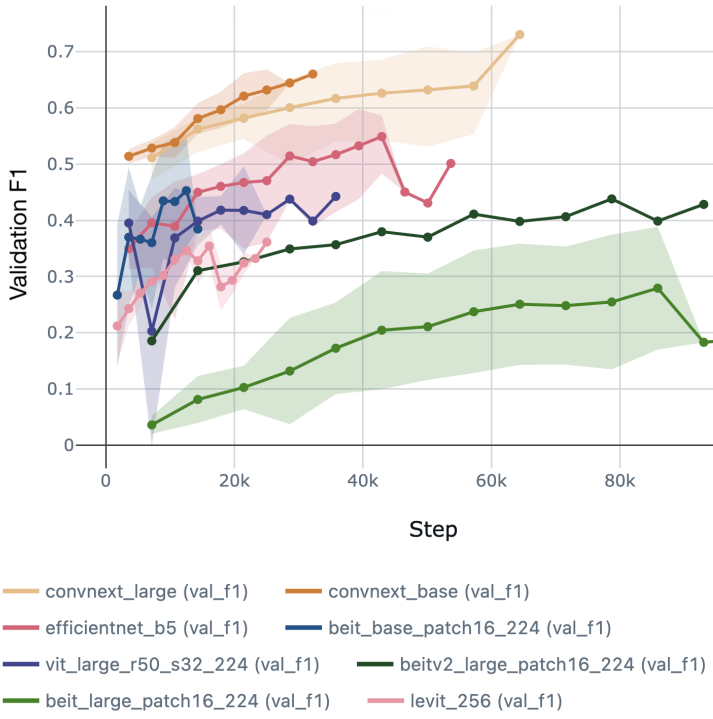


Figure 6. *Validation F1 training run curves for candidate model architectures. The filled band shows the min and max range across runs, and the line shows the average across runs.*

The chosen ConvNeXt V2 base model contains 87.7 million parameters and operates on 224 $\times$224 pixel input crops. In Figure 6, its performance is represented by the dark orange curve.

5.2. Results

5.2.1. *Pretrained model performance*

As with videos, metrics for camera trap tasks are highly dataset-dependent and can vary widely based on deployment context and class balance. Here, we report some illustrative metrics for Zamba's pretrained image model `lila.science` using holdout validation.

The holdout set reported here contains 449,263 randomly sampled observations held out from training. Splits were performed on a transect-by-transect basis[12] while ensuring at least one transect representing each label class is present in all splits. The holdout set is

[12]This means all images from a given transect (camera location) are assigned entirely to one split. This methodology is the same as what we used for splitting the video data.

highly unbalanced, with a exponential-like distribution of class sizes. Figure 7 shows an empirical cumulative distribution function (ECDF) of the number of observations in each class.

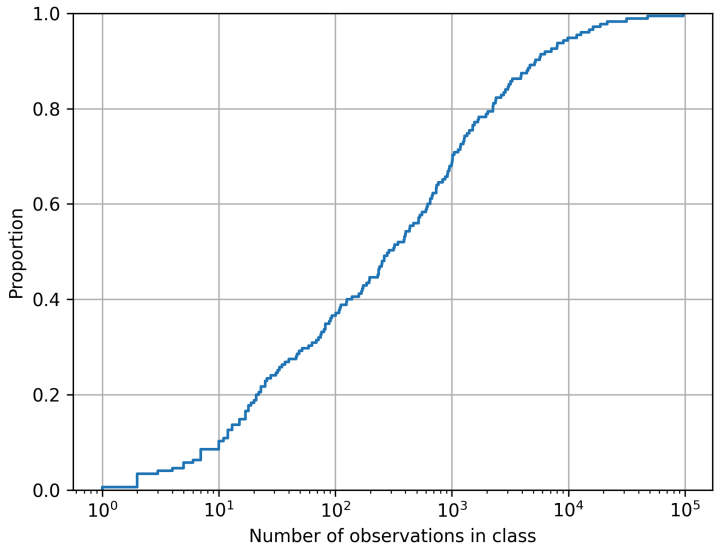


Figure 7. *ECDF of showing the distribution of class sizes. The curve shows the proportion of the label classes with up to the number of observations in the holdout set given by the x-axis value. Note that the x-axis is shown in log-scale.*

On the holdout set, the `lila.science` model achieved 90% top-1 accuracy. Performance varies widely by label class, although the majority of label classes see relatively strong performance. Figure 7 shows the distribution of F1 scores across label classes for classes with at least 30 observations. Of those (133 classes), 40% (53 classes) have an F1 score of over 0.9 and 71% (94) have an F1 score of over 0.8.

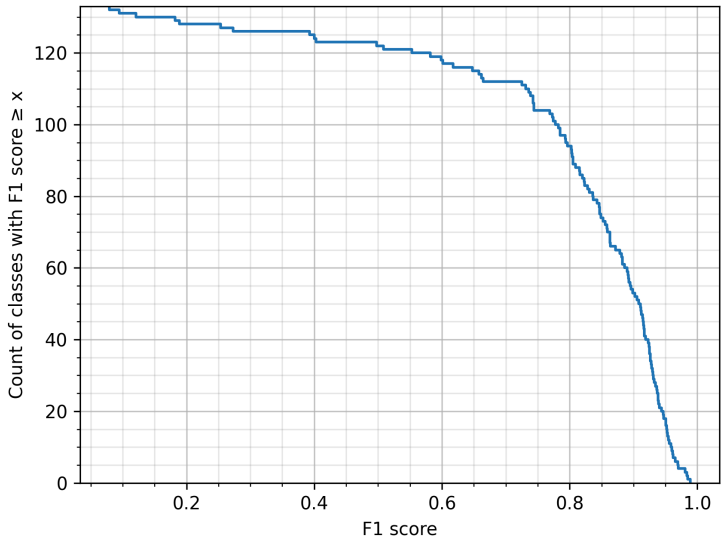


Figure 8. *Complementary ECDF of F1 score for label classes in the holdout set with at least 30 observations. The curve shows the count of label classes whose F1 score exceeds the value given by the x-axis.*

Figure 9 shows the precision and recall scores for top 10 and bottom 10 label classes when ranked by F1 score. This shows that the model performs highly accurately on the top performing label classes. For the weakest label classes, there is a mix of strong-precision–weak-recall, weak-precision–strong-recall, or weak performance in both metrics. The weakest classes tend to be classes with the fewest observations in the test set (<100), although some classes with many observations like `hippotragini` and `columbimorph_bird` with thousands of test observations show relatively low recall scores. Cases of weak performance are often driven by confusion between phenotypically similar classes; for instance, observations involving types are birds are often predicted at different taxonomic granularity from the label.

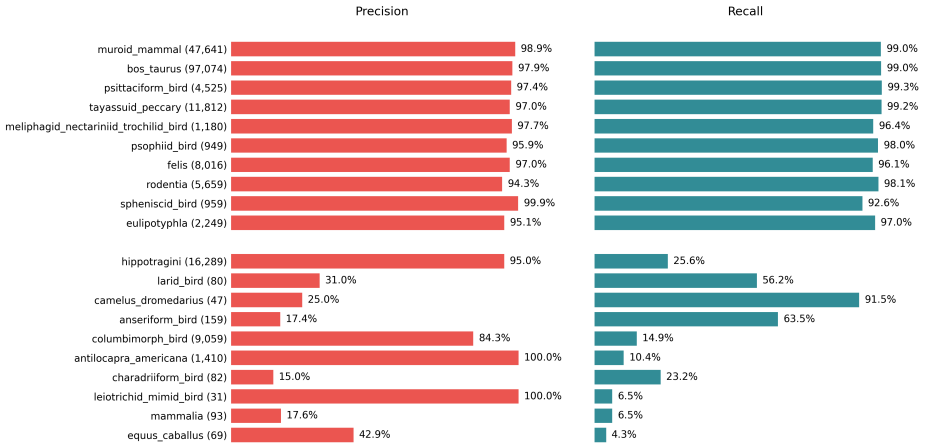


Figure 9. *Precision and recall values for a selection of label classes. The top half are the top 10 label classes ranked by F1 score, while the bottom half are the bottom 10 label classes ranked by F1 score. The parenthetical annotation gives the number of observations of that class.*

For an indirect comparison[13], SpeciesNet, the model used by Wildlife Insights, report a 81.9% accuracy, 80.7 species-weighted-average F1 score, and 54.2 macro-averaged F1 score on a test dataset of 42,791 images with 101 label classes[44]. This comparison shows that Zamba's pretrained model is in an approximate comparable range of performance as the published results of similar tools.

6. Depth estimation for videos

Statistical models can be used to estimate animal abundance from camera trapping [59]. One such method is the distance sampling framework [14], which combines the frequency of animal sightings with the distance from the camera to each animal to estimate a species' full population size [8]. In the following sections, we outline the training data, model architecture, and performance of Zamba's depth estimation approach for videos.

6.1. Methods

As discussed in the "Development History" section above, the Deep Chimpact machine learning challenge was held to develop and train models for performing depth estimation on monocular camera trap videos.

[13] As discussed for video species classification, it is difficult to report meaningful direct comparisons to other tools or study results. There are not well-established standard benchmarking procedures and datasets, and the high degree of variance in attributes across camera trap deployments means that performance is highly sensitive to similarity to the training data, and that metrics reported on one dataset do not necessarily translate well to others.

The top performing model from the second place ensemble[14] was integrated into Zamba as an inference-only module.[15] As no additional retraining was conducted, the below sections describe the competition data and results.

6.1.1. *Training data*

Zamba's depth estimation model was developed using a unique dataset of approximately 3,900 videos, created by research teams from the Max Planck Institute for Evolutionary Anthropology (MPI-EVA) and the Wild Chimpanzee Foundation (WCF). Videos were from Taï National Park in Côte d'Ivoire and Moyen-Bafing National Park in the Republic of Guinea and captured six different species groups: bushbucks, chimpanzees, duikers, elephants, leopards, and monkeys.

Ground truth distances in the dataset were manually annotated with the aid of reference videos, which are recordings of field researchers walking away from each camera trap holding a sign up every meter to indicate how far they were from the camera. Such reference videos are a typical way to perform depth estimation for camera trap videos [14]; however, they are time and resource intensive to record and to use in labeling.

Figure 10. *A researcher in Taï National Park, Côte d'Ivoire holding up a sign at 5 meters away as part of a depth reference video. Image courtesy of Wild Chimpanzee Foundation.*

Videos in the original dataset that included more than one animal were excluded as there was not a way in the labeled data to identify which distance label corresponded to which animal. This removed 18% of frames from the original Taï National Park dataset and 22% of frames from the original Moyen-Bafing dataset.

Data was split evenly into a training dataset and a test dataset. Each site was either entirely in the train set or the test set, so all backgrounds used to evaluate submissions were new to the model.

Table 5. *Counts per location for the final dataset used in the competition*

Park	Train set	Test set

[14]The second-place submission was chosen for modeling framework compatibility reasons. While the full competition-winning approach was an ensemble, the performance of the best single model was almost identical to the ensemble (MAE of 1.635m for the single model compared to 1.625m for the ensemble) so only one model was integrated into Zamba. This supports lower computing costs and faster inference speeds without sacrificing performance.

[15]Training and fine-tuning is not supported as of version v2.6.1 (May 2025).

Taï National Park, Côte d'Ivoire	530 videos 4,173 frames	508 videos 3,802 frames
Moyen-Bafing National Park, Guinea	1,542 videos 11,056 frames	1,328 videos 8,130 frames

Table 6. *Distribution of species labels in the train and test sets*

Species	Train videos	Train frames	Test videos	Test frames
duiker	744	6,805	695	5,718
buckbuck	261	3,550	230	2,603
chimpanzee	304	2,876	249	2,150
monkey	738	1,757	642	1,279
leopard	27	230	20	162
elephant	1	11	1	20

The videos ranged between 5 seconds and 1 minute long. Each annotation includes a timestamp in seconds since the start of the video and the ground truth distance in meters.

Table 7. *Example rows of depth estimation annotations*

video_id	time	distance
zxsz.mp4	0	7.0
zxsz.mp4	2	8.0
zxsz.mp4	4	10.0

6.1.2. *No references videos*

The depth estimation model was not trained using any reference videos directly. The model was instead required to learn purely from monocular images and the distance ground truth labels. This is a more challenging task but enables more complete automation of the workflow.

6.1.3. *Model architecture*

The depth estimation pipeline applies the following steps. First, videos are resampled to one frame per second to match the training dataset. Next, an object detection model is used on each frame to find frames with animals in them. The object detection model used is MegadetectorLite. Then, the depth estimation model estimates distance between the animal and the camera for each frame.[16]

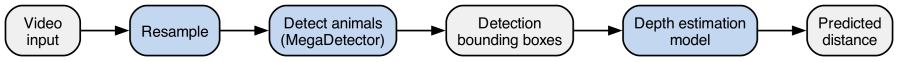


Figure 11. *Flow diagram of the video depth estimation pipeline. Gray nodes indicate data and blue nodes indicate a processing step.*

The depth estimation model uses an EfficientNetV2 [38] backbone and the output is a scalar estimated distance in meters between the animal and the camera. The inputs to the model are 5 frames stacked channelwise, where the frames are chronological and the middle frame corresponds to the present frame for which the depth estimation is made. The frames are downsampled to 270x480. The model was trained with heavy augmentations for 80

[16] Although the model was trained on single-individual videos, it produces an output for each detected animal in a frame. However, for frames with multiple individuals, the distance for all animals will be the same. If there is no animal in the frame, the distance will be null.

epochs with a batch size of 32, AdamW optimizer, and CosineAnnealingLR scheduler with a period of 25 epochs.[17]

6.2. Results

Model performance is reported for the test set of nearly 12,000 frames described above. The distribution of distances is shown in Figure 12. Nearly half (45%) of frames had a labeled distance of 5 meters or less and 80% of frames had a labeled distance of 10 meters or less.

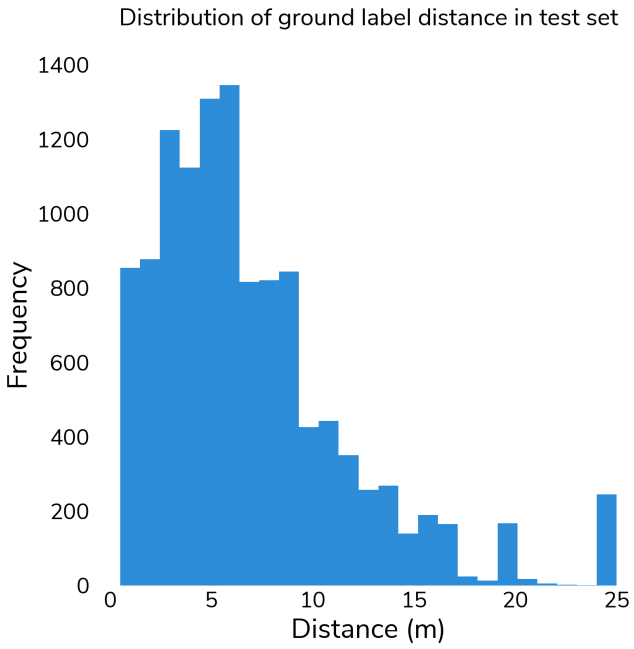


Figure 12. *Distribution of distances in the depth estimation test set*

The depth estimation model has a mean absolute error (MAE) of 1.635 m. The model is more accurate when predicting depth for animals closer to the camera. The MAE for animals ≤10 m from the camera trap is 1.06 m, while at further distances, the model underestimates distance. However, accuracy at closer distances is more important for distance sampling methods [60].

T. Haucke, H. S. Kühl, J. Hoyer, and V. Steinhage [14] applied machine learning to monocular depth *using reference videos* and reported a mean absolute error (MAE) of 1.85 m. The relatively similar performance (albeit on different datasets) suggests that accurate machine learning predictions are possible without reference videos.

[17] These hyperparameters were part of the second-place winning solution of the Deep Chimpact Challenge and found empirically.

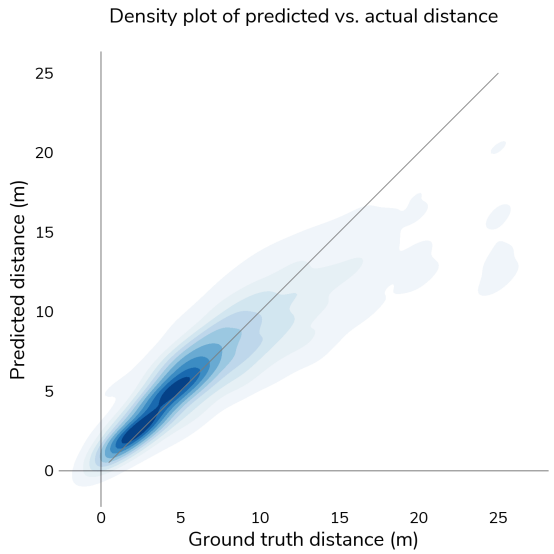


Figure 13. *Density plot of predicted vs. actual distance for the depth estimation model*

The median absolute error was relatively similar across species, though the distribution of errors differs a bit. Figure 14 show the interquartile range (shaded portion) of absolute errors by species. Outliers extending beyond 1.5 times the interquartile range are not plotted.

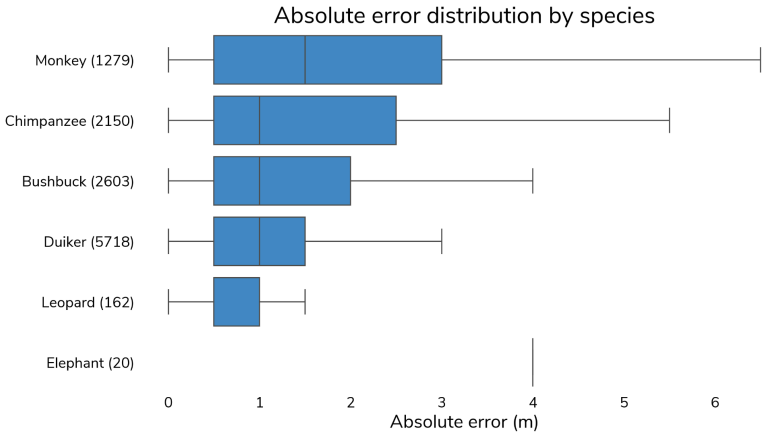


Figure 14. *Distribution of test set absolute error by species.*

7. Zamba Cloud

As a command-line program and Python library distributed as a package, Zamba is capable and easily deployable. However, there are still two major limitations. First, not all researchers have such hardware available with graphical processing units (GPUs) to run deep learning models in a practical way. Without this specialized hardware running Zamba's models on large datasets of videos or images is inefficient. Secondly, many wildlife conservationists are not programmers and may have little experience with command-line programs or scripting in Python, and it can be a barrier to create the environment and install the dependencies to run these tools.

In order to make Zamba easy to use and accessible to any camera trap researcher, a point-and-click graphical-user interface program was developed. Zamba Cloud is a web application with no-code workflows to use Zamba's capabilities. Zamba Cloud currently has workflows that support the species classification for videos and species classification for images tasks. For both tasks, users can fine-tune from any of Zamba's pretrained models, and they can perform inference on unlabeled videos/images using a pretrained model, one of their fine-tuned models, or a fine-tuned model that another user has chosen to make publicly available.

All of the workflows start with uploading videos or images using a drag-and-drop interface. Alternatively, uploads via an FTP server are also supported. One key tradeoff of implementing a web application, as opposed to a desktop application, is that users are required to have internet access with sufficient bandwidth. To mitigate the bandwidth requirements, Zamba Cloud implements an optional upload process that performs client-side video resizing. This option uses ffmeg.wasm [61], a WebAssembly port of FFmpeg [62] to reduce the resolution and framerate of submitted videos to fixed sizes. If fine-tuning, users additionally upload label data and confirm whether they match to existing label classes or are new classes.

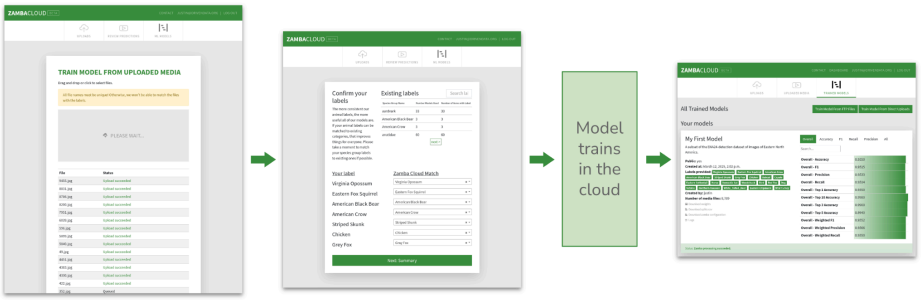


Figure 15. *Screenshots showing the custom model fine-tuning workflow in Zamba Cloud.*

Zamba Cloud runs all machine learning workloads on managed cloud infrastructure with GPUs. This means users can easily upload their data and use Zamba without needing specialized hardware or to install complex software.

For the inference workflow, Zamba Cloud has a review interface that allows users to easily visually inspect each image or video alongside predicted species classifications with confidence scores. For images, the bounding boxes from object detection are also shown. See Figure 16 for an example screenshot.

Figure 16. *Screenshot of the prediction review interface in Zamba Cloud.*

Predictions from the inference workflow are exportable in comma-separated values (CSV) format. CSV is chosen as a simple, open format that is portable and supported by common spreadsheet software.

Zamba Cloud's server is implemented in Python using Django [63]. Machine learning workloads are containerized and submitted as batch jobs to a Kubernetes cluster.

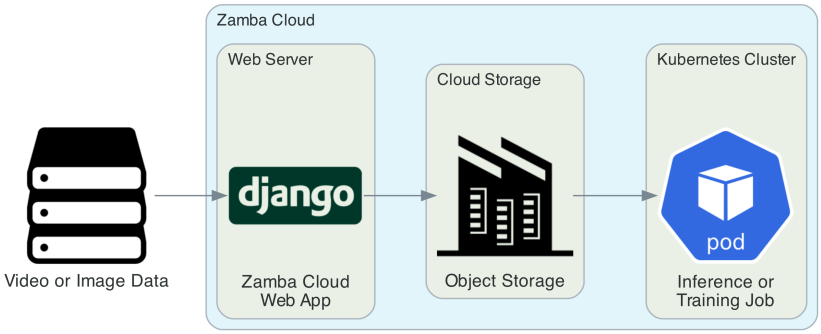


Figure 17. *Architecture diagram of Zamba Cloud.*

8. Discussion

Zamba implements workflows for inference on unlabeled videos or images using pretrained models as well as creating custom models by fine-tuning pretrained models with labeled data. Each of these tasks are available through the CLI or the Python library.

Custom model training represents Zamba's flagship contribution and unique capability among camera trap analysis tools. Camera traps are used across hugely variable habitats, taxonomic rank, and camera positions. A single general pretrained model could not accommodate fish detected by underwater deployments, large mammals in the savannah, and insects captured in a hole in the ground. Zamba's customization features dramatically expand applicability beyond users studying the species that existing pretrained models were trained on.

In addition, no-code workflows to use Zamba's capabilities on Zamba Cloud means custom machine learning models are now accessible to non-programmers. To the best of our knowledge, Zamba Cloud is currently the only tool that supports fine-tuning custom video-native machine learning algorithms for camera traps without writing any code. To date, over 300 users from around the globe have used Zamba Cloud to process more than 1.1 million videos.

8.1. Guidance on use

Zamba is designed to accelerate human workflows rather than replace human expertise, reducing the volume of videos that require manual review and removing procedural steps like identifying blank videos. Camera trap research often involves human coding (e.g., assessments of animal behavior) regardless of automated filtering or selection. At its current capability level, we recommend Zamba be used to inform research workflows rather than automate final outputs. The most common applications are:

- **Blank filtering**: Zamba can be used to filter blank camera trap videos using probabilistic classifications, saving researchers significant time and storage costs. Researchers can adjust probability thresholds based on their priorities—setting conservative thresholds to avoid losing any animal videos, or more aggressive ones to maximize blank video removal.
- **Targeted species search**: Zamba can be used to identify videos containing specific species of interest by sorting videos in descending order of probability for the target

species. Targeted search saves researchers significant time and allows them to get more value out of camera trap data. Even with an imperfect model, researchers can focus on high confidence predictions overall or high confidence predictions within specific labels. Targeted search also enables collaboration between research groups, as one team's bycatch can become another team's primary data source.

- **Custom models**: Custom models trained on user-provided data support use in new habitats or with different species. They can also be used to train site-specific models, which can yield more accurate predictions even if the species labels overlap with existing models, or improve classification of rare species. We recommend users first evaluate existing pretrained models before investing in custom training. When developing custom models, researchers should understand that the amount of data needed depends greatly on the specific data and use case, so we recommend an iterative approach. Label distribution in the training data will also affect model behavior: rare classes are harder for models to learn effectively, while balanced datasets may produce more false positives for genuinely rare species in deployment.

9. Conclusion

Zamba is a powerful tool in supporting wildlife conservation. Initially developed to tackle the technically demanding task of processing camera-trap videos, it has since expanded to handle images as well. A key capability of Zamba is the ability to fine-tune models to better target specific habitats or expand to new subjects of interest, with support for species classification for both videos and images. The Zamba open source package supports programmatic use at the command line or as a Python library, while Zamba Cloud provides a no-code option for non-programmer users. The use of automated machine learning workflows for camera trap data can save countless hours of valuable time, speed up conservation interventions, and enable camera traps to be used to their fullest potential.

Acknowledgments

This project has been supported by funding from the Max Planck Institute for Evolutionary Anthropology, the Arcus Foundation, the Patrick J. McGovern Foundation, and the WILDLABS Awards 2024.

The authors express their gratitude to collaborators, advisors, and data providers at the following organizations: the Max Planck Institute for Evolutionary Anthropology, the German Centre for Integrative Biodiversity Research (iDiv) Halle-Jena-Leipzig, the Helversen'sche Stiftung, and Dan Morris.

The authors thank the following PanAf collaborators for supporting and collecting the PanAf video dataset: Abel Nzeheke, Alexander Tickle, Amelia Meier, Anne-Celine Granjon, Anthony Agbor, Dervla Dowd, Emmanuel Ayuk Ayimisin, Emmanuel Dilambaka, Emmanuelle Normand, Fiona Stewart, Geoffrey Muhanguzi, Giovanna Maretti, Henk Eshuis, Hilde Vanleeuwe, Jean Claude Dengui, John Hart, Joshua M. Linder, Jospehine Head, Juan Lapuente, Karsten Dierks, Katherine Corogenes, Kathryn J. Jeffery, Kevin Lee, Lucy Jayne Ormsby, Manasseh Eno-Nku, Martijn Ter Heegde, Mohamed Kambi, Nadege Wangue Njomen, Parag Kadam, Paul Telfer, Robinson Orume, Samuel Angedakin, Sergio Marrocoli, Sorrel Jones, Theophile Desarmeaux, Thurston Cleveland Hicks, Vera Leinert, Vianet Mihindou, Vincent Lapeyre, and Virginie Vergnes. The authors also thank the organizations and their members that have collaborated with the PanAf to collect this data: Budongo Conservation Field Station (Uganda), Fongoli Savanna Chimpanzee Project (Senegal), Gashaka Primate Project (Nigeria), Goualougo Triangle Ape Project, Greater Mahale Ecosystem Research and Conservation, Jane Goodall Institute Spain (Dindefelo, Senegal), Korup Rainforest Conservation Society (Cameroon), Lukuru Wildlife Research Foundation

(DRC), Ngogo Chimpanzee Project (Uganda), Ozouga Chimpanzee Project and Loango Gorilla Project (Gabon), Station d'Etudes des Gorilles et Chimpanzees (Gabon), Tai Chimpanzee Project (Cote d'Ivoire), the Aspinall Foundation (Gabon), WCS (Conkouati-Douli NP, R-Congo), Wild Chimpanzee Foundation (Cote d'Ivoire, Guinea, Liberia), Wildlife Conservation Society (WCS Nigeria), WWF (Campo Ma'an NP, Cameroon), and WWF Congo Basin (DRC). The authors further thank the PanAf project and data managers: Paula Dieguez, Mizuki Murai, and Yasmin Moebius. The authors also thank the PanAf video data cleaners and Chimp&See community scientists and moderators who annotated the PanAf video data: Nuria Maldonado, Anja Landsmann, Laura K. Lynn, Zuzana Rockaiova, Heidi Pfund, Heike Wilken, Lucia Hacker, Libby Smith, Karen Harvey, Tonnie Cummings, Carol Elkins, Briana Harder, Kristeena Sigler, Jane Widness, Amelie Pettrich, Antonio Buzharevski, Eva Martinez Garcia, Ivana Kirchmair, Sebastian Schütte, Joana Pereira, Silke Atmaca, Sadie Tenpas, and all community scientists. The authors extend their deepest gratitude to the government organizations that have permitted data collection in their countries: Ministere de la Recherche Scientifique et de l'Innovation, Cameroon; Ministere des Forets et de la Faune, Cameroon; Ministere des Eaux et Forets, Cote d'Ivoire; Ministère de l'Enseignement Supérieur et de la Recherche Scientifique, Côte d'Ivoire; Agence Nationale des Parcs Nationaux, Gabon; Centre National de la Recherche Scientifique, Gabon; Ministere de l'Agriculture de l'Elevage et des Eaux et Forets, Guinea; Forestry Development Authority, Liberia; National Park Service, Nigeria; Ministere de l'Economie Forestiere, R-Congo; Ministere de la Recherche Scientifique et Technologique, R-Congo; Direction des Eaux, Forêts et Chasses, Senegal; Tanzania Commission for Science and Technology, Tanzania; Tanzania Wildlife Research Institute, Tanzania; Makerere University Biological Field Station Uganda; Uganda National Council for Science and Technology, Uganda; Uganda Wildlife Authority, Uganda; National Forestry Authority, Uganda. The authors also thank the generous funders of PanAf: Max Planck Society, Max Planck Society Innovation Fund, Heinz L. Krekeler Foundation, Arcus Foundation, the Patrick J. McGovern Foundation, Google AI for Social Good, and Facebook.

The authors thank the other contributors to the Zamba and Zamba Cloud codebases: Justin Chung Clark, Casey Fitzpatrick, Robert Gibboni, Tamara Glazer, Isaac Slavitt, Stan Triepels, and Katie Wetstone. The authors also thank Dmytro Poplavskiy, the winner of the Pri-Matrix Challenge, whose winning solution was the basis for the original Zamba V1 video species classification model; and Kirill Brodt, the second-place winner of the Deep Chimpact Challenge, whose solution is the basis for the depth estimation model. The authors thank Hannah Moshontz de la Rocha for review and edits to the manuscript. Finally, the authors thank the reviewers who provided constructive comments and suggestions to improve the quality of the manuscript.

References

[1] A. C. Burton *et al.*, "REVIEW: Wildlife camera trapping: a review and recommendations for linking surveys to ecological processes," *Journal of Applied Ecology*, vol. 52, no. 3, pp. 675–685, 2015, doi: 10.1111/1365-2664.12432.

[2] M. Arandjelovic *et al.*, "Highly precise community science annotations of video cameratrapped fauna in challenging environments," *Remote Sensing in Ecology and Conservation*, vol. 10, no. 6, pp. 702–724, 2024, doi: 10.1002/rse2.402.

[3] J. Vélez *et al.*, "An evaluation of platforms for processing cameratrap data using artificial intelligence," *Methods in Ecology and Evolution*, vol. 14, no. 2, pp. 459–477, 2022, doi: 10.1111/2041-210x.14044.

[4] DrivenData, P. Bull, E. Dorne, R. Gibboni, J. Qi, and K. Wetstone, "Zamba." [Online]. Available: https://zenodo.org/doi/10.5281/zenodo.15116959

[5] F. Rovero and R. Kays, "Camera trapping for conservation," in *Conservation Technology*, Oxford University PressOxford, 2021, pp. 79–104. doi: 10.1093/oso/9780198850243.003.0005.

[6] M. W. Tobler, A. Zúñiga Hartley, S. E. CarrilloPercastegui, and G. V. N. Powell, "Spatiotemporal hierarchical modelling of species richness and occupancy using camera trap data," *Journal of Applied Ecology*, vol. 52, no. 2, pp. 413–421, 2015, doi: 10.1111/1365-2664.12399.

[7] R. Sollmann, A. Mohamed, H. Samejima, and A. Wilting, "Risky business or simple solution – Relative abundance indices from camera-trapping," *Biological Conservation*, vol. 159, pp. 405–412, 2013, doi: 10.1016/j.biocon.2012.12.025.

[8] E. J. Howe, S. T. Buckland, M. DesprésEinspenner, and H. S. Kühl, "Distance sampling with camera traps," *Methods in Ecology and Evolution*, vol. 8, no. 11, pp. 1558–1565, 2017, doi: 10.1111/2041-210x.12790.

[9] T. G. O'Brien *et al.*, "Camera trapping reveals trends in forest duiker populations in African National Parks," *Remote Sensing in Ecology and Conservation*, vol. 6, no. 2, pp. 168–180, 2019, doi: 10.1002/rse2.132.

[10] A. Caravaggi *et al.*, "A review of camera trapping for conservation behaviour research," *Remote Sensing in Ecology and Conservation*, vol. 3, no. 3, pp. 109–122, 2017, doi: 10.1002/rse2.48.

[11] M. Bessone *et al.*, "Drawn out of the shadows: Surveying secretive forest species with camera trap distance sampling," *Journal of Applied Ecology*, vol. 57, no. 5, pp. 963–974, 2020, doi: 10.1111/1365-2664.13602.

[12] S. E. Green, J. P. Rees, P. A. Stephens, R. A. Hill, and A. J. Giordano, "Innovations in Camera Trapping Technology and Approaches: The Integration of Citizen Science and Artificial Intelligence," *Animals*, vol. 10, no. 1, p. 132, 2020, doi: 10.3390/ani10010132.

[13] S. Beery, D. Morris, and S. Yang, "Efficient Pipeline for Camera Trap Image Review." [Online]. Available: https://arxiv.org/abs/1907.06772

[14] T. Haucke, H. S. Kühl, J. Hoyer, and V. Steinhage, "Overcoming the distance estimation bottleneck in estimating animal abundance with camera traps," *Ecological Informatics*, vol. 68, p. 101536, 2022, doi: 10.1016/j.ecoinf.2021.101536.

[15] J. A. Ahumada *et al.*, "Wildlife Insights: A Platform to Maximize the Potential of Camera Trap and Other Passive Sensor Wildlife Data for the Planet," *Environmental Conservation*, vol. 47, no. 1, pp. 1–6, 2019, doi: 10.1017/s0376892919000298.

[16] P. van Lunteren, "AddaxAI: A no-code platform to train and deploy custom YOLOv5 object detection models," *Journal of Open Source Software*, vol. 8, no. 88, p. 5581, 2023, doi: 10.21105/joss.05581.

[17] P. Bull, I. Slavitt, and G. Lipstein, "Harnessing the Power of the Crowd to Increase Capacity for Data Science in the Social Sector." [Online]. Available: https://arxiv.org/abs/1606.07781

[18] DrivenData, "Pri-matrix Factorization." [Online]. Available: https://www.drivendata.org/competitions/49/deep-learning-camera-trap-animals/

[19] DrivenData, "Results from Pri-Matrix Factorization and a New Open Source Tool for Wildlife Research and Conservation." [Online]. Available: https://drivendata.co/blog/camera-trap-wildlife-winners

[20] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition." [Online]. Available: https://arxiv.org/abs/1512.03385

[21] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the Inception Architecture for Computer Vision." [Online]. Available: https://arxiv.org/abs/1512.00567

[22] F. Chollet, "Xception: Deep Learning with Depthwise Separable Convolutions." [Online]. Available: https://arxiv.org/abs/1610.02357

[23] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, "Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning." [Online]. Available: https://arxiv.org/abs/1602.07261

[24] T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, in KDD '16. 2016, pp. 785–794. doi: 10.1145/2939672.2939785.

[25] DrivenData, "Deep Chimpact: Depth Estimation for Wildlife Conservation." [Online]. Available: https://www.drivendata.org/competitions/82/competition-wildlife-video-depth-estimation/

[26] A. K. Kalan *et al.*, "Novelty Response of Wild African Apes to Camera Traps," *Current Biology*, vol. 29, no. 7, pp. 1211–1217, 2019, doi: 10.1016/j.cub.2019.02.024.

[27] M. S. McCarthy *et al.*, "Camera traps provide a robust alternative to direct observations for constructing social networks of wild chimpanzees," *Animal Behaviour*, vol. 157, pp. 227–238, 2019, doi: 10.1016/j.anbehav.2019.08.008.

[28] C. Boesch *et al.*, "Chimpanzee ethnography reveals unexpected cultural diversity," *Nature Human Behaviour*, vol. 4, no. 9, pp. 910–916, 2020, doi: 10.1038/s41562-020-0890-1.

[29] B. Debetencourt, M. M. Barry, M. Arandjelovic, C. Stephens, N. Maldonado, and C. Boesch, "Camera traps unveil demography, social structure, and home range of six unhabituated Western chimpanzee groups in the Moyen Bafing National Park, Guinea," *American Journal of Primatology*, vol. 86, no. 2, 2023, doi: 10.1002/ajp.23578.

[30] M. S. McCarthy *et al.*, "Chimpanzee identification and social network construction through an online citizen science platform," *Ecology and Evolution*, vol. 11, no. 4, pp. 1598–1608, 2020, doi: 10.1002/ece3.7128.

[31] J. S. Head, C. Boesch, M. M. Robbins, L. I. Rabanal, L. Makaga, and H. S. Kühl, "Effective sociodemographic population assessment of elusive species in ecology and conservation management," *Ecology and Evolution*, vol. 3, no. 9, pp. 2903–2916, 2013, doi: 10.1002/ece3.670.

[32] M. Henrich *et al.*, "A semiautomated camera trap distance sampling approach for population density estimation," *Remote Sensing in Ecology and Conservation*, vol. 10, no. 2, pp. 156–171, 2023, doi: 10.1002/rse2.362.

[33] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778. doi: 10.1109/cvpr.2016.90.

[34] D. Tran, H. Wang, L. Torresani, J. Ray, Y. LeCun, and M. Paluri, "A Closer Look at Spatiotemporal Convolutions for Action Recognition," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018. doi: 10.1109/cvpr.2018.00675.

[35] G. Bertasius, H. Wang, and L. Torresani, "Is Space-Time Attention All You Need for Video Understanding?." [Online]. Available: https://arxiv.org/abs/2102.05095

[36] C. Feichtenhofer, "X3D: Expanding Architectures for Efficient Video Recognition," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 200–210. doi: 10.1109/cvpr42600.2020.00028.

[37] J. Carreira and A. Zisserman, "Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. doi: 10.1109/cvpr.2017.502.

[38] M. Tan and Q. V. Le, "EfficientNetV2: Smaller Models and Faster Training," 2021, doi: 10.48550/ARXIV.2104.00298.

[39] fastai developers, "TimeDistributed." [Online]. Available: https://docs.fast.ai/layers.html#timedistributed

[40] C. Feichtenhofer, H. Fan, J. Malik, and K. He, "SlowFast Networks for Video Recognition." [Online]. Available: https://arxiv.org/abs/1812.03982

[41] Z. Ge, S. Liu, F. Wang, Z. Li, and J. Sun, "YOLOX: Exceeding YOLO Series in 2021." [Online]. Available: https://arxiv.org/abs/2107.08430

[42] G. Hinton, O. Vinyals, and J. Dean, "Distilling the Knowledge in a Neural Network." [Online]. Available: https://arxiv.org/abs/1503.02531

[43] DrivenData, "What's a takahē?." [Online]. Available: https://drivendata.co/blog/zamba-find-takahe

[44] T. Gadot *et al.*, "To crop or not to crop: Comparing wholeimage and cropped classification on a large dataset of camera trap images," *IET Computer Vision*, vol. 18, no. 8, pp. 1193–1208, 2024, doi: 10.1049/cvi2.12318.

[45] D. Morris, "Labeled Information Library of Alexandria: Biology and Conservation." [Online]. Available: https://lila.science/

[46] S. Beery, G. Van Horn, and P. Perona, "Recognition in Terra Incognita," in *Computer Vision – ECCV 2018*, Springer International Publishing, 2018, pp. 472–489. doi: 10.1007/978-3-030-01270-0_28.

[47] The Nature Conservancy, "Channel Islands Camera Traps." [Online]. Available: https://lila.science/datasets/channel-islands-camera-traps/

[48] Desert Lion Conservation Project, "Desert Lion Conservation Camera Traps." [Online]. Available: https://lila.science/datasets/desert-lion-conservation-camera-traps/

[49] H. Yousif, R. Kays, and Z. He, "Dynamic Programming Selection of Object Proposals for Sequence-Level Animal Species Classification in the Wild," *IEEE Transactions on Circuits and Systems for Video Technology*, 2019.

[50] Idaho Department of Fish and Game, "Idaho Camera Traps." [Online]. Available: https://lila.science/datasets/idaho-camera-traps/

[51] Island Conservation, "Island Conservation Camera Traps." [Online]. Available: https://lila.science/datasets/island-conservation-camera-traps/

[52] Z. Zhang, Z. He, G. Cao, and W. Cao, "Animal Detection From Highly Cluttered Natural Scenes Using Spatiotemporal Object Region Proposals and Patch Verification," *IEEE Transactions on Multimedia*, vol. 18, no. 10, pp. 2079–2092, 2016, doi: 10.1109/tmm.2016.2594138.

[53] M. A. Tabak *et al.*, "Machine learning to classify animal species in camera trap images: Applications in ecology," *Methods in Ecology and Evolution*, vol. 10, no. 4, pp. 585–590, 2018, doi: 10.1111/2041-210x.13120.

[54] New Zealand Trailcams, "Trail Camera Images of New Zealand Animals." [Online]. Available: https://lila.science/datasets/nz-trailcams

[55] L. E. Pardo *et al.*, "Snapshot Safari: A large-scale collaborative to monitor Africa's remarkable biodiversity," *South African Journal of Science*, vol. 117, no. 1/2, 2021, doi: 10.17159/sajs.2021/8134.

[56] Saola Working Group, "Northern and Central Annamites Camera Traps 2.0." [Online]. Available: https://lila.science/datasets/swg-camera-traps

[57] Wildlife Conservation Society, "WCS Camera Traps." [Online]. Available: https://lila.science/datasets/wcscameratraps

[58] V. Anton, S. Hartley, A. Geldenhuis, and H. U. Wittmer, "Monitoring the mammalian fauna of urban areas using remote cameras and citizen science," *Journal of Urban Ecology*, vol. 4, no. 1, 2018, doi: 10.1093/jue/juy002.

[59] N. A. Gilbert, J. D. J. Clare, J. L. Stenglein, and B. Zuckerberg, "Abundance estimation of unmarked animals based on cameratrap data," *Conservation Biology*, vol. 35, no. 1, pp. 88–100, 2020, doi: 10.1111/cobi.13517.

[60] S. T. Buckland, D. R. Anderson, K. P. Burnham, J. L. Laake, D. L. Borchers, and L. Thomas, *Introduction to Distance Sampling: Estimating Abundance of Biological Populations*. Oxford University PressOxford, 2001. doi: 10.1093/oso/9780198506492.001.0001.

[61] ffmpeg.wasm developers, "ffmpeg.wasm." [Online]. Available: https://github.com/ffmpegwasm/ffmpeg.wasm

[62] FFmpeg developers, "FFmpeg." [Online]. Available: https://ffmpeg.org/

[63] Django developers, "Django." [Online]. Available: https://www.djangoproject.com/

Open Access | https://doi.org/10.25080/tyyd7356

SciPy 2025

July 7 - July 13, 2025

Proceedings of the 24th Python in Science Conference
ISSN: 2575-9752

A Practical Guide to Data Quality: Problems, Detection, and Strategy

Edson Bomfim[1,2]

[1]Final04, [2]Universidade de São Paulo, USP

Abstract

The proliferation of data-driven systems, from machine learning models to business intelligence platforms, has placed unprecedented importance on the quality of the underlying data. The principle of "garbage in, garbage out" has never been more relevant, as poor data quality can lead to flawed models, incorrect business decisions, and a fundamental lack of trust in data systems.

This paper serves as a practical guide for software developers and data practitioners to understand, identify, and address data quality issues. We present a comprehensive taxonomy of 23 common data quality problems, synthesizing definitions and contexts from empirical software engineering and database literature. Subsequently, we catalogue 22 distinct methods for detecting these issues, categorizing them into actionable strategies: Rule-Based & Constraint Enforcement, Statistical & Machine Learning Analysis, Data Comparison & Standards, and Human & Manual Review.

By mapping specific problems to their detection-method categories, we provide a structured framework to help teams develop robust, efficient, and targeted data validation strategies.

We conclude by discussing how the systematic practice of "data testing," analogous to software testing, not only improves pipeline maintainability and dataset trustworthiness but also enhances the overall design and reliability of data-intensive systems.

Keywords data, data quality, software quality

1. Introduction

In the modern technological landscape, data is the foundational asset that powers innovation, from scientific research, training sophisticated machine learning (ML) models to driving critical business analytics. A dataset can be defined as a structured collection of individual data points, or instances, that hold information about a set of entities sharing some common characteristics. The utility and reliability of any system built upon this data are inextricably linked to its quality.

This paper treats the quality of data as the characteristic of the dataset to match the natural distribution, in other words the degree to which it accurately and faithfully represents the real-world phenomena it purports to describe [1], [2]. This is a crucial distinction. For example, a common challenge in machine learning is class imbalance, where one class is significantly underrepresented. However, if this imbalance accurately reflects the real world (e.g., fraudulent transactions are rare), the dataset itself is not of poor quality; rather, it is a perfect representation of a difficult problem. The challenge then lies with the modeling technique, not the data's fidelity. Our focus is on errors where the data *fails* to match the natural distribution.

Published Jul 10, 2025

Correspondence to
Edson Bomfim
research@final04.com

Open Access

Copyright © 2025 Bomfim. This is an open-access article distributed under the terms of the Creative Commons Attribution 4.0 International license, which enables reusers to distribute, remix, adapt, and build upon the material in any medium or format, so long as attribution is given to the creator.

The consequences of poor data quality are severe, leading to the well-known "garbage in, garbage out" paradigm. Flawed data can introduce subtle biases in algorithms, generate misleading analytical reports, and erode stakeholder trust, ultimately undermining the value of data-driven initiatives [3], [4]. Despite its importance, a systematic, developer-focused approach to identifying and mitigating these issues is often lacking.

This paper aims to fill that gap by providing a practical, empirical guide for data practitioners. We synthesize a comprehensive taxonomy of data quality problems and a corresponding catalogue of detection methods, drawing from extensive research in empirical software engineering, data cleaning, database management and the author own experience. The goal is to equip developers with a structured framework to reason about, test for, and strategically address data quality within their systems.

The paper is structured as follows:

- Data Quality Problems details a taxonomy of 23 common data quality problems, providing concise definitions and relevant context for each, and organizes them into practical categories.
- Methods to Detect Data Quality Problems presents a catalogue of 22 methods used to detect these problems, discusses their automation potential, and groups them into four strategic categories. This section includes a mapping of which method categories are effective for which errors.
- Case Study: A Practical Analysis of a Public Health Dataset provides a practical analysis of a real-world public health dataset, demonstrating how the catalogued problems and detection methods can be applied to identify concrete data quality issues.
- Conclusions discuss with the strategic implications of adopting a proactive data quality assurance mindset, drawing parallels between data testing and established software testing practices to argue for a more robust approach to building data systems.

2. Data Quality Problems

A data quality problem, or "dirty data," is any instance where the data fails to correctly represent the real-world entity or event it describes [5], [6], [7]. These problems can be subtle or overt, but all carry the risk of corrupting analysis and undermining model performance. Below is a taxonomy of common data quality problems compiled from the literature.

2.1. Taxonomy of Data Quality Problems

1. **Missing Data / Incompleteness / Empty Values:** Records or fields that should contain a value but are null, empty, or otherwise not populated. This is one of the most common issues, forcing decisions about imputation or removal [8], [9].
2. **Duplicates / Redundancy:** The same real-world entity is represented by two or more records in a dataset. These can be exact duplicates or approximate (fuzzy) duplicates with minor variations [7], [10], [11].
3. **Inconsistency / Contradicting Records / Semantic Consistency Violations:** A single real-world entity is described by multiple records that contain conflicting information (e.g., the same customer with two different birth dates). This also includes violations of semantic rules (e.g., a patient's discharge date is before their admission date) [7], [12].
4. **Wrong / Incorrect Data / Invalid Data:** Data values that are syntactically valid but factually wrong (e.g., an age of "30" for a person who is actually 40). This is one of the hardest problems to detect without an external source of truth [6], [10].

5. **Misspellings:** Typographical errors in textual data, which can hinder matching and categorization (e.g., “Jhon” instead of “John”) [6], [10].
6. **Outdated Temporal Data / Timeliness / Currency / Volatility:** The data was correct at the time of recording but is no longer valid due to real-world changes (e.g., a customer’s address before they moved). This dimension measures how up-to-date the data is [6], [13].
7. **Non-standardized / Non-Conforming Data / Irregularities:** The use of different formats or units to represent the same information (e.g., “USA”, “U.S.A.”, “United States”; or measurements in both Celsius and Fahrenheit) [6], [6].
8. **Ambiguous Data / Incomplete Context:** Data that can be interpreted in multiple ways without additional context. This can arise from abbreviations (“J. Stevens” for John or Jack) or homonyms (“Miami” in Florida vs. Ohio) [6], [10].
9. **Embedded / Extraneous Data / Value Items Beyond Attribute Context:** A single field contains multiple pieces of information that should be separate (e.g., a “name” field containing “President John Stevens”) or information that belongs in another field (e.g., a zip code in an address line) [6], [10].
10. **Misfielded Values:** Correct data is stored in the wrong field (e.g., the value “Portugal” in a “city” attribute) [10].
11. **Outliers / Noise:** Erroneous data points that lie significantly outside the typical distribution of values, often due to measurement or entry errors. They are distinct from valid but extreme data points [8].
12. **Insufficient / Imprecise Metadata:** The metadata (data about the data) is missing or does not adequately describe the metrics, entities, or collection process, leading to a high risk of misinterpretation [13], [14].
13. **Schema Violations / Wrong Data Type / Structural Conflicts:** Data that violates the formal schema of the database, such as incorrect data types (e.g., “XY” in an age field), or different structural representations for the same object across systems [7], [10].
14. **Dangling Data / Referential Integrity Violation:** A record refers to another record that does not exist, breaking a relational link (e.g., an order record with a `customer_id` that does not exist in the customers table) [6], [10].
15. **Concurrency Control / Transaction Issues:** Data corruption arising from the lack of proper transaction management in databases, leading to issues like lost updates or dirty reads. While typically handled by the DBMS, it is a source of poor data quality [6].
16. **Wrong Categorical Data:** A value for a categorical attribute is outside the predefined set of valid categories (e.g., `shipping_method` is “air” when the only valid options are “ground” or “sea”) [6].
17. **Name Conflicts / Homonyms / Synonyms:** When integrating data, the same name is used for different objects (homonyms) or different names are used for the same object (synonyms), causing structural ambiguity [7].
18. **Different Aggregation Levels:** In data integration, the same entity is represented at different levels of detail across sources (e.g., sales data aggregated by day in one source and by month in another) [15].
19. **Circularity Among Tuples in Self-Relationship:** A logical impossibility where entities in a self-referencing relationship form a cycle (e.g., Product A is a sub-product of B, and Product B is a sub-product of A) [15].
20. **Semi-empty Tuple:** A record where a large number of its attributes are empty, but not all. This may indicate a partial data entry failure or a distinct sub-type of entity that was not modeled correctly [15].
21. **Data Miscoding:** The incorrect assignment of data types during processing, such as treating a numerical feature as a string. This is a process-induced error rather than a source data error [16].

22. **Distribution Shift / Data Drift:** The statistical properties of the data change over time or between different contexts (e.g., training vs. production), rendering models trained on older data less accurate [16].
23. **Plausibility:** Data values that are syntactically correct and within a valid range, but are highly improbable in the given context [17].

2.1.1. *A Note on Excluded Concepts*

- **Trust/Trustworthiness:** While critically important, trust is a property of the relationship between a data consumer and a data source, not an intrinsic property of the dataset itself. It is often established over time through repeated positive interactions and is influenced by the provenance and perceived reliability of the source [8]. Therefore, it is considered outside the scope of direct data quality measurement.
- **Accessibility/Availability/Security:** These dimensions relate to the systems and procedures governing access to the data, not the content of the data itself [13]. While a dataset that cannot be accessed is useless, we focus here on the quality of the data once it has been accessed.
- **Representational Validity/Sampling Bias:** A dataset can be internally accurate yet fail to represent its target phenomenon. This issue, known as sampling bias, stems from fundamental flaws in the data collection strategy—for instance, surveying only a specific subset of a population. Because these are problems of study design concerning data that was never collected, they fall outside this paper's scope, which is confined to the intrinsic quality and fidelity of the data at hand.

2.2. Detection Scalability

Data quality problems can be broadly divided into two groups based on the scope required for their detection:

- **Single-Datapoint Errors:** These errors can be identified by examining a single record or data point in isolation. Examples include `Missing Data`, `Schema Violations`, and `Misspellings`. Detection for these errors is highly parallelizable and scales linearly with the size of the dataset, as each data point can be checked independently.
- **Multi-Datapoint Errors:** These errors require the context of other records, or even the entire dataset, to be identified. `Duplicates`, `Outliers`, and `Distribution Shift` fall into this category. Detection can be computationally expensive, often requiring full-dataset statistics. While optimizations like sorting or indexing can help, adding new data may necessitate a full re-evaluation of the entire dataset.

2.3. Categorizing Data Quality Problems

While a flat list of problems is useful, categorization helps in forming a mental model for tackling them. A common approach in the literature is to group them by the aspect of quality they violate [13]:

- **Content Quality (Accuracy & Integrity):** Problems related to the correctness and internal consistency of the data. Includes *Inconsistency*, *Wrong Data*, *Misspellings*, *Outliers*, *Schema Violations*, and *Dangling Data*.
- **Completeness (Sufficiency of Information):** Problems concerning missing information. Includes *Missing Data*, *Semi-empty Tuple*, and *Insufficient Metadata*.
- **Representational & Structural Quality (Form & Understandability):** Problems with how the data is structured and represented. Includes *Non-standardized Data*, *Embedded Data*, *Misfielded Values*, *Name Conflicts*, and *Wrong Categorical Data*.

- **Timeliness & Relevance (Fitness for Current Context):** Problems related to the data being out-of-date or contextually inappropriate. Includes *Outdated Temporal Data* and *Distribution Shift*.

While this categorization is useful for understanding the *impact* of poor data quality, a more practical approach is to categorize problems based on the *methods required to detect them*. This pragmatic view directly informs the implementation of a data quality assurance strategy, which we explore in the next section.

3. Methods to Detect Data Quality Problems

Detecting the problems outlined in Section 2 requires a diverse toolkit of methods, ranging from simple, deterministic rules to complex statistical models. Below, we catalogue these methods and discuss some aspects of their application.

Table 1. *Mapping Detection Methods to Data Quality Problems*

Method	Applicable Data Quality Problems Found
Database Constraints (Not Null, Unique, Primary Key, Foreign Key)	Missing Data, Duplicates, Dangling Data, Schema Violations
User-Specified Integrity Constraints & Triggers	Inconsistency, Wrong Data, Wrong Categorical Data, Circularity
Direct Checks (for Null, Empty Strings, Dummy Values)	Missing Data, Semi-empty Tuple
Quantitative & Statistical Metrics	Missing Data (e.g., completion rate), Distribution Shift
Comparison & Similarity (Direct, Distance, String Metrics)	Duplicates, Inconsistency, Misspellings, Non-standardized Data
Sorted Neighbourhood Method	Duplicates (approximate)
Attribute Cardinality Checks	Duplicates (where cardinality should equal row count)
Redundancy Analysis	Redundancy (correlated features)
Semantic & Rule-Based Testing (incl. Regex, Set Validation)	Inconsistency, Wrong Data, Misfielded Values, Schema Violations, Non-standardized Data
Probability-based Metrics & Statistical Tests	Inconsistency (Semantic), Plausibility
Association Rule Mining	Inconsistency, Wrong Data
Comparison to Reference Data (Real World, Legal Values)	Wrong / Incorrect Data, Outdated Temporal Data
Explicit Type Checking & Domain Format Checks	Schema Violations, Non-standardized Data
Spell Checker	Misspellings
Temporal Constraint Checks	Outdated Temporal Data
Distribution Analysis (L-infinity, Jensen-Shannon, etc.)	Distribution Shift, Outliers
Outlier Detection Algorithms (ML, Statistical, Distance-based)	Outliers, Noise
Clustering Algorithms	Outliers, Duplicates (by grouping similar records)
Lookup Tables & Dictionaries (Thesaurus, Name/Address Dirs)	Non-standardized Data, Ambiguous Data, Misspellings

Metadata & Schema Evaluation (Metamodels, DQRs)	Insufficient / Imprecise Metadata, Schema Violations, Name Conflicts
Cycle Detection Algorithms	Circularity Among Tuples
Data Profiling & Human Review	All (serves as a general-purpose discovery and verification method)

3.1. Automation and Maintenance:

The methods above vary significantly in their potential for automation. Database constraints are fully automated, while methods like statistical analysis and rule-based testing can be incorporated into automated data pipelines. Others, such as resolving ambiguous data using a thesaurus or verifying incorrect data against the real world, fundamentally require human-in-the-loop intervention.

Just as software testing is an integral part of CI/CD pipelines, data quality tests should be an integral part of data pipelines. Whenever data is updated, ingested, or transformed, a corresponding suite of data tests should be executed. Single-datapoint checks can be run efficiently on new or changed data. However, for multi-datapoint checks, an incremental update may not be sufficient; a full-dataset re-evaluation is often necessary to ensure issues like new duplicates or shifts in the overall distribution are caught.

3.2. Categories of Detection Methods

To provide a clear, actionable framework, we first present a table mapping individual detection methods to the data quality problems they are primarily used to find. A more strategic approach is to group these methods into broader categories. We propose four categories of data quality detection methods, which group the 22 techniques into strategic approaches:

3.2.1. *Rule-Based & Constraint Enforcement*

This category involves defining explicit, deterministic rules that data must adhere to, often enforced directly at the database level through mechanisms like schema constraints. These methods are highly automatable and effective for catching violations of known business logic, structural integrity, and formatting requirements, such as ensuring a field is never null, conforms to a specific data type, or follows a predefined pattern.

3.2.2. *Statistical & Machine Learning Analysis*

This approach leverages mathematical and algorithmic techniques to identify data quality issues by analyzing the aggregate properties and distributions within a dataset. By establishing statistical norms, these methods can automatically flag outliers and uncover improbable or inconsistent data patterns that would not violate simple, hard-coded rules.

3.2.3. *Data Comparison & Standards*

This category focuses on validating data by comparing it against a trusted source of truth or by identifying inconsistencies through intra-dataset comparisons. It includes techniques like matching records against external reference data (e.g., a list of valid postal codes), using string similarity metrics to find fuzzy duplicates, and employing lookup tables or dictionaries to standardize terminology.

3.2.4. *Human & Manual Review*

This category encompasses all methods that rely on human intelligence, domain expertise, and contextual understanding to identify and verify data quality problems. It serves as the final line of defense for detecting complex, subtle, or novel issues that automated systems miss, such as evaluating the sufficiency of metadata, verifying the real-world accuracy of a questionable value, or interpreting ambiguous data through expert judgment.

Table 2. *Categories of Data Quality Detection Methods*

Category	Methods Included
Rule-Based & Constraint Enforcement	Not Null Constraints, Unique/Primary Key Constraints, General DB Integrity Constraints, Triggers, Rule-based Testing, Regex Pattern Verification, Set Validation, Type Checking/ Constraints, Temporal Constraints, Domain Format Checks, Cycle Detection Algorithms, Formal Specifications (DQRs, DPRs).
Statistical & Machine Learning Analysis	Quantitative Metrics, Statistical Criterion/Sampling, Redundancy Analysis, Probability-based Metrics, Association Rule Mining, Distribution Analysis (L-infinity, Jensen-Shannon, etc.), All Outlier Detection Methods (ML, Statistical, Distance-based), Clustering, Trimmed Means.
Data Comparison & Standards	Direct Comparison, Distance Metrics, String Similarity Metrics, Sorted Neighbourhood, Comparison to Reference Data, Standardization/ Normalization, Lookup Tables/Dictionaries, Thesaurus, Name/Address Directories.
Human & Manual Review	Identifying Dummy Values, Data Profiling, Metadata Evaluation, Human Verification/Manual Inspection, Domain Expert Intervention.

This categorization allows for a structured approach to building a data quality firewall. A team can start with the most automatable category, “Rule-Based & Constraint Enforcement,” and progressively add more sophisticated “Statistical & ML Analysis” and “Data Comparison” methods, while reserving “Human & Manual Review” for exceptions and complex cases.

Also we present the Table 3 which provides a high-level map for practitioners, showing which categories of methods are effective at identifying specific data quality problems.

Table 3. *Linking Data Problems to Detection Method Categories*

Data Quality Problem	Rule-Based & Constraint	Statistical & ML	Data Comparison & Standards	Human & Manual
Missing Data / Incompleteness	✓	✓		✓
Duplicates / Redundancy	✓	✓	✓	✓
Inconsistency / Contradictions	✓	✓		✓
Wrong / Incorrect Data	✓	✓	✓	✓

Misspellings				✓	✓
Outdated Temporal Data	✓			✓	✓
Non-standardized Data	✓			✓	✓
Ambiguous Data				✓	✓
Embedded / Extraneous Data	✓				✓
Misfielded Values	✓				✓
Outliers / Noise			✓		✓
Insufficient / Imprecise Metadata					✓
Schema Violations	✓				✓
Dangling Data / Referential Integrity	✓				
Concurrency / Transaction Issues	✓				
Wrong Categorical Data	✓		✓		✓
Name Conflicts					✓
Different Aggregation Levels				✓	✓
Circularity Among Tuples	✓				
Semi-empty Tuple	✓				✓
Data Miscoding	✓				✓
Distribution Shift / Data Drift			✓		✓
Plausibility			✓		✓

4. Case Study: A Practical Analysis of a Public Health Dataset

To bridge the gap between the theoretical taxonomy of problems and the catalogue of detection methods, this section presents a practical analysis of a real-world dataset: Brazil's Mortality Information System (SIM - Sistema de Informação sobre Mortalidade) [18]. As a large, publicly available dataset aggregated from numerous sources across the country, it serves as an excellent example of the types of data quality challenges practitioners face. We will examine data from 2024 to illustrate how specific problems can be identified using the methods discussed. This dataset is publicly available from Brazil's Open Data SUS portal.

Below is a sample from the SIM dataset, showing a selection of key fields.

Table 4. *Sample Data from the Brazilian Mortality Information System (SIM) 2024*

DTOBITO (Date of Death)	HORAOBITO (Time of Death)	DTNASC (Date of Birth)	IDADE (Age)	SEXO (Sex)	RACACOR (Race/ Color)	ESTCIV (Marital Status)	SERIESCFAL (Last Grade)	CAUSABAS (Basic Cause)	ATESTADO (Death Cert. Causes)
19042024	2301	24121964	459	1	1	1		I279	J81/ I500/ I279/ I10
18012024	1030	29041958	465	2	1	2		A09	I219/ A09
22012024	0700	23031939	484	2	4	2		E112	I219/ I10/ E112
12012024	0800	08031950	473	2	2	5		I219	I219/ I509/ I10
01032024	1030	12051938	485	2	1	9		N19	J969/ I469/ D649/ N19*I509

4.1. Example 1: Detecting Data Miscoding with Human & Manual Review

Problem: A preliminary inspection of the IDADE (Age) field reveals values such as 459 and 484. Without domain context, these appear to be erroneous. A simple cross-validation by subtracting DTNASC from DTOBITO for the first record reveals an age of 59 years, suggesting the value 459 is not a raw integer but an encoded value.

Detection: This is a classic case where Human & Manual Review, informed by domain knowledge, is the primary detection method. The initial observation of impossible age values prompts a deeper investigation. Consulting the official data dictionary confirms that the IDADE field uses a special encoding where the leading digit signifies the time unit (e.g., ‘4’ for years) and the subsequent digits represent the quantity. Failure to identify this results in Data Miscoding, a severe process-induced error. Once detected, the remediation is a straightforward Rule-Based transformation.

4.2. Example 2: Quantifying Missing Data with Rule-Based Enforcement

Problem: The SERIESCFAL (Last Grade) column is blank in every record of the sample, representing a clear case of Missing Data. The impact of this problem depends entirely on the intended use case.

Detection: A simple Rule-Based check can identify null or empty values on a per-record basis. However, the true utility comes from applying this rule across the entire dataset to establish a data quality metric. For an analysis where education level is a critical feature, a team might define an acceptance criterion, such as requiring at least 50% of records to be non-null. The following programmatic check implements this quality gate:

```
# Assumes 'df' is a pandas DataFrame of the full 2024 dataset
acceptance_threshold = 0.5
completeness_ratio = df.SERIESCFAL.notnull().sum() / len(df)
assert completeness_ratio >= acceptance_threshold
```

Executing this check against the full dataset would return False, indicating that the data fails this specific quality test and is not fit for this particular use.

4.3. Example 3: Identifying Inconsistency with Rule-Based Validation

Problem: A dataset can contain values that are individually valid but logically impossible in combination. According to the SIM documentation, SEXO=1 indicates a person born male, while GRAVIDEZ=1 signifies the person has been pregnant.

Detection: This Inconsistency can be detected using Rule-Based & Constraint Enforcement that evaluates a multi-field invariant. The following test efficiently isolates all records that violate this semantic rule:

```
# Filters for records where a person born male is recorded as having been pregnant
inconsistent_records = df[(df.GRAVIDEZ == 1) & (df.SEXO == 1)]
```

This test effectively detects that an error exists in the record but cannot determine which of the two fields is incorrect. The remediation strategy—whether to nullify the fields or discard the record—depends on further analysis or predefined business rules.

4.4. Example 4: Assessing Plausibility using Statistical Analysis

Problem: Certain data points, while not strictly impossible, may be so statistically improbable that they warrant scrutiny. The IDADEMAE (Mother's Age) field provides a clear example. Detection: By employing Statistical Analysis, we can profile the distribution of IDADEMAE to identify anomalous values. The table below shows the age distribution in bins for all non-null entries in the 2024 dataset.

Table 5. *Distribution of Mother's Age (IDADEMAE) in SIM 2024*

Age Bin	Count
(0, 10]	4
(10, 20]	4867
(20, 30]	11800
(30, 40]	7705
(40, 50]	1240
(50, 60]	10
(60, 70]	0
(70, 80]	0
(80, 90]	0
(90, 100]	26

This distribution immediately highlights potential Plausibility issues at the tails. While pregnancies in the 10-20 age group are common, the 4 instances below age 10 are highly suspect.

Even more striking are the 26 recorded pregnancies for mothers aged over 90. While not theoretically impossible with modern medicine, these are statistically extreme outliers and

highly likely to be data entry errors. Such records should be flagged as low-quality and considered for exclusion from sensitive analyses.

4.5. The Role of Data Quality Frameworks

Except for the Human & Manual check above, all others programmatic checks can be systemized using dedicated data quality frameworks. Great Expectations [19] is a open-source tool that institutionalizes "data testing" by allowing teams to declare assertions about their data in a readable, JSON-based format called "Expectations." This approach formalizes the data quality rules, making them version-controllable, shareable, and executable within automated pipelines, directly addressing the need for systematic data testing discussed in this paper.

The library provides a rich, built-in vocabulary of Expectations that directly map to the detection methods and problems we have catalogued. For instance, `expect_column_values_to_not_be_null` addresses Missing Data, `expect_column_values_to_be_in_set` handles Wrong Categorical Data, and `expect_column_mean_to_be_between` can detect Distribution Shift. More complex, multi-field invariants, such as the SEXO/GRAVIDEZ inconsistency, can be implemented using custom Expectations.

A key advantage of Great Expectations is its ability to automatically generate "Data Docs" human-readable documentation that presents the results of data validation runs. This feature transforms abstract quality metrics into tangible reports, fostering trust and communication between data producers and consumers. By integrating such a framework, the analysts can operationalize the principles outlined here, moving from ad-hoc data cleaning to a proactive, scalable, and automated data quality assurance strategy.

5. Conclusions

This paper has provided a comprehensive, practical-focused framework for understanding, identifying, and addressing data quality problems. By cataloguing 23 distinct types of "dirty data" and 22 corresponding detection methods, we have created a guide that can the turned into actionable strategies.

The key takeaway is the need to treat data quality assurance as a first-class citizen in the development of data systems. Understanding the fundamentals—the specific ways data can be dirty and the array of tools available to detect it—allows a team to optimize its validation strategy. Instead of applying a generic, one-size-fits-all approach, they can select the most efficient methods for their specific risks, whether that is enforcing a strict schema with **Rule-Based** methods, discovering novel anomalies with **Statistical** analysis, or standardizing inputs with **Data Comparison** techniques.

Integrating these methods into automated "data testing" suites within data pipelines is critical for long-term maintainability and trust. Just as unit tests verify code logic, data tests verify data integrity. This practice provides continuous feedback, catches regressions when data sources or processing logic change, and builds confidence in the final datasets used for researches, analytics, and machine learning.

Perhaps the most profound benefit of this approach lies in the design phase. The very process of defining data quality requirements and designing data tests forces a team to deeply consider potential failure modes. It requires them to ask critical questions: What are the invariants of our data? What are the semantic rules that must hold? What are the statistical norms? This process, much like writing the design of a prospective study or writing a formal specification for software, clarifies assumptions and exposes ambiguities *before* a

single line of data processing code is written. The resulting data system is not only more robust but also better designed, because it is built on a foundation of clearly articulated expectations about the data it will handle.

References

[1] Y. Wand and R. Y. Wang, "Anchoring data quality dimensions in ontological foundations," *Commun. ACM*, vol. 39, no. 11, pp. 86–95, 1996, doi: 10.1145/240455.240479.

[2] N. Hynes, D. Sculley, and M. Terry, "The Data Linter: Lightweight Automated Sanity Checking for ML Data Sets," 2017. [Online]. Available: http://learningsys.org/nips17/assets/papers/paper_19.pdf

[3] B. Saha and D. Srivastava, "Data quality: The other face of Big Data," in *2014 IEEE 30th International Conference on Data Engineering*, 2014, pp. 1294–1297. doi: 10.1109/ICDE.2014.6816764.

[4] K. Kerr, T. Norris, and R. Stockdale, "Data quality information and decision making: A healthcare case study," *ACIS 2007 Proceedings - 18th Australasian Conference on Information Systems*, 2007.

[5] W. Kim, B.-J. Choi, E. Hong, S.-K. Kim, and D. Lee, "A Taxonomy of Dirty Data," *Data Min. Knowl. Discov.*, vol. 7, pp. 81–99, 2003, doi: 10.1023/A:1021564703268.

[6] H. Müller and J.-C. Freytag, "Problems, methods, and challenges in comprehensive data cleansing," 2003.

[7] J. Barateiro and H. Galhardas, "A Survey of Data Quality Tools.," *Datenbank-Spektrum*, vol. 14, pp. 15–21, 2005.

[8] M. Bosu and S. MacDonell, "A Taxonomy of Data Quality Challenges in Empirical Software Engineering," in *Proceedings of the Australian Software Engineering Conference, ASWEC*, 2013, pp. 97–106. doi: 10.1109/ASWEC.2013.21.

[9] M. Shepperd, Q. Song, Z. Sun, and C. Mair, "Data Quality: Some Comments on the NASA Software Defect Datasets," *IEEE Transactions on Software Engineering*, vol. 39, no. 9, pp. 1208–1215, 2013, doi: 10.1109/TSE.2013.11.

[10] H. Galhardas, D. Florescu, D. Shasha, and E. Simon, "AJAX: an extensible data cleaning tool," in *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, in SIGMOD '00. Dallas, Texas, USA, 2000, p. 590. doi: 10.1145/342009.336568.

[11] A. E. Monge and C. P. Elkan, "The field matching problem: Algorithms and applications," in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, in KDD'96. Portland, Oregon, 1996, pp. 267–270.

[12] B. Heinrich, M. Klier, A. Schiller, and G. Wagner, "Assessing data quality – A probability-based metric for semantic consistency," *Decision Support Systems*, vol. 110, pp. 95–106, 2018, doi: https://doi.org/10.1016/j.dss.2018.03.011.

[13] F. Ridzuan and W. M. N. W. Zainon, "A Review on Data Quality Dimensions for Big Data," *Procedia Computer Science*, vol. 234, pp. 341–348, 2024, doi: https://doi.org/10.1016/j.procs.2024.03.008.

[14] Y. E. Ouatiti, M. Sayagh, N. Kerzazi, B. Adams, and A. E. Hassan, "The impact of concept drift and data leakage on log level prediction models," *Empirical Software Engineering*, vol. 29, no. 5, 2024, doi: 10.1007/s10664-024-10518-9.

[15] P. Oliveira, F. Rodrigues, and P. Rangel Henriques, "A Formal Definition of Data Quality Problems.," in *Proceedings of the 2005 International Conference on Information Quality, ICIQ 2005*, 2005.

[16] A. Bhatia, D. Lin, G. K. Rajbahadur, B. Adams, and A. E. Hassan, "Data Quality Antipatterns for Software Analytics." [Online]. Available: https://arxiv.org/abs/2408.12560

[17] W. W. Cohen, P. Ravikumar, and S. E. Fienberg, "A comparison of string distance metrics for name-matching tasks," in *Proceedings of the 2003 International Conference on Information Integration on the Web*, in IIWEB'03. Acapulco, Mexico, 2003, pp. 73–78.

[18] Brasil, Ministério da Saúde, "Banco de dados do Sistema´Unico de Saúde-DATASUS." [Online]. Available: https://opendatasus.saude.gov.br/pt_BR/dataset/sim

[19] A. Gong, J. Campbell, and Great Expectations, "Great Expectations." [Online]. Available: https://github.com/great-expectations/great_expectations

Open Access | https://doi.org/10.25080/yjpp7553

SciPy 2025

July 7 - July 13, 2025

Proceedings of the 24th Python in Science Conference
ISSN: 2575-9752

On the Path to Seamless Python Serverless Data Analytics

Enrique Molina-Giménez[1], **German T. Eizaguirre**[1], and **Pedro García-López**[1]

[1]Universitat Rovira i Virgili, Tarragona, Spain

Abstract

Traditionally, most data scientists rely heavily on clusters of virtual machines to handle large-scale workloads. While these clusters are effective, they entail substantial operational and maintenance costs, alongside increasing complexity in configuration and scaling — often demanding dedicated infrastructure specialists.

The rise of the serverless paradigm introduced a radically different model, centered on functions with automatic scaling and pay-per-use billing. To enable data analytics on serverless architectures, Lithops emerges as a powerful Python framework that abstracts away infrastructure complexities, empowering seamless parallel execution through serverless functions and allowing users to focus purely on their code.

A persistent challenge in serverless data analytics is the need to partition scientific data formats that were not natively designed for cloud computing. Proper partitioning is a crucial factor for maximizing parallelism and minimizing latency in data transfer from storage to compute resources. Dataplug addresses this by enabling on-the-fly partitioning for non-cloud-native data formats, optimizing data distribution among workers.

To further streamline the scientific development experience, we tackle the steep learning curve posed by configuring and managing serverless environments for users without DevOps expertise. We present PyRun: a browser-based IDE offering the familiar interface of notebook tools (*e.g.*, JupyterLab, Google Colab) but with native cloud integration. PyRun drastically reduces the cognitive burden of setup — delivering a ready-to-execute environment with zero configuration required.

Ultimately, the integration of Lithops, Dataplug, and PyRun forms a compelling stack that unlocks the full potential of serverless data analytics in Python, delivering clear and transformative benefits to scientific workflows.

Keywords serverless, data analytics, cloud computing, partitioning, transparency

Published Jul 10, 2025

Correspondence to
Enrique Molina-Giménez
enrique.molina@urv.cat

Open Access

Copyright © 2025 Molina-Giménez *et al.*. This is an open-access article distributed under the terms of the Creative Commons Attribution 4.0 International license, which enables reusers to distribute, remix, adapt, and build upon the material in any medium or format, so long as attribution is given to the creator.

1. From Cluster-Centric to Serverless Data Analytics

Data analytics has traditionally been conducted on clusters—networks of interconnected servers collaboratively processing and analyzing vast volumes of data. These clusters deliver the computational power, storage capacity, and parallelism necessary to address the growing scale and complexity of datasets across diverse domains such as finance, healthcare, and social media. By distributing workloads across multiple nodes, clusters enable efficient parallel execution, fault tolerance, and scalability, establishing themselves as the backbone of many big data frameworks like Apache Hadoop [1] and Apache Spark [2]. The cluster-centric approach has proven effective for batch processing and iterative algorithms that demand high throughput and sustained computational resources.

Nonetheless, inherent limitations persist within this architecture for data analytics workloads. Resource overprovisioning is commonplace, resulting in suboptimal and inefficient utilization. This inefficiency is particularly pronounced in scenarios with infrequent analytics executions, where users maintain clusters in a ready state without active workloads. Scalability also remains a critical challenge: server provisioning and deprovisioning often incur latencies of several minutes [3], underscoring limited resource elasticity. Furthermore, the operational complexity and maintenance overhead of cluster-based frameworks such as Spark and Hadoop necessitate specialized expertise to configure, optimize, and sustain the computing environment.

Conversely, simpler computing models have emerged, notably with the introduction of Function-as-a-Service (FaaS) by AWS in 2014. This paradigm abstracts infrastructure management entirely, allowing developers to focus solely on their application logic. Additional advantages—including a pay-as-you-go pricing model and low invocation latencies—have sparked significant interest within the scientific community. While FaaS was not originally designed with data analytics workloads in mind, its potential for scalable and parallel computation has motivated investigations into its viability as a core engine for distributed data analytics applications.

2. Lithops: The Pythonic Serverless Data Analytics Framework

Despite the previous suggested potential, leveraging FaaS platforms for large-scale data analytics presents several inherent challenges. These include the lack of direct peer-to-peer communication between functions, the absence of native synchronization mechanisms between stages (like in a MapReduce model), difficulties in managing dependencies and libraries across different cloud environments, and a general lack of portability due to proprietary APIs. The initial efforts on serverless data analytics field partially addressed these issues [4], but they often fell short of providing a comprehensive solution.

Lithops [5] entered to scene to tackle these challenges in a unified manner. Lithops is a serverless, multi-cloud framework designed for building both embarrassingly parallel and MapReduce-like applications using cloud functions. Lithops empowers users, even those without extensive cloud expertise, to effortlessly port their programs—from Monte Carlo simulations to analytics workflows—to the cloud. By transparently orchestrating hundreds of concurrent function instances, Lithops enables significant performance gains for "occasional" cloud users who might otherwise be daunted by the complexities of cluster resource provisioning.

From the programming perspective, Lithops addresses the aforementioned challenges by implementing a simple MapReduce interface that allows single-machine Python code to be executed as parallel jobs in the cloud. It also includes an intuitive storage API to facilitate communication between functions and with the Lithops client. Synchronization between map and reduce stages, including nested function compositions, is handled implicitly, reducing user burden, and Lithops automatically manages dependencies by leveraging Python's dynamic nature. Finally, its architecture is designed to abstract away cloud provider specifics, ensuring multi-cloud portability and mitigating vendor lock-in across platforms like IBM Cloud, Google Cloud and AWS.

In essence, Lithops provides a unified system to leverage cloud functions for massively-parallel execution of programs, enabling hundreds-way parallelism on short-lived functions and significantly accelerating everyday and scientific computing tasks.

2.1. Architecture

The core design of Lithops is centered around abstracting the complexities of cloud functions and providers to offer a seamless experience for parallel program execution. At its heart, Lithops operates as a multi-cloud framework, capable of deploying and managing functions while abstracting away the underlying cloud interface. This multi-cloud capability is a fundamental architectural choice, that mitigates vendor lock-in and offering users same interface across different choices.

A picture of the Lithops architecture is shown in Figure 1. The components can be initially splitted into two main categories: local components *(blue)* which run on the user's host, and remote components *(orange)* which run on the cloud provider in the form of serverless functions. On local side, the user uses the *Futures API* to submit jobs, which are then orchestrated by the *Executor*, which push/pull resources to/from the cloud provider, and manages the execution of the functions. Meanwhile, the *Invoker* is the component responsible for invoking the remote *Workers*, whose runtime is aware of the Lithops framework and can execute the user code remotely in a transparent way.

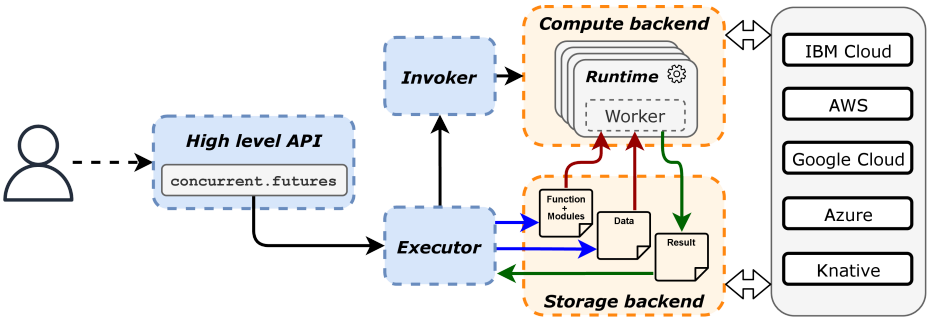


Figure 1. *Lithops Architecture Overview*

The typical operational flow of Lithops is as follows:

1. The user submits a job using the *Futures API*, which is handled by the *Executor*.
2. The *Executor* partitions the job into smaller tasks and push the necessary data and dependencies to the object storage.
3. The *Invoker* creates the remote workers, which pull the user code and dependencies from the object storage.
4. The remote *Workers* read the input data from the object storage, execute the user code, and write the results back to the object storage.
5. The *Executor* pulls the results from the object storage and returns them to the user.
6. The user can then retrieve the results from the *Futures API*.

Overall, the Lithops architecture follows a layered design, where the (client) framework layer builds on top of the serverless execution layer. This design effectively addresses the constraints of the serverless model by providing a unified and abstracted interface for large-scale parallel computing, enabling users to focus on their code and logic while the framework manages the orchestration and infrastructure transparently.

2.2. Programming Model

Lithops exposes a set of high-level APIs that simplify access to both compute and storage backends. The key interfaces are outlined below.

Futures API. Lithops offers a simple yet powerful programming model that enables users to transform single-machine Python code into massively parallel, cloud-based executions. The primary objective of this model is to substantially lower the entry barrier for adopting

cloud Function-as-a-Service (FaaS) platforms, especially for workloads that exhibit natural parallelism, similar to the MapReduce paradigm. To support this, the Lithops Futures API provides the `map()` function, which launches multiple workers to execute the same function across different data partitions. Building on this, the `map_reduce()` function extends this functionality by executing a final reduce function that gathers and processes the outputs from the map phase.

```
"""
Simple Lithops example using the map() call to estimate PI
"""
import lithops
import random

def is_inside(n):
    count = 0
    for i in range(n):
        x = random.random()
        y = random.random()
        if x*x + y*y < 1:
            count += 1
    return count

if __name__ == '__main__':
    np, n = 10, 15000000
    part_count = [int(n/np)] * np
    fexec = lithops.FunctionExecutor()
    fexec.map(is_inside, part_count)
    results = fexec.get_result()
    pi = sum(results)/n*4
    print("Estimated Pi: {}".format(pi))
```

The code snippet above illustrates a straightforward example of using the Lithops framework to estimate the value of π, using the Monte Carlo method. It first defines the quarter of a circle with radius 1, specifically that of the first quadrant. The algorithm then randomly generates points within the unit square of this quadrant. We can simply know if a point falls within the circle by checking if the coordinates satisfy the equation $x^2 + y^2 < 1$. As the area of the quarter circle is $\pi/4$, we can expect approximately that proportion of the points to fall within it. Therefore, we can estimate π by calculating the ratio of points that fall inside the circle to the total number of points.

Because each point calculation is completely independent, Lithops can run thousands of these trials in parallel across different workers, dramatically speeding up the estimation by simply summing their individual results. First, an instance of `lithops.FunctionExecutor()` is created. The `map()` method then launches multiple serverless functions that execute the `is_inside()` function concurrently, each processing a portion of the input data. Once all tasks complete, `get_result()` collects the results from the distributed execution and receives them locally, enabling the user to obtain the final estimation seamlessly.

Analogously to how π is estimated through the parallel execution of multiple functions, Lithops enables users to apply this pattern to a wide range of parallel computing problems, as simple as demonstrated in this example. While the core Lithops framework provides fundamental operators like MapReduce, it is part of a comprehensive software stack that significantly extends its capabilities. As we will explain throughout this work, this stack includes IO abstractions that serve as building blocks for group communication patterns, alongside a public workload registry offering programmatic examples of more complex operations such as joins and sorts.

Storage API. Lithops not only aims to simplify the experience of executing parallel code on serverless platforms, but also abstracts away data access operations—typically performed

over object storage. Whether interacting from the Lithops client or from remote workers, the Storage API provides a unified interface that, like the Futures API, avoids vendor lock-in.

In practice, this means that users can write code that works seamlessly not only across object storage from different cloud providers, but also with memory storage systems like Redis [6]. The following snippet shows a minimal working example—writing an object to storage and then reading it back—that operates seamlessly across all cloud providers supported by Lithops

```
from lithops import Storage

if __name__ == "__main__":
    st = Storage()
    st.put_object(bucket='mybucket', key='test.txt', body='Hello World')
    print(st.get_object(bucket='mybucket', key='test.txt'))
```

2.3. High-Performance Insights

Practical baseline performance. Several features make Lithops particularly well-suited for some high-performance parallel data analytics workloads, enabling users not only to execute large-scale computations with minimal effort but also leverage serverless computing upsides. When leveraging the `map()` primitive, users distribute their data processing tasks across numerous cloud functions, that presents key highlights that make it stand out, as presented below:

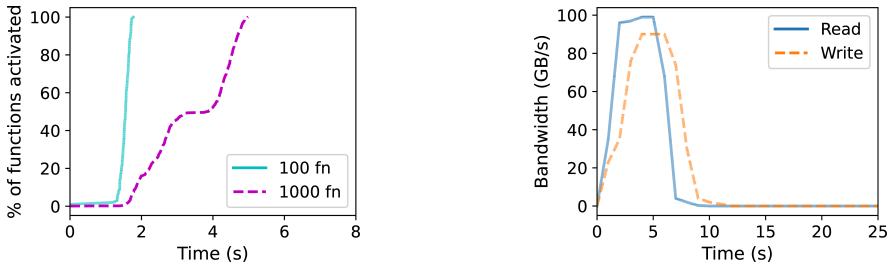


Figure 2. *Lithops performance: invocation latency (a) and aggregate bandwidth (b).*

- The elasticity of Function-as-a-Service (FaaS) platforms when launching functions is a key performance feature. As shown in Figure 2, we observe that 100 AWS Lambda functions are instantiated and start executing in less than 2 seconds; for 1000 functions, this delay increases only up to 5 seconds. The ability to scale from zero to thousands of vCPUs in such a short amount of time offers a level of burstiness previously unseen in traditional cluster environments.
- Once the compute resources (FaaS functions) are instantiated, they typically load their input data from object storage. As depicted in Figure 2b, the aggregate bandwidth offered by Amazon S3 reaches up to 100 GB/s for reads and 90 GB/s for writes when accessed concurrently by 1000 functions. This demonstrates the S3 high I/O parallelism, which Lambda functions can efficiently leverage for highly scalable data pull/push operations.

Leveraging effective transparency. As previously stated, Lithops leverages AWS Lambda and S3 as its default backend, yet its design is not dependent on these specific services. To evaluate its architectural flexibility, we design an I/O-intensive analytics experiment that outlines three key aspects: (1) the performance characteristics of Lithops' default AWS Lambda and S3 configuration, (2) its seamless integration capabilities with alternative cloud

providers and services, and (3) the practical transparency achieved through its abstracted Storage API.

For our experimental evaluation, we employ a 5GB TeraSort workload. This benchmark is I/O-intensive, comprising two parallel stages of functions connected by an all-to-all communication pattern. We conduct this experiment across three distinct setups:

1. AWS Lambda utilizing AWS S3 for intermediate storage.
2. Google Cloud Run utilizing Google Cloud Storage for intermediate storage.
3. AWS Lambda utilizing Redis as the intermediate storage. To streamline deployment and management, we provision Redis as an AWS ElastiCache [7] managed cluster.

Crucially, the experimental methodology maintains identical application code across all three setups, based on the original implementation detailed in M. Sánchez-Artigas and G. T. Eizaguirre [8]. This serves as a direct demonstration of Lithops' transparency and portability. The only difference is the Lithops configuration file, which specifies the cloud provider and the storage backend to use.

Our empirical results, depicted in Figure 3, show that Google Cloud Run exhibits demonstrably inferior performance compared to the AWS-based configurations. Conversely, the performance of AWS Lambda with S3 is marginally slower than that of AWS Lambda with Redis, underscoring the competitive performance of Lithops' default setup. However, it is important to note that execution variability was significantly higher when using S3, which can likely be attributed to the throughput constraints associated with object storage systems.

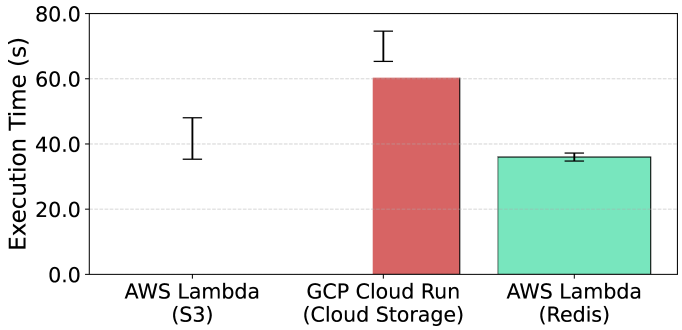


Figure 3. *TeraSort performance of Lithops across different cloud providers and storage backends (in parentheses).*

2.4. Source code

The entire Lithops source code is openly accessible to the public via its GitHub repository: https://github.com/lithops-cloud/lithops.

3. Optimized Data Partitioning with Dataplug

Scientific computing in the cloud is becoming increasingly data-driven. The success of large-scale data-parallel workloads depends not only on scalable computation, but also on scalable data access. In this section, we examine the limitations of traditional static data partitioning, introduce the concept of cloud-aware formats, and describe how the Dataplug [9] framework enables dynamic on-the-fly partitioning of unstructured scientific data. We conclude with an overview of Dataplug operational flow, allowing users to efficiently access and process large unstructured scientific datasets.

3.1. Static Partitioning

Static data partitioning is the traditional approach to data distribution in parallel serverless computing. It consists of physically splitting datasets into smaller chunks ahead of time, often based on size (e.g., 1 GB per file), and storing these chunks as individual objects (as shown on top of Figure 4). This enables distributed execution frameworks like Lithops or Dask [10] to process data in parallel by assigning each file to a separate worker.

While conceptually simple, static partitioning comes with significant limitations:

1. High preprocessing cost: Partitioning usually requires reading, transforming, and re-writing large datasets back to storage. In cloud object storage, where data is immutable, this implies full data duplication, leading to high I/O, storage, and compute costs.
2. Inflexibility: Once partitioned, the chunk size is fixed. Adapting to new workloads (e.g., smaller partitions for finer-grained parallelism or larger ones for fewer tasks) requires reprocessing and rewriting the dataset again.

These limitations become increasingly problematic as scientific datasets grow to the huge scale. In practice, many research teams cannot afford the cost of rewriting legacy datasets just to enable parallel access. Moreover, partitioning choices made early in a project may later hinder performance optimization, preventing effective use of scalable resources.

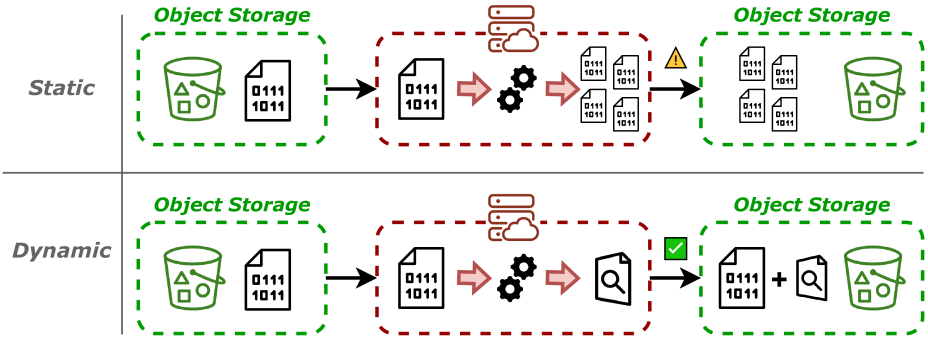


Figure 4. *Static and dynamic partitioning approaches.*

3.2. Cloud-Aware Formats Approach

One proposed solution to partitioning data is to adopt so-called *cloud-optimized formats*. These are file formats specifically designed to support partial reads (i.e. queries) using HTTP range requests over object storage. Examples include some Cloud-Optimized Formats as Cloud-Optimized GeoTIFF (COG) and Cloud-Optimized Point Cloud (COPC). These formats allow users to access data in parallel without needing to rewrite it, as they can be read directly from object storage using HTTP range requests. This enables efficient data access patterns, such as reading only the required chunks for a specific analysis, without incurring the cost of full dataset duplication.

While effective, this approach has its own challenges. Converting existing datasets to cloud-optimized formats requires a full rewrite of the data, which can be prohibitively expensive for huge-scale archives. Moreover, not all scientific formats support casting to a cloud-optimized format—for example, text-based genomic formats like FASTQ or compressed binary LAS files for spatial data.

So, the alternative head towards a new model: dotate legacy formats with cloud-awareness, enabling dynamic partitioning without rewriting the data. This approach allows users to

access and process large datasets in parallel without incurring the high costs of static partitioning.

3.3. Dataplug: Enabling Dynamic Partitioning in Scientific Formats

Dataplug is a Python framework that implements cloud-aware, on-the-fly dynamic partitioning model. The key idea is to maintain legacy data blobs without modification while enabling efficient access patterns. Rather than rewriting the data into a new format, Dataplug extracts structural metadata from the raw files via read-only preprocessing and stores this information externally (e.g., in a database or a metadata bucket). This enables parallel access to the original object without modifying it.

There are three essential building blocks that, in turn, compose the three key directives of the Dataplug framework.

- Cloud-aware preprocessing (flow shown in bottom of Figure 4): Each supported format defines a preprocessing heuristic that analyzes the file structure (e.g., sequence boundaries, block offsets, spatial extents) and generates the metadata. This process is one-time and non-intrusive—it never rewrites the input data.
- Partitioning strategies: Once preprocessed, a dataset can be partitioned in multiple ways using user-defined strategies. A partitioning function queries the metadata and defines logical partitions (called slices). For instance, one strategy might divide genomic data into 1 million reads per slice, while another partitions spatial files by bounding box. These strategies are pluggable and workload-specific.
- Lazily-evaluated slices: Each slice represents a logical data partition. When a worker performs the get operation over the slice, it retrieves the corresponding byte ranges from object storage (using HTTP range requests) to the worker memory. This happens entirely at runtime, just-in-time for processing.

The partitioning and getting of slices mechanism is illustrated in Figure 5.

```
from dataplug import CloudObject
from dataplug.formats.genomics.fastq import FASTQGZip, partition_reads_batches

# Assign FASTQGZip data type for the object
co = CloudObject.from_s3(FASTQGZip, "s3://genomics/dataset.fastq.gz")

# Data must be pre-processed first ==> This only needs to be done once per dataset
co.preprocess()

# Partition the FASTQGZip object into 200 chunks
# This does not move data around, it only creates data slices from the indexes
data_slices = co.partition(partition_reads_batches, num_batches=200)

def process_fastq_data(data_slice):
    # Evaluate the data_slice, which will perform the
    # actual HTTP GET requests to get the FASTQ partition data
    fastq_reads = data_slice.get()
    ...

# Use Lithops to process the data slices in parallel
from lithops import FunctionExecutor
with FunctionExecutor() as fexec:
    # Map the process_fastq_data function to each data slice
    fexec.map(process_fastq_data, data_slices)
```

At the same time, the entire workflow explained in previous lines is straightforward to use via Dataplug API. The above code snippet illustrates how to use Dataplug to partition one genomic (FASTQGZip) file into several slices, and then process each slice in parallel using Lithops. In a very quick overview, the code above illustrates how to use Dataplug to partition a FASTQGZip file into 200 slices, and then process each slice in parallel using Lithops. First,

the `preprocess()` method generates the required metadata for the FASTQGZip file, then the `partition()` method creates the slices based on the specified partitioning strategy, and the `get()` method retrieves the data for each slice when called, in a lazy way.

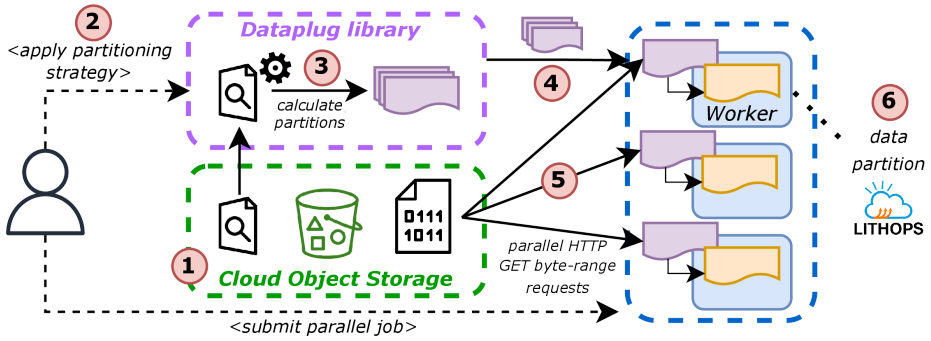


Figure 5. *Dataplug workflow.*

Dataplug approach offers several key advantages over static partitioning, that make it particularly well-suited for scientific data analytics:

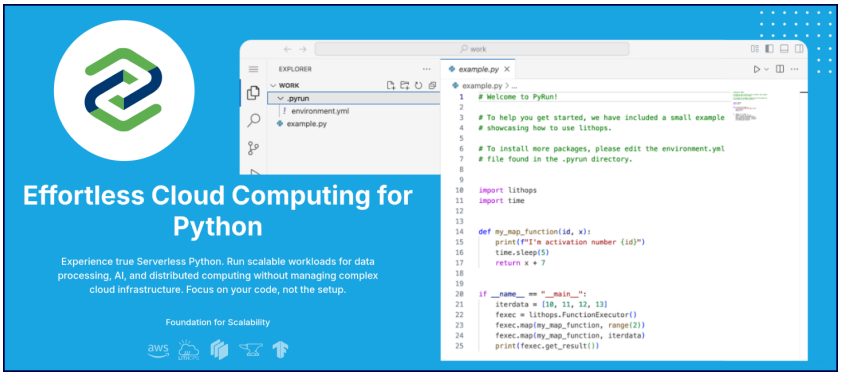
- Zero-cost repartitioning: Because partitioning operates over metadata, the same dataset can be re-sliced on demand with different strategies, without re-reading or re-writing the raw data.
- Efficient resource usage: The metadata is lightweight (often $< 1\%$ of the original file size), and slices can be distributed to any number of workers, enabling elastic autoscaling.
- Format extensibility: Support for new formats is added by implementing a preprocessor, a strategy, and a slice handler—without need to modify the core.

3.4. Source code

The entire Dataplug source code is openly accessible to the public via its GitHub repository: https://github.com/CLOUDLAB-URV/dataplug.

4. PyRun: Simplifying Serverless Environments for Data Science

In the same plane as serverless computing, which minimizes the infrastructure management burden, PyRun emerges as an integrated development environment (IDE) that further simplifies the development and execution of applications, particularly in the context of data science and scientific computing. By providing a ready-to-use, browser-based development environment, PyRun eliminates the need for users to configure or manage infrastructure, making it accessible even to those without DevOps experience. With a minimal configuration step, users can quickly set up their cloud environment and start developing applications without the complexities typically associated with cloud computing.



At a glance, the PyRun environment offers the following features:

- Ready-to-execute Web IDE: For each PyRun application, users are provided with a fully remote, web-based development environment accessible via browser. This environment is also natively synchronized with GitHub, enabling a seamless development workflow without the need for local installations. PyRun delivers a practical interface through an interactive programming notebook, similar to established tools like JupyterLab [11] or Google Colab [12], but with the added benefit of being pre-configured for cloud execution and automatic, cross-service runtime management.
- Pre-configured frameworks: PyRun currently supports native integration with parallel computing frameworks such as Lithops, Das and Dataplug, with plans to extend this list in the future. These integrations allow users to access powerful APIs without the overhead of manual pre-configuration, significantly accelerating the development and deployment process.
- Collaborative and Reproducible Workflows: PyRun's *Public Resources* feature promotes scientific collaboration and reproducibility by allowing users to share parallel applications and pipelines with the broader community. These shared resources also serve as a valuable compendium of programmatic patterns, which developers can adapt as building blocks for their own solutions.
- Monitoring: PyRun provides a built-in observability layer for scientific parallel applications. Users have access to a monitoring interface that displays detailed execution metrics, including task status, CPU/memory/network usage, failure reports, and runtime statistics.
- Simplified AI/ML workflows: PyRun facilitates AI/ML development by offering an integrated environment with automated infrastructure management. It supports widely-used machine learning frameworks such as TensorFlow and PyTorch, with straightforward runtime configuration. Scalable training and data processing can be achieved through Lithops-based distributed execution. Additionally, PyRun includes support for interactive LLM experimentation, allowing users to run models such as LLaMA 3 locally via Ollama within their workspace.

In summary, PyRun is a platform that simplifies the development and execution of serverless, data analytics and AI/ML applications. For further details, visit https://pyrun.cloud/.

5. Case Study: Python Serverless Data Analytics Empowering METASPACE Metabolomics Pipelines

One prominent example of leveraging serverless data analytics in Python is METASPACE [13], a cloud platform developed by the European Molecular Biology Laboratory (EMBL) to support spatial metabolomics research. This field aims to identify and localize

small molecules—such as metabolites and lipids—within tissue sections using imaging mass spectrometry (MS). These spatial insights are critical for understanding biological processes related to disease, development, and immune response

METASPACE allows researchers to submit MS datasets, automatically annotate detected molecules, and visualize their spatial distribution. It provides standardized quality control, enabling comparisons across experiments, laboratories, and instruments. The platform has evolved into a community-driven knowledge base of spatial metabolomes, supporting open data sharing and collaboration. With over 1,000 users and tens of thousands of processed datasets, METASPACE has become a widely adopted tool in biomedical research, making spatial metabolomics more accessible, reproducible, and scalable.

5.1. Lithops for Elasticity

METASPACE initially relied on a Spark cluster for its computational backend. However, as user numbers grew and workloads became more demanding, the limitations of this setup became increasingly evident. Spark's static resource allocation and rigid execution model made it difficult to handle large, diverse datasets efficiently. This often led to resource underutilization or performance bottlenecks, restricting the system's ability to scale effectively.

To address these issues, METASPACE transitioned to a serverless architecture, aiming for greater elasticity and ease of management. By adopting Lithops, the platform gained the ability to scale computing resources dynamically based on workload demands. This move significantly reduced operational overhead while improving performance and resource efficiency. The switch to a serverless model allowed METASPACE to overcome the constraints of its Spark-based infrastructure and provided a more responsive and scalable environment for data processing.

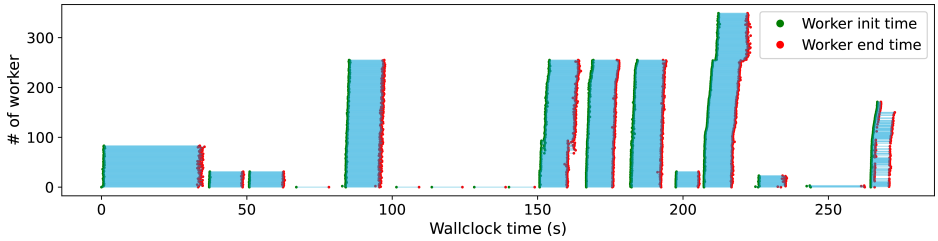


Figure 7. *Execution timeline for the METASPACE metabolomics pipeline.*

Figure 7 shows the execution timeline for a METASPACE metabolomics workload, a complex workflow with multiple parallel stages. The timeline reveals significant variability in resource requirements, with some steps requiring substantially more computational power than others. This fluctuation is a key driver for adopting serverless computing, a paradigm where Lithops excels by dynamically scaling resources to precisely match the needs of each pipeline step.

5.2. Dataplug for Dynamic Partitioning

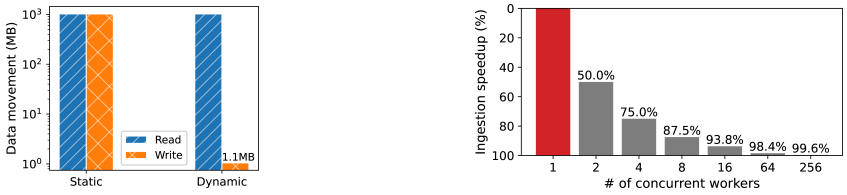


Figure 8. *Dataplug performance: preprocessing and ingestion.*

A common issue in metabolomics involves processing large, non-cloud-optimized datasets, such as files in the standard imzML format [14]. While widely used for mass spectrometry imaging, imzML lacks native support for cloud-aware access patterns. The Dataplug framework overcomes this limitation with a plugin that enables efficient, on-the-fly partitioning of imzML files. By extracting key metadata, Dataplug facilitates dynamic partitioning strategies tailored to the specific need of each workload, eliminating the need to create static, pre-processed data chunks. This approach yields significant advantages: it reduces preprocessing time and storage costs (Figure 8) and enables parallel data access, which drastically cuts data ingestion times (Figure 8b). Furthermore, it offers researchers the flexibility to experiment with different partitioning schemes without modifying the original dataset.

5.3. Practical Insights of a Serverless-native Architecture

To better illustrate the structural benefits of using Lithops, we compared the performance of the original METASPACE Spark-based pipeline with the new Lithops-based implementation. For the Spark execution we used PySpark over the original cluster by EMBL, consisting on four c5.4xlarge AWS EC2 instances. This architecture provided a sum of 64 vCPUs and 128GB of physical memory. We measured the performance, average CPU usage and cost of both deployments. We could assess the resource usage of the Lithops implementation by using the Lithops monitoring [1] interface, which provides detailed metrics on CPU usage (along with memory and network) for each function invocation. The cost was calculated based on the AWS Lambda pricing model, which charges per 1s of execution time and per GB of memory used, and we included the read/write requests to the S3 object storage, which are charged separately.

Table 1. *Execution insight comparison of the PySpark and Lithops deployments of the METASPACE pipeline.*

Architecture	Execution Time (min)	Cost ($)	Avg Resource Usage (%)
PySpark	14.21	0.66	54.56
Lithops Serverless	4.87	1.35	75.36

We achieved a significant performance improvement with the Lithops serverless implementation, reducing the execution time from 14.21 minutes to just 4.87 minutes, which represents a 65% speedup. This improvement is largely due to the dynamic scaling capabilities of Lithops, which allowed us to run many more parallel tasks than the Spark cluster could handle. The average resource usage also increased from 54.56% to 75.36%, indicating that the Lithops implementation was able to utilize resources more efficiently. However, the cost of the Lithops implementation was higher, at $1.35 compared to $0.66 for the Spark cluster, mainly due to the increased number of parallel vCPUs provisioned.

Despite this, the overall performance gain and resource efficiency make the Lithops serverless architecture a compelling choice for sporadic, and bursty workloads like the METASPACE metabolomics pipeline. Provisioning a static cluster for such workloads would have resulted in significant resource underutilization and higher costs in the long term, as the cluster would have been idle most of the time. In contrast, Lithops allows us to provision resources immediately and in real time, as annotation workload requests are submitted. We depict such advantage in Figure 9, where we simulate the monthly savings switching from a static cluster to a Lithops deployment, provisioning resources on demand on a per-workload basis.

[1] https://lithops-cloud.github.io/docs/source/monitoring.html

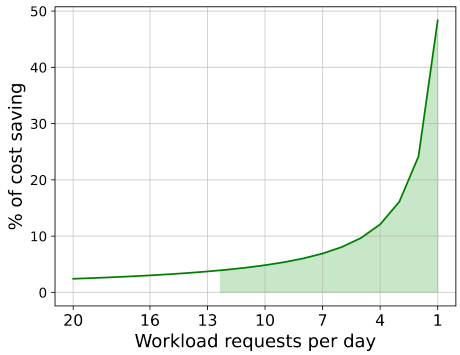


Figure 9. *Simulated monthly cost savings: ratio of the monthly cost between a static PySpark deployment vs. an on-demand Lithops deployment for varying workload frequencies.*

Lithops delivers substantial cost savings at low workload frequencies—achieving nearly a 50% reduction in monthly costs when running a single workload per day. Even as the frequency increases (e.g., up to 20 requests per day), Lithops continues to provide cost benefits over the static approach. It is important to note that this simulation assumes a constant workload size per request; in real-world scenarios like METASPACE, where workload sizes can vary significantly, Lithops' elasticity would further amplify these savings by dynamically allocating resources to match each workload's actual requirements.

In terms of usability, some METASPACE applications have been integrated into PyRun to provide a user-friendly interface for running these serverless data pipelines. PyRun hinders the complexity of managing the environment and its dependencies, allowing researchers to focus on their data analysis tasks without worrying about configuration issues. This integration has made it easier for scientists to leverage the power of serverless computing and dynamic partitioning in their research workflows.

6. Conclusion

Ultimately, harnessing the power and advantages of serverless computing for data analytics in Python has transformed how scientists and developers tackle large-scale data processing. With production-ready tools like Lithops, which enables efficient and scalable execution of functions in the cloud; Dataplug, which simplifies partitioning of complex scientific datasets not originally designed for cloud environments; and PyRun, which streamlines development environment setup, a robust ecosystem has emerged that allows users to focus completely on their code, rather than the underlying infrastructure.

References

[1] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop Distributed File System," in *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, 2010, pp. 1–10. doi: 10.1109/MSST.2010.5496972.

[2] M. Zaharia *et al.*, "Apache Spark: a unified engine for big data processing," *Commun. ACM*, vol. 59, no. 11, pp. 56–65, 2016, doi: 10.1145/2934664.

[3] D. Barcelona-Pons, A. Arjona, P. Garcia Lopez, E. Molina-Giménez, and S. Klymonchuk, "Burst Computing Artifact (paper in USENIX ATC'25)." [Online]. Available: https://doi.org/10.5281/zenodo.15423616

[4] E. Jonas, Q. Pu, S. Venkataraman, I. Stoica, and B. Recht, "Occupy the cloud: distributed computing for the 99%," in *Proceedings of the 2017 Symposium on Cloud Computing*, in SoCC '17. Santa Clara, California, 2017, pp. 445–451. doi: 10.1145/3127479.3128601.

[5] J. Sampé, M. Sánchez-Artigas, G. Vernik, I. Yehekzel, and P. García-López, "Outsourcing Data Processing Jobs With Lithops," *IEEE Transactions on Cloud Computing*, vol. 11, no. 1, pp. 1026–1037, 2023, doi: 10.1109/TCC.2021.3129000.

[6] S. Sanfilippo and R. community, "Redis," 2009.

[7] Amazon Web Services (AWS), "Amazon ElastiCache: Managed InMemory Data Store & Cache," 2025. [Online]. Available: https://aws.amazon.com/elasticache/

[8] M. Sánchez-Artigas and G. T. Eizaguirre, "A seer knows best: optimized object storage shuffling for serverless analytics," in *Proceedings of the 23rd ACM/IFIP International Middleware Conference*, in Middleware '22'. Quebec, QC, Canada, 2022, pp. 148–160. doi: 10.1145/3528535.3565241.

[9] A. Arjona, P. García-López, and D. Barcelona-Pons, "Dataplug: Unlocking extreme data analytics with on-the-fly dynamic partitioning of unstructured data," in *2024 IEEE 24th International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, 2024, pp. 567–576. doi: 10.1109/CCGrid59990.2024.00069.

[10] M. Rocklin, "Dask: Parallel Computation with Blocked Algorithms and Task Scheduling," in *Proceedings of the 14th Python in Science Conference*, K. Huff and J. Bergstra, Eds., 2015, pp. 130–136. doi: 10.25080/Majora-7b98e3ed-013.

[11] Jupyter Development Team, "JupyterLab: A Next-Generation Notebook Interface," 2025. [Online]. Available: https://github.com/jupyterlab/jupyterlab

[12] Google LLC, "Google Colaboratory," 2025. [Online]. Available: https://colab.research.google.com/

[13] T. Alexandrov *et al.*, "METASPACE: A community-populated knowledge base of spatial metabolomes in health and disease." 2019. doi: 10.1101/539478.

[14] T. Schramm *et al.*, "imzML – A Common Data Format for the Flexible Exchange and Processing of Mass Spectrometry Imaging Data," *Journal of proteomics*, vol. 75, pp. 5106–5110, 2012, doi: 10.1016/j.jprot.2012.07.026.

Open Access | https://doi.org/10.25080/ectn7568

SciPy 2025

July 7 - July 13, 2025

Proceedings of the 24th Python in Science Conference
ISSN: 2575-9752

An Active Learning Plugin in napari to Fine-Tune Models for Large-scale Bioimage Analysis

Fernando Cervantes-Sanchez[1]

[1]The Jackson Laboratory

Abstract

The "napari-activelearning" plugin provides a framework to fine-tune deep learning models for large-scale bioimage analysis, such as digital pathology Whole Slide Images. The development of this plugin was motivated by easing the integration of deep learning tools into bioimage analysis workflows. This plugin implements the concept of Active Learning for reducing the time spent on labeling samples when fine-tuning models. Because this plugin is integrated into napari and leverages the use of Next Generation File Formats (e.g. Zarr), it is suitable for fine-tuning deep learning models on large-scale images with little image preparation.

Keywords deep learning, active learning, napari

1. Introduction

Adoption of deep learning methods for bioimage analysis has grown exponentially in recent years. Part of such success is thanks to transfer learning that enables using models that were trained on large volumes of data from diverse domains, such as the ImageNet [1] and Segment Anything 1 Billion (SA-1B) [2] datasets, into tasks where annotated data is scarce. An example is cell segmentation in biological microscopy images [3], which requires human annotation of cell structures in images of different modalities and scales. Such images can be up to tens or hundreds of thousands of pixels per side (e.g. WSI used in histopathology), depending on the acquisition magnification and imaging modality. Moreover, the research community has made considerable efforts during the last years to curate databases for training models relevant to the bioimage analysis field. These databases include LIVECell [4], TissueNet [3], and CellSeg [5], among many others. Such databases can be used for de-novo training or be used for fine-tuning models that were trained with image datasets from general domains.

1.1. Related work

Segmentation of structures in biological image data is a recurrent task in bioimage analysis that serves as an intermediate step for downstream applications. Some of the most common downstream applications for cell segmentation involve cell type classification, counting specific types of cells, and measuring morphological properties of cells in an image. These applications require high quality cell segmentations to obtain accurate information that is used in further bioimage understanding and research. The most relevant deep learning segmentation methods include Cellpose [6], Stardist [7], and Micro-SAM [8]. These methods offer pre-trained models for carrying out segmentation of biological structures in multiple imaging modalities, and tools for fine-tuning those same models to new data with user-defined annotations. Training deep learning models for bioimage analysis involves: 1)

Published Jul 10, 2025

Correspondence to
Fernando Cervantes-Sanchez
fernando.cervantes@jax.org

Open Access

Copyright © 2025 Cervantes-Sanchez. This is an open-access article distributed under the terms of the Creative Commons Attribution 4.0 International license, which enables reusers to distribute, remix, adapt, and build upon the material in any medium or format, so long as attribution is given to the creator.

extracting several rectangular sections of a specified size (i.e. image tiles) from the original image files, 2) annotating each of those tiles according to the tasks being learned, and 3) storing the pairs of tiles and annotations in separate containers or folders as training and testing datasets. However, this approach involves some technical difficulties such as the costs and time associated with transferring training data between researchers for sharing and reproducibility purposes, costs for storing duplicated data from the image tiles already present in the original images, and lack of context and coordinates from where each tile was originally extracted.

The field of Active Learning studies human-in-the-loop strategies in deep learning that can reduce the time and effort required for de-novo training or fine-tuning models. Applications of active learning have also been developed to address computer vision tasks [9]. This is relevant to biological image analysis where data annotation is one of the most time-consuming tasks [10]. The main reason is that due the scale of the images in some modalities (e.g. WSI), the number of samples that can be extracted for labeling can overwhelm the annotator. On the other hand, for relatively smaller images acquired through other modalities, the annotator would require to annotate large amounts of samples to generate sufficient training data. An active learning sampling strategy, based on acquisition functions, can be used to prevent this by presenting a limited number of samples at a time. Moreover, there are acquisition functions such as BALD [11] that focus on obtaining only cases that could improve a model's performance when correctly labeled.

The bioimage analysis community has been steering towards efficient creation and sharing of repositories containing large volumes of data. NGFFs [12] are one of the most preferred options for large-scale image storage and management, such as the OME-Zarr data format [13]. The Zarr data format establishes a standard structure for compressed, chunked-based, n-dimensional data storage that can be efficiently stored either in local or cloud repositories [13]. Multiple projects have been created on top of Zarr, including APIs such as the zarr-python library [14]. Some image processing softwares have adopted this data format, like Fiji [15] with the MobIE plugin [16], and QuPath [17] with its own Zarr data loader. Visualization tools such as napari [18], neuroglancer [19], viv [20], and webKnossos [21] have also added support to this data format. Additionally, image data stored as Zarr can be used for training deep learning models without duplicating data. That can be achieved by accessing chunked image data instead of extracting image tiles and storing them separately as is commonly needed in standard deep learning training.

In this work, the "napari-activelearning" plugin for napari is introduced as a tool for easing fine-tuning existing models on large-scale images. This plugin leverages napari's user-friendly features for annotating new samples following an active learning workflow. Additionally, the "napari-activelearning" plugin uses NGFF to store annotations in a storage-efficient manner, ready for deep learning training.

2. Methods

The "napari-activelearning" plugin relies on three components to provide a user-friendly framework to train and fine-tune deep learning models for bioimage analysis: 1) NGFF, for efficient storage of the data generated during the active learning workflow, 2) napari, for visualization and graphical interface with the user, and 3) Active Learning, for assisting users with selection of samples of interest that require annotation for improving the deep learning model performance.

2.1. NGFFs

NGFFs [12], such as OME-Zarr [13], have been increasingly adopted by the bioimage analysis community thanks to its computation and storage advantages. Zarr format stores large-

scale image data as independent n-dimensional tiles, also called chunks, either on local disk or cloud storage. By using chunks as units of storage, the amount of data required to be loaded into memory when accessing specific regions of the image is reduced. Accessing image chunks is parallel-safe, which enables acceleration of image processing through parallel computing. This is useful when applying a model for inference in larger-than-memory image data, where deep learning inference can be applied to regions of the image separately, and applying a subsequent reduction operation on the results. This reduction operation could be an accumulation function for whole image classification, or a stitching algorithm for segmentation tasks.

2.2. napari Visualization Tool

napari [18] is a user-friendly n-dimensional data viewer with extensible capabilities via plugins. This visualizer already offers tools for data annotation, and it is compatible with Zarr, enabling visualization of large-scale image data efficiently. napari has been used to develop deep learning applications for bioimaging analysis such as napari-cellpose [6], and Micro-SAM [8] plugins. These plugins can be used to segment biological structures in data acquired with multiple imaging modalities. However, these plugins are intended to be used with tiles of images that have already been extracted and stored, in contrast to the plugin presented in this work, which can be applied on regions defined by the user directly on original images without storing duplicate data from image tiles.

2.3. Active Learning framework

To reduce the amount of data presented to the user for annotation, concepts from the Active Learning framework are implemented in this plugin. This field studies methods for human-in-the-loop learning workflows that avoid overwhelming the annotator with samples to labeling for training. This is achieved through a computation of Acquisition Functions that determine what samples require labeling to subsequently improve the performance of a specific model.

Some of the most used acquisition functions involve Random acquisition, which takes a random unlabeled sample from the dataset with an uniform distribution; Maximize mean standard deviation, that computes the average standard deviation of the probabilities of each class predicted by the model; Maximize variation ratios, which uses the probability of the predicted class of each sample; Maximize the predictive entropy, that is computed from the predicted probabilities for each sample; and Maximize the mutual information between the model parameters and the predicted classes, which is also known as BALD [11]. In this plugin, the BALD [11] acquisition function is implemented to score and sort a set of image patches sampled from an image. This acquisition function was selected because it was demonstrated that can be used efficiently with bioimage data for deep learning training when compared with other functions such as Random acquisition [9]. Such efficiency on selecting the most promising samples is highly desirable for increasing the amount of samples presented to the annotator that could improve the model's performance while reducing redundant samples. In this plugin, the score assigned to a sample extracted at random from the input image is computed following N. Houlsby, F. Huszár, Z. Ghahramani, and M. Lengyel [11]:

$$a(x, M) = \mathbb{I}[y, \theta \mid x, D], \tag{1}$$

where $a(x, M)$ is the acquisition function score for sample x for model M, the mutual information $\mathbb{I}$ is computed from the prediction y made by model M for sample x in the dataset D with the current state of parameters θ. Following Y. Gal, R. Islam, and Z. Ghahramani [9], the mutual information can be computed through Monte Carlo integration to compute the BALD function as follows:

$$\mathbb{I}[y, \theta \mid x, D] = -\sum_{c} \left(\frac{1}{T} \sum_{t} \hat{p_c}^{\,t} \right) log \left(\frac{1}{T} \sum_{t} \hat{p_c}^{\,t} \right) + \frac{1}{T} \sum_{c,t} \hat{p_c}^{\,t} log\, \hat{p_c}^{\,t}, \tag{2}$$

where T is the total number of steps in the Monte Carlo integration, $\hat{p_c}^{\,t}$ is the component of the prediction made by model M for sample x at class c. To apply the Monte Carlo integration, the prediction made with model M needs to be converted into a gaussian process. In this plugin, this is done by introducing a Dropout operation after each activation layer found in the deep learning model M [22]. Such introduction of Dropout operations are handled automatically by this plugin as follows: 1) Dropout layers are enabled in evaluation time for models that already have Dropout layers inside their architecture, or 2) Dropout layers are introduced with a default probability of 0.05 after each activation layer found in the model's module layers.

The active learning process implemented in this plugin is illustrated in Figure 1.

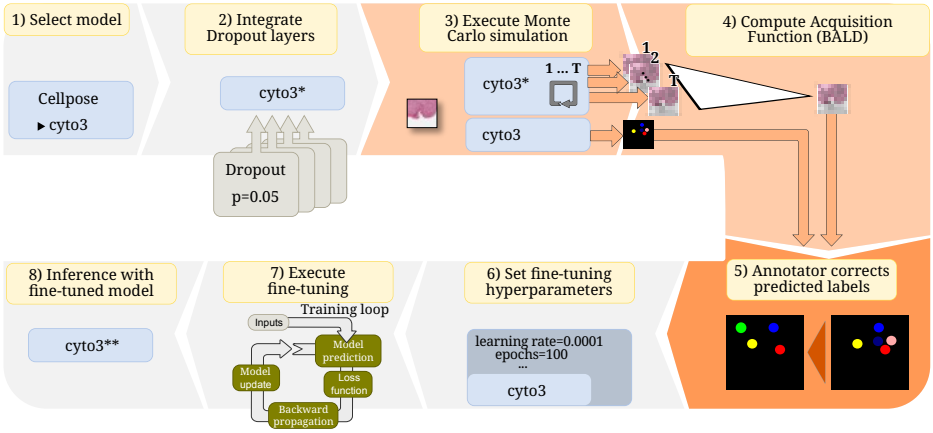


Figure 1. *Flowchart of the active learning framework implemented in the "napari-activelearning" plugin. The process is the following: 1) the user selects a model from the models registry in the plugin, 2) the plugin integrates Dropout layers into a bayesian version of the model (cyto*), 3) prediction on a sample image patch is repeated T times with the bayesian model (cyto3*) in a Monte Carlo simulation process; additionally, the baseline model (cyto3) is used to evaluate the same sample image patch once to obtain the objects labels, 4) the BALD acquisition function is computed from the results of the Monte Carlo simulation process; the resulting BALD acquisition function and objects labels are added to napari's layers list separately, 5) the annotator selects the next patch with the highest BALD score and corrects its labels using napari's builtin annotation tools, 6) the user sets up the hyper-parameters of the fine-tuning process corresponding to the baseline model (cyto3) in the plugin, 7) the wrapped fine-tuning process from the selected model is executed using the corrected labels as ground-truth, 8) the sample image patches are predicted again with the fine-tuned model (cyto3**) for comparison. Note that steps 3 and 4 are repeated for a number of times defined by the annotator to extract sufficient sample patches from the images. Additionally, the annotator can decide how many samples to correct in step 5, as well as skip labels with low BALD score.*

2.4. Implementation details

The "napari-activelearning" plugin is an open-source project implemented in the Python programming language that relies on the napari plugin architecture to offer a user-friendly interface. This plugin consists of three main component widgets that allow the execution of an end-to-end fine-tuning process from within a napari window. Such components are 1) an image groups manager widget, 2) an acquisition function configuration widget, and 3) a labels manager widget.

2.4.1. *Image Groups Manager*

This component is used to gather the metadata of the images used for training and fine-tuning into a single data structure called “image group”. The purpose of this structure, shown in Figure 2, is to define how each layer shown on the napari window will be used in the active learning workflow, such as input data, labels or annotations, and sampling masks. The gathered metadata is used to connect the input data with the selected deep learning model. This is done in this plugin by streaming the pixel data from the input image in the structure and format that the model expects it. The reformating and structuring process commonly involve reordering the spatial axes and color channels of the image data, and even casting the data into a different data type required by the model. The metadata also contains the origin of the image data, which can be a path to an image file on disk or a location in a remote storage. The origin is used by this plugin to access the pixel data directly avoiding duplicating the data when possible. Moreover, sampling masks allow the user to define the regions in the image from where the plugin can sample image patches to be used through the fine-tuning process.

Figure 2. *Image groups manager widget used to manage metadata of napari’s layers to be used for inference and fine-tuning along with the Acquisition Function Manager component. This widget is composed of four main components: a) a set of buttons to create and add image groups, layer groups, and layer channels, b) an editor that allows to define metadata related to each of the layers in the image groups, c) an editor to create sampling masks that allow to focus the active learning process on specified regions of an image defined by the user, and d) a list of the layers added to each image group along with their specific use, such as input image, labels for training, and sampling masks.*

2.4.2. *Acquisition Function Manager*

In this component, the model used for active learning can be selected from a list of registered models, and their hyper-parameters can be configured according to the user’s needs. This component uses NumPy [23] and PyTorch [24] libraries to implement the BALD [11] acquisition function and the Dropout operation insertion [22]. The outputs generated in the intermediate steps of the active learning framework are transformed using the Sci-kit Image library [25] and stored using the OME-Zarr [13] specification to make them shareable and reproducible under FAIR [26] guidelines. This component computes the acquisition function score for a set of image patches sampled from the input image defined in the Image Groups Manager component. The number of samples extracted and the total steps for the Monte Carlo integration process can be defined by the user in this widget, as shown in Figure 3. For each sampled patch, the inference and acquisition function are computed and presented to the user in a descending list on the Labels Manager widget. The “napari-activelearning” uses Cellpose [6] as default deep learning framework for inference and fine-tuning with its pre-trained models. Moreover, the code that implements this component is intended for its extension to other deep learning models with PyTorch backend [24]. This can be achieved through the inheritance of a segmentation method class (`TunableMethod`) that is used to execute pure inference, probability computations, and even fine-tuning of model weights. As an example, the integration of Cellpose models into the napari-activelearning plugin is illustrated in Figure 4. The hyper-parameters that appear at the right in Figure 3

d) had been defined in the fine-tuning function (Figure 4 c) when integrating the Cellpose model into this plugin.

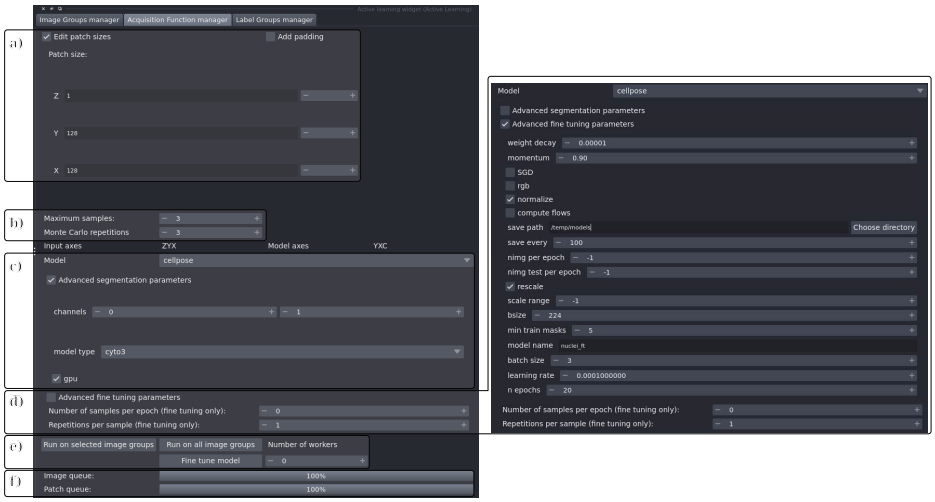


Figure 3. *Acquisition function manager widget used mainly to execute model inference and fine-tuning, along with configuration of model parameters and the active learning sampling parameters. The acquisition function manager widget is composed of six main components that allow a) select the size of the image patches extracted from the input images, b) determine the number of patches that are randomly extracted from the image and the number of repetitions in the Monte Carlo simulation for estimating the acquisition function value, c) configure the inference parameters of the selected deep learning model, d) configure the hyper-parameters for the fine-tuning process, e) execute the inference step on a single image, or all the images added to the image groups manager, as well as execute the fine-tuning process once the labels have been corrected, and f) visualize the progress of the inference step as it is applied to the images added to the image groups manager.*

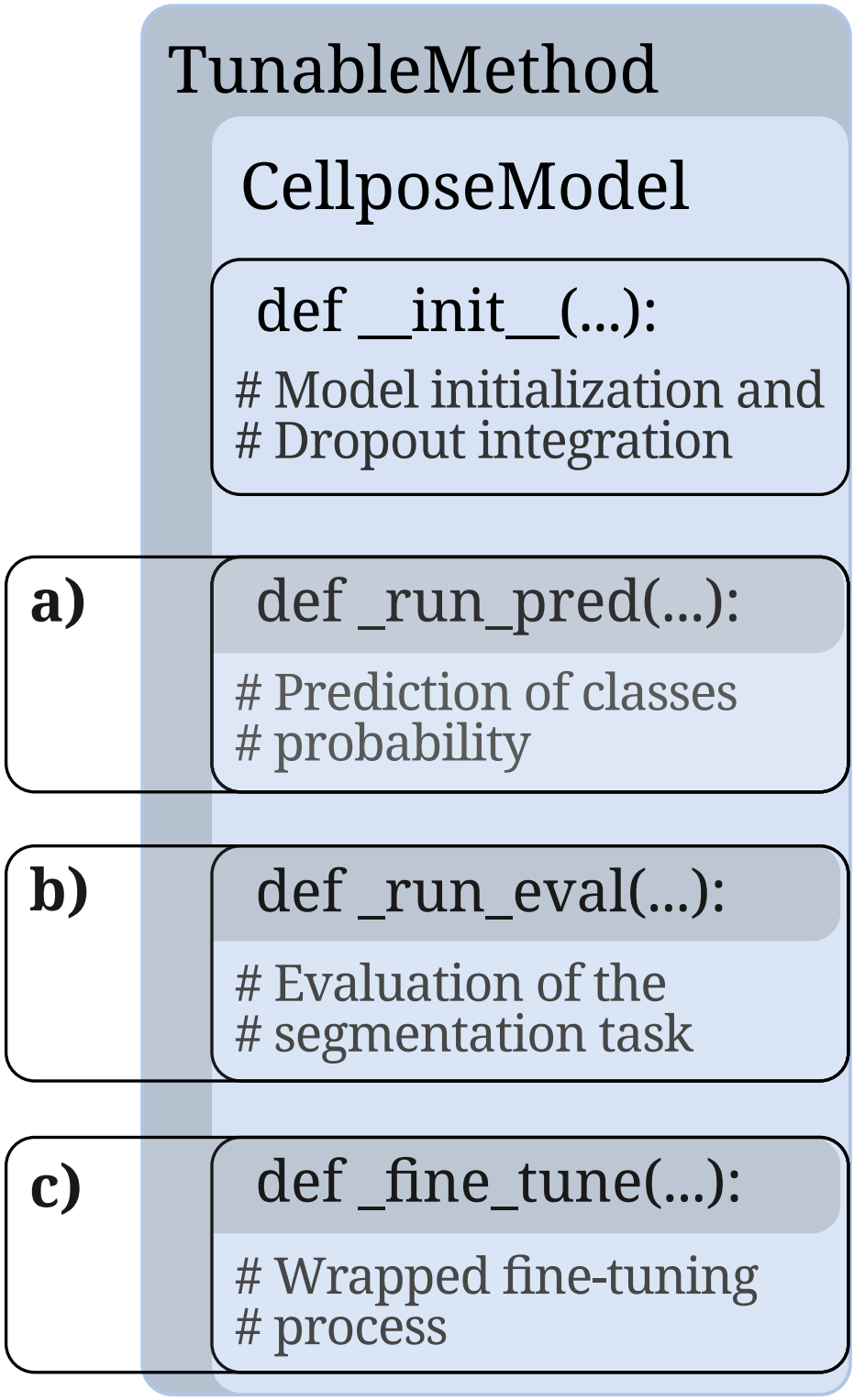


Figure 4. *Visual representation of the integration of Cellpose models as tunable methods in the napari-activelearning plugin through inheritance of the* `TunableMethod` *class. The derived class (*`CellposeModel`*) is required to implement: a) the* `_run_pred` *function that predicts the probability of each instance being assigned to the distinct domain classes; in the case of Cellpose that is the probability of each individual pixel being assigned to the foreground or background class, b) the* `_run_eval` *function that executes the*

normal evaluation process of the Cellpose model, and c) the `_fine_tune` function that wraps the existing fine-tuning process implemented in the Cellpose Python package [6].

2.4.3. *Labels Manager*

After the samples are extracted and their acquisition function scores and inferences are computed by the Acquisition Function Manager, these can be reviewed by the user. The labels predicted by the selected model can be corrected using built-in napari annotation tools. These corrected annotations can be used as new targets for fine-tuning the selected model. Additionally, this component provides a simple navigation system to move between inferred labels in space as presented in Figure 5. This system is ideal for reviewing the output generated by the selected model across the image, which can be time consuming in large-scale images, such as WSI or three-dimensional image data.

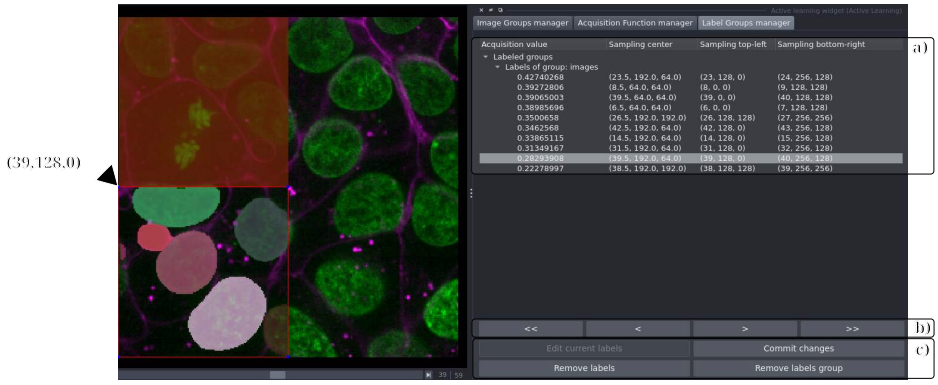


Figure 5. *Label groups manager widget used for handling sampled patches for labels correction following an active learning workflow. This widget is composed of three main components: a) the list of "label groups" that displays information of the location of the labeled image patches and their corresponding acquisition function value, b) a set of navigation buttons to move between labeled image patches, and c) a set of buttons to edit, commit changes, and remove labels and labels groups. Note that the currently selected label (at coordinates $Z = 39$, $Y = 128$, and $X = 0$ as displayed in the "Sampling top-left" column of this widget) is in edit mode, which allows to modify the labels of the objects inside the active region without altering the labels of any surrounding image patch.*

3. Results

The "napari-activelearning" source code is open source and available to the research community at the http://github.com/thejacksonlaboratory/activelearning repository. This package is distributed through PyPI (https://pypi.org/project/napari-activelearning) and can also be installed using the napari plugin manager.

3.1. Fine-tuning a Cellpose model

A comparative analysis was carried out to show the benefit of using the active learning framework implemented in this plugin for fine-tuning a deep learning model for bioimage analysis. The experiments consisted of fine-tuning the Cellpose's Cyto3 model for cell segmentation using training sets of different sizes: 5, 10, 15, 20, 25, and 30 images. The images used in the experiments are part of the Cellpose dataset (https://www.cellpose.org/dataset) [6]. This dataset was created with images from multiple domains and imaging modalities; however, only a set of 89 images for training and 11 images for testing were used in these experiments. The subset of images selected share the same imaging modality and domain, and it serves as an use case of fine-tuning a model to carry out cell segmentation on a specific imaging modality. The training sets with different sizes were created using the images with the top 5, 10, 15, 20, 25, and 30 BALD score values computed for every image

in the pool of 89 training images. The segmentation performance of the different variants of the Cyto3 model were measured in terms of the F1-score computed on the testing set of 11 images. Additionally, a paired Wilcoxon rank test was performed to determine the statistical meaningfulness of the difference between the baseline and fine-tuned models' segmentation performance. The results of the computational experiments are presented in Table 1.

Table 1. *Comparison of the segmentation performance, in terms of F1-score, between the baseline Cyto3 model from Cellpose and fine-tuned version of the same model that were trained with 5, 10, 15, 20, 25, and 30 images taken from a pool of 89 training images. The images used to fine-tune the Cyto3 model were selected based on the highest BALD score computed with this plugin.*

Model	Version	Training size	Training F1-score	Training Wilcoxon rank test (p-value)	Testing F1-score	Testing Wilcoxon rank test (p-value)
Cyto3	baseline	•	0.8561	•	0.8447	•
Cyto3	fine-tuned	5	0.8268	0.9293	0.8217	0.6036
Cyto3	fine-tuned	10	0.8838	0.0032	*0.8800*	*0.1620*
Cyto3	fine-tuned	15	0.8912	$8.18e^{-5}$	0.8785	0.2057
Cyto3	fine-tuned	20	0.8937	$3.06e^{-5}$	**0.9013**	**0.0654**
Cyto3	fine-tuned	25	0.8971	$2.17e^{-5}$	0.8854	0.1185
Cyto3	fine-tuned	30	**0.8982**	$\mathbf{1.07e^{-5}}$	0.8908	0.0741

The comparative analysis results showed the plugin implementation of the active learning framework enables improving the performance of the Cyto3 model through fine-tuning. The results show that the cell segmentation performance of the fine-tuned model can achieve a F1-score of up to 0.9013 on the selected imaging modality. This is significantly greater than the F1-score of 0.8447 obtained by the baseline Cyto3 model (p-value of 0.0654). Moreover, the computational experiments suggest that the active learning plugin can reduce the amount of samples required to be labeled in order to improve the baseline model performance. For this use case, labeling only 10 of the 89 images in the training pool is sufficient to outperform the baseline model performance in the test set (F1-score of 0.8800).

4. Conclusion

In this work, the "napari-activelearning" plugin has been introduced for carrying out fine-tuning of deep learning models for large-scale bioimage data. The napari visualization tool was used to host the plugin and serve as a graphical user interface. Additionally, by leveraging NGFF, the annotated data is stored efficiently by creating only image chunks that contain any labels information and avoiding writing empty chunks. Management of the data created through the active learning workflow is annotated following the OME-Zarr v2 specification making it ready for sharing and reproduction. Whereas this plugin was developed to facilitate adoption of deep learning models in bioimage analysis, it is not restricted to these imaging modalities and can be extended to be used with any image stored in the Zarr format. Moreover, a comparative analysis showed that this plugin can be used to improve the performance of a baseline cell segmentation model, reaching a F1-score of 0.9013 while requiring to annotate only 20 images from a pool of 89 images to achive this performance. Finally, the plugin can be used for transfer learning or simply as an interface for deep learning methods that lack graphical user interfaces for inference. A tutorial on how to use this plugin for fine-tuning a Cellpose model to carry out nuclei segmentation on the Cells 3D+2 Channels image from Sci-kit Image (included as a sample image in napari)

can be found in the supporting documents section of this paper and on the plugin's documentation website at https://thejacksonlaboratory.github.io/activelearning/tutorials.html.

References

[1] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255. doi: https://doi.org/10.1109/CVPR.2009.5206848.

[2] A. Kirillov *et al.*, "Segment Anything," in *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023, pp. 3992–4003. doi: https://doi.org/10.1109/ICCV51070.2023.00371.

[3] N. F. Greenwald *et al.*, "Whole-cell segmentation of tissue images with human-level performance using large-scale data annotation and deep learning," *Nat Biotechnol*, vol. 40, no. 4, pp. 555–565, 2021, doi: https://doi.org/10.1038/s41587-021-01094-0.

[4] C. Edlund *et al.*, "LIVECell—A large-scale dataset for label-free live cell segmentation," *Nature Methods*, vol. 18, no. 9, pp. 1038–1045, 2021, doi: https://doi.org/10.1038/s41592-021-01249-6.

[5] M. Y. Lee *et al.*, "CellSeg: a robust, pre-trained nucleus segmentation and pixel quantification software for highly multiplexed fluorescence images," *BMC Bioinformatics*, vol. 23, no. 1, p. 46, 2022, doi: https://doi.org/10.1186/s12859-022-04570-9.

[6] C. Stringer, T. Wang, M. Michaelos, and M. Pachitariu, "Cellpose: a generalist algorithm for cellular segmentation," *Nature Methods*, vol. 18, no. 1, pp. 100–106, 2021, doi: https://doi.org/10.1038/s41592-020-01018-x.

[7] M. Weigert and U. Schmidt, "Nuclei Instance Segmentation and Classification in Histopathology Images with Stardist," in *The IEEE International Symposium on Biomedical Imaging Challenges (ISBIC)*, 2022. doi: https://doi.org/10.1109/ISBIC56247.2022.9854534.

[8] A. Archit *et al.*, "Segment Anything for Microscopy," *Nature Methods*, vol. 22, no. 3, pp. 579–591, 2025, doi: https://doi.org/10.1038/s41592-024-02580-4.

[9] Y. Gal, R. Islam, and Z. Ghahramani, "Deep Bayesian Active Learning with Image Data." [Online]. Available: https://arxiv.org/abs/1703.02910

[10] S. Budd, E. C. Robinson, and B. Kainz, "A survey on active learning and human-in-the-loop deep learning for medical image analysis," *Medical Image Analysis*, vol. 71, p. 102062, 2021, doi: https://doi.org/10.1016/j.media.2021.102062.

[11] N. Houlsby, F. Huszár, Z. Ghahramani, and M. Lengyel, "Bayesian Active Learning for Classification and Preference Learning," *ArXiv*, 2011, doi: https://doi.org/10.48550/arXiv.1112.5745.

[12] J. Moore *et al.*, "OME-NGFF: a next-generation file format for expanding bioimaging data-access strategies," *Nature Methods*, vol. 18, no. 12, pp. 1496–1498, 2021, doi: https://doi.org/10.1038/s41592-021-01326-w.

[13] J. Moore *et al.*, "OME-Zarr: a cloud-optimized bioimaging file format with international community support," *Histochemistry and Cell Biology*, vol. 160, no. 3, pp. 223–251, 2023, doi: https://doi.org/10.1007/s00418-023-02209-1.

[14] A. Miles *et al.*, "zarr-developers/zarr-python: v2.4.0." Zenodo, 2020. doi: https://doi.org/10.5281/zenodo.3773450.

[15] J. Schindelin *et al.*, "Fiji: an open-source platform for biological-image analysis," *Nature Methods*, vol. 9, no. 7, pp. 676–682, 2012, doi: https://doi.org/10.1038/nmeth.2019.

[16] C. Pape *et al.*, "MoBIE: A Fiji plugin for sharing and exploration of multi-modal cloud-hosted big image data," *bioRxiv*, 2022, doi: https://doi.org/10.1101/2022.05.27.493763.

[17] P. Bankhead *et al.*, "QuPath: Open source software for digital pathology image analysis," *Scientific Reports*, vol. 7, no. 1, p. 16878, 2017, doi: https://doi.org/10.1038/s41598-017-17204-5.

[18] N. Sofroniew *et al.*, "napari: a multi-dimensional image viewer for Python." Zenodo, 2025. doi: https://doi.org/10.5281/zenodo.15465370.

[19] J. Maitin-Shepard *et al.*, "google/neuroglancer:." Zenodo, 2021. doi: https://doi.org/10.5281/zenodo.5573294.

[20] T. Manz *et al.*, "Viv: multiscale visualization of high-resolution multiplexed bioimaging data on the web," *Nature Methods*, 2022, doi: https://doi.org/10.1038/s41592-022-01482-7.

[21] K. M. Boergens *et al.*, "webKnossos: efficient online 3D data annotation for connectomics," *Nature Methods*, vol. 14, no. 7, pp. 691–694, 2017, doi: https://doi.org/10.1038/nmeth.4331.

[22] Y. Gal and Z. Ghahramani, "Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning." [Online]. Available: https://arxiv.org/abs/1506.02142

[23] C. R. Harris *et al.*, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, 2020, doi: https://doi.org/10.1038/s41586-020-2649-2.

[24] A. Paszke *et al.*, “PyTorch: An Imperative Style, High-Performance Deep Learning Library.” [Online]. Available: https://arxiv.org/abs/1912.01703

[25] S. van der Walt *et al.*, “scikit-image: image processing in Python,” *PeerJ*, vol. 2, p. e453, 2014, doi: https://doi.org/10.7717/peerj.453.

[26] M. D. Wilkinson *et al.*, “The FAIR Guiding Principles for scientific data management and stewardship,” *Scientific Data*, vol. 3, no. 1, p. 160018, 2016, doi: https://doi.org/10.1038/sdata.2016.18.

Open Access | https://doi.org/10.25080/yunw5822

SciPy 2025

July 7 - July 13, 2025

Proceedings of the 24th Python in Science Conference
ISSN: 2575-9752

Advancing High Energy Physics Data Analysis with Julia -- A Case for JuliaHEP

Ianna Osborne[1], and **Jerry Ling**[2]

[1]Princeton University, [2]Harvard University

Abstract

For the past 25 years, the High-Energy Physics (HEP) community has steadily adopted Python as its primary language for data analysis, supported by compiled-backend libraries like NumPy and Awkward Array, along with other Python tools like Uproot for I/O. However, the Julia programming language offers a compelling alternative, addressing the two-language problem with C++-comparable performance and Python-like ease of use. JuliaHEP is an informal organization that aims to unify effort in developing Julia projects related to HEP, outlining its advantages, ongoing developments, and integration with existing Python-based tools.

Keywords Awkward Array, Julia, High-Energy Physics, Data Analysis

1. Introduction

The High-Energy Physics (HEP) community has long relied on Python and C++ for data analysis. While Python provides ease of use and a rich scientific ecosystem, it struggles with performance for large-scale analyses. C++, on the other hand, offers speed but comes with increased complexity and slower development cycles. Julia, a modern language designed for scientific computing, promises the best of both worlds: high-level expressiveness with near-native execution speed.

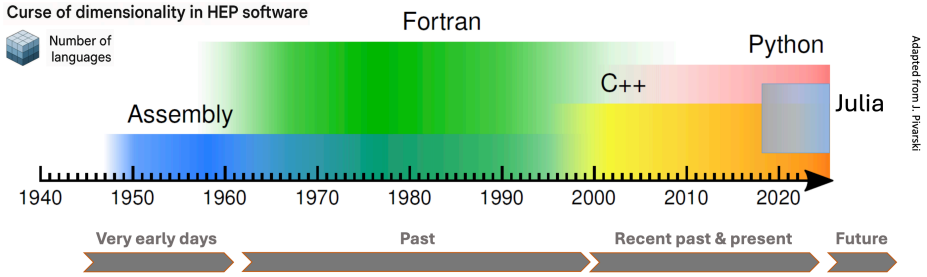


Figure 1. *HEP Software: it all revolves around a language, or several.*

JuliaHEP, with its emerging set of tools and libraries [1], aims to facilitate HEP data analysis in Julia. However, given the long timescales of HEP experiments and the substantial legacy software they depend on, introducing Julia alongside existing C++ and Python ecosystems inevitably creates a three-language problem — though this is intended as a transitional, rather than permanent, arrangement.

To navigate this, three main development strategies have emerged within the JuliaHEP community. The first involves wrapping mature C++ libraries to make them accessible from Julia — for example, Geant4.jl provides a Julia interface to the widely used Geant4 particle transportation toolkit, and ROOT.jl offers bindings to the ROOT data analysis framework. The second approach opts for complete reimplementations of existing tools in native Julia, as demonstrated by BAT.jl for Bayesian analysis and UnROOT.jl for reading ROOT files

Published Jul 10, 2025

Correspondence to
Ianna Osborne
ianna.osborne@cern.ch

Open Access

Copyright © 2025 Osborne & Ling. This is an open-access article distributed under the terms of the Creative Commons Attribution 4.0 International license, which enables reusers to distribute, remix, adapt, and build upon the material in any medium or format, so long as attribution is given to the creator.

without relying on the C++ libraries. The third strategy focuses on integrating Julia with Python; we are following this strategy to expose the Python-based Awkward Array library via its dedicated Julia backend, AwkwardArray.jl, enabling efficient access to nested, irregular data structures commonly used in HEP.

This backend addresses one of the key challenges in multi-language environments: data sharing across language boundaries. By exposing the underlying data buffers directly to Julia, AwkwardArray.jl avoids data copying and offers a gentle, incremental path for physicists to explore Julia's capabilities within their existing Python workflows.

2. Awkward Array in Julia

Awkward Array is a key library for handling complex and jagged HEP data in the Python ecosystem. To access this data from Julia, it is integrated via a dedicated backend, AwkwardArray.jl. What sets this backend apart is its tight integration with Julia's type system.

To bridge the Python and Julia ecosystems, we make extensive use of PythonCall.jl [2], a package that provides bi-directional interoperability between the two languages. This enables us to offload computationally heavy operations to Julia — following a pattern familiar from other JIT compilation tools in Python, such as Numba.

We evaluated this integration by comparing performance and usability using Julia-native types like Vector of Vectors, which provide similar capabilities for representing nested, irregular data. Our results show no significant overhead when combining the two ecosystems. Additionally, we explored Julia's interoperability and native packages to further optimize HEP computations, offering a practical pathway for gradually introducing Julia to the Python-based HEP community.

3. Performance Benefits

We present the benefits of using Julia for HEP workloads by benchmarking the same operation with JuliaHEP tools and comparing the results to Python and C++ implementations. As shown in Figure 2, Julia offers a noticeable speedup with less boilerplate.

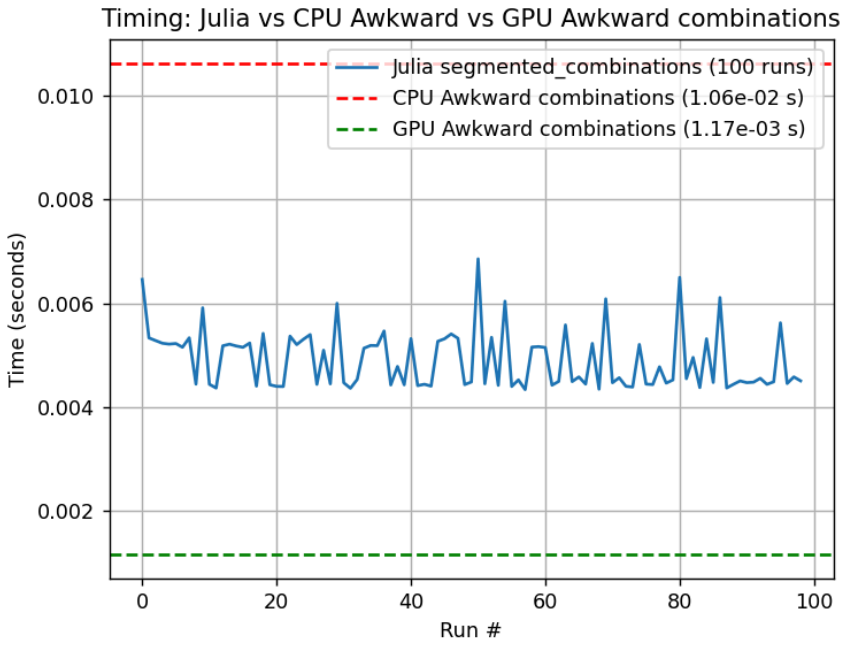


Figure 2. *This is the combinations performance plot, comparing the ak.combinations function with a Julia kernel on an awkward array. The Julia kernel uses JuliaMath/Combinatorics.jl [3].*

This is without yet exploiting Julia's potential for parallel computing and GPU support, which opens the door to workflows where data loaded with AwkwardArray.jl is processed using Julia-native GPU kernels or multi-threaded code — offering scalable performance without disrupting existing pipelines.

4. Scaling it up

While performance gains matter, integrating Julia into established HEP workflows also involves practical challenges. AwkwardArray.jl enables physicists to explore Julia's capabilities alongside existing Python tools by accessing shared data buffers directly, without copying. This allows heavy computations to be offloaded to Julia where appropriate, while maintaining full compatibility with Python's data ecosystem.

Naturally, introducing a third language raises long-term questions. Could Julia eventually take on a larger role, perhaps replacing parts of the Python or C++ stack? It remains too early to tell. Legacy software must be maintained, Python remains dominant, and new contenders like Rust are emerging. For now, the focus is on enabling a flexible, incremental path to using Julia where it offers clear advantages — without disrupting current workflows.

The main technical hurdle is efficient data interoperability — a challenge addressed by the Awkward Array design, which uses contiguous data buffers and descriptive metadata. This model has also proven effective in related optimizations, such as its virtual array implementation.

Data access remains central. ROOT TTrees are still the dominant format, with RNTuple expected to grow in use. Both formats are accessible as Awkward Arrays via uproot (Python) and UnROOT.jl (Julia), ensuring consistency across languages.

Managing a shared Python–Julia environment remains challenging. Conda for Python, juliaup, and Julia's package manager integrate well, though keeping package versions aligned requires care. A coordination tool like pixi is currently under evaluation to simplify this.

Overall, this approach offers a pragmatic, low-friction path for gradually introducing Julia into HEP workflows — letting physicists benefit from its strengths in JIT compilation, multiple dispatch, and GPU acceleration without leaving established ecosystems behind.

Acknowledgements

This work was supported by the National Science Foundation under Cooperative Agreement PHY-2323298.

References

[1] Graeme Andrew Stewart, "Julia in HEP," 2025, doi: https://doi.org/10.48550/arXiv.2503.08184.

[2] C. Rowley, "PythonCall.jl: Python and Julia in harmony." [Online]. Available: https://github.com/JuliaPy/PythonCall.jl

[3] "Combinatorics.jl: A combinatorics library for Julia." [Online]. Available: https://github.com/JuliaMath/Combinatorics.jl

Open Access | https://doi.org/10.25080/mfwm5796

SciPy 2025

July 7 - July 13, 2025

Proceedings of the 24th Python in Science Conference
ISSN: 2575-9752

CFSpy: A Python Library for the Computation of Chen-Fliess Series

Ivan Perez Avellaneda[1]

[1]Monaghan Medical Corp.

Abstract

In this paper, CFSpy a package in Python that numerically computes Chen-Fliess series is presented. The reachable sets of non-linear systems are also calculated with this package. The method used obtains batches of iterated integrals of the same length instead one of at a time. For this, we consider the alphabetical order of the words that index the series. By redefining the iterated integral reading the index word in the opposite direction from right to left, we allow the broadcasting of the computation of the iterated integral to all permutations of a length. Assuming the input is sufficiently well approximated by piecewise step functions in a fine partition of the time interval, the minimum bounding box of a reachable set is computed by means of polynomials in terms of the inputs. To solve the optimization problem, the SciPy library is used.

Keywords Chen-Fliess series, Nonlinear Control Systems, Python

1. Introduction

Control systems describe the dynamics of mechanisms driven by an input vector. In engineering, many of these systems are affected by disturbances or are complex enough that a simplified version of the model is used instead. This endows the systems with uncertainty and impairs their safety operations. The reachable set is a tool that helps analyse safety-related properties. For a given final time, it is defined as the set of all outputs as a response of the system to a given set of initial inputs and states. The overestimation of the reachable set is associated to safety and obstable avoidance and the underestimation to the *liveness* of the system [1], [2].

Different methodologies are used to compute the reachable set or an approximation of it. Among the most popular techniques are the Hamilton-Jacobi framework (HJ) [3], [4], [5] which uses game theory between two players where one player drives the system away from the goal while the other moves it towards the goal, contraction-based which uses contraction theory [6], [7] on the Jacobian of the vector field of the system, set-based that uses different set representations, monotone systems [8], [9], [10], and mixed-monotonicity [11]. There is also the simulation-based reachability [12], [13] and recently Chen-Fliess series (CFS) have been used for this purpose.

An important problem in the computation of the reachable set is the *curse of dimensionality*. This consists in the increasing of the complexity of the algorithm as the dimension increases. To tackle this in HJ-based methods, in C. He, Z. Gong, M. Chen, and S. Hebert [14], the authors use a new approach that requires the definition of an admissible control set to provide control policies that are consistent on the coupled subsystems. This fixes an incosistency issue with the controls called the *leaking corner*. For set-based methods, the use of zonotopes is known to have low complexity [15].

Published Jul 10, 2025

Correspondence to
Ivan Perez Avellaneda
iperezave@gmail.com

Open Access

Copyright © 2025 Avellaneda. This is an open-access article distributed under the terms of the Creative Commons Attribution 4.0 International license, which enables reusers to distribute, remix, adapt, and build upon the material in any medium or format, so long as attribution is given to the creator.

A CFS provides a local representation of the output of a non-linear control-affine system in terms of its input [16]. Given its coefficients, the series overlooks the dynamics when computing the output. This is important in cases where the system is unknown or affected by uncertainty. Then the coefficients are learned by using online learning methods [17]. In I. Perez Avellaneda and L. A. Duffaut Espinosa [18], a version of noncommutative differential calculus was developed to represent the derivative of a CFS. This was used to perform reachability analysis by applying second degree optimization of CFS in I. Perez Avellaneda and L. A. Duffaut Espinosa [19].

In the present work, CFSpy [20], a Python library for the computation of Chen-Fliess series is introduced and it is used to perform reachability analysis of non-linear system by assuming the input functions can be piece-wise approximated. A polynomial of the CFS is obtained and then optimized to get the points of an overstimation of the reachable set. The SciPy package [21] is used to performed the optimization.

The outline of the paper is the following: in Preliminaries, the preliminary concepts and results in reachability, formal languages and CFS are presented. In Main Results, the algorithms for the computation of the iterated integrals and the lie derivatives are provided which are the components of CFS. Then, the numerical computation of CFS is shown and examples are given. Finally, in Conclusions, the conclusions are stated.

2. Preliminaries

In the current section, we give the definitions and results needed to explain our main contribution: the CFSpy Python package. For this, in Section 1, tools from linear algebra and matrix operations are presented to explain the algorithms. In Section 2, concepts from formal language theory are presented to characterize CFS which are defined in Section 3. These provide a representation of the output of a nonlinear-affine systems in terms of iterated integrals of the input.

2.1. Matrix Operations

As we will see in Main Results, to avoid computing the components of the CFS one by one, this is, permutation by permutation, we can use matrix operations such as the Kronecker product to represent the stacking of two matrices and the Hadamard product for elementwise computation.

In particular, stacking a matrix B vertically k times is written in terms of this product and the unitary vector as follows

The next operation is useful to perfom the iterated integration of the inputs represented as matrices.

It will also be useful to define an operator for the vertical stacking of two different matrices.

Definition 1. *Kronecker Product*

Given the matrix $A_{m,n} = [a_{ij}]$ and $B_{p,q}$, the Kronecker product $A \otimes B$ is defined by

$$A \otimes B = \begin{bmatrix} a_{11}B & \cdots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \cdots & a_{mn}B \end{bmatrix} \tag{1}$$

Example 1.

Denote 1_k the vector of k ones and take the arbitrary matrix B, then

$$1_k \otimes B = \begin{bmatrix} B \\ \vdots \\ B \end{bmatrix} \tag{2}$$

Definition 2. *Hadamard Product*

Given the matrix $A_{m,n}$ and $B_{m,n}$, the Hadamard product is defined by

$$(A \odot B)_{ij} = (A)_{ij}(B)_{ij} \tag{3}$$

Example 2.

Consider the matrices $A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$ and $B = \begin{bmatrix} 7 & 8 & 9 \\ 10 & 11 & 12 \end{bmatrix}$. The Hadamard product $A \odot B$ is the following

$$A \odot B = \begin{bmatrix} 7 & 16 & 27 \\ 40 & 55 & 72 \end{bmatrix} \tag{4}$$

Definition 3. *Vertical Direct Sum*

Given the matrix $A_{m,n}$ and $B_{r,n}$, the vertical direct sum is defined by

$$A \oplus_v B = \begin{bmatrix} A \\ B \end{bmatrix} \tag{5}$$

2.2. Formal Language Theory

The CFS is indexed by words of any length. Formally, these words are elements of an algebraic structure called a *free monoid* where the *alphabet* is a subset that along with the *concatenation* operation act as the generator. Words are the noncommutative counterpart of monomials and they extend as the basis of polynomials and then to power series. These are important in the theory of CFS since power series are isomorphic to CFS. In the present section, concepts of formal language theory [22] are presented.

To make notation concise, given the monoid $(S, \cdot)$, we write $\cdot\,(s_1, s_2)$ as the concatenation $s_1 s_2$ then by the associativity $(s_1 s_2)s_3 = s_1(s_2 s_3) = s_1 s_2 s_3$. Thus, the operation $\cdot : S \times S \to S$ is

Example 3.

Consider the matrices $A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$ and $B = \begin{bmatrix} 7 & 8 & 9 \\ 10 & 11 & 12 \\ 13 & 14 & 15 \end{bmatrix}$. The vertical direct sum $A \oplus_v B$ is the following

$$A \oplus_v B = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \\ 13 & 14 & 15 \end{bmatrix} \tag{6}$$

Definition 4. *Monoid*

The tuple $(S, \cdot, e)$ is a *monoid* if the operation $\cdot : S \times S \to S$ satisfies the associativity property and e is the identity element under S.

Definition 5. *Length of a Word*

Given the free monoid $(X^*, \cdot, \emptyset)$ with alphabet $X = \{x_0, \cdots, x_m\}$, the *length* of a word is the function $|\cdot| : X^* \to \mathbb{N}$ such that for an arbitrary word $x_{i_1} \cdots x_{i_n} \in X^*$, it assigns

$$| x_{i_1} \cdots x_{i_n} | = n \tag{7}$$

Definition 6. *Formal Power Series*

A *formal power series* c with indeterminates in X^* and coefficients $(c, \eta) \in \mathbb{R}^\ell$ has the form:

$$c := \sum_{\eta \in X^*} (c, \eta)\eta \tag{8}$$

referred to as *concatenation*. The concatenation of a finite number of elements of S is called a *word*.

A *free monoid* $(X^*, \cdot, \emptyset)$ generated by the set X is the monoid of all finite concatenations of the elements of X. The generating set X is called *alphabet* and its elements *letters*. Next, we define a function that helps us classify words and define the algorithms in Main Results.

The set of all words of length k is written as X^k. Then, we can express $X^* = \cup_{k=0}^{\infty} X^k$. Next, we define the set of formal power series over $\mathbb{R}$ with indeterminates in X^*.

The set of all power series c is denoted $\mathbb{R}^\ell \langle\langle X \rangle\rangle$.

2.3. Chen-Fliess Series

CFS has its roots in the works of [23] and [16] and provide an input-output representation of nonlinear control-affine system. They are defined in terms of iterated integrals. To identify them with a system, the coefficients are written in terms of Lie derivatives of the vector field. In the current section, The CFS are presented.

We refer to the following set of equations as a nonlinear control-affine system:

$$\begin{aligned} \dot{z} &= g_0(z) + \sum_{i=1}^{m} g_i(z) u_i \\ y &= h(z) \end{aligned} \tag{9}$$

where $z = (z_1, \cdots, z_n)$ is the vector state of the system, $u = (u_1, \cdots, u_m)$ is the control input vector of the system and y is the output.

The iterated integral associated with the word index $\xi = x_{i_1}, \cdots, x_{i_r}$ maps each letter of an alphabet with a coordinate of the vector input function and integrates recursively each input coordinate in the order of the letters in the given word. Specifically, we have the following:

The definition of the iterated integral associated to a word is naturally extended to power series in the following manner:

Definition 7.

Given the free monoid $(X^*, \cdot, \emptyset)$, and the word $\xi = x_i\eta \in X^*$, the *iterated integral* of $u \in L^m[0,T]$ associated to ξ is the operator $E_\xi : L^m[0,T] \to C[0,T]$, described recursively by

$$E_\emptyset[u](t) = 1 \tag{10}$$

$$E_{x_i\eta}[u](t) = \int_0^t u_{x_i}(\tau) E_\eta[u](\tau) d\tau \tag{11}$$

Definition 8.

Given the free monoid $(X^*, \cdot, \emptyset)$, and the formal power series $c \in \mathbb{R}^\ell\langle\langle X\rangle\rangle$, the *Chen-Fliess series* associated to c is the functional, $F_c[\cdot](t) : L^p[0,T] \to \mathbb{R}^\ell$, described by

$$F_c[u](t) = \sum_{\eta \in X^*} (c,\eta) E_\eta[u](t) \tag{12}$$

The support $\sup(c)$ of a formal power series $c = \sum_{\eta\in X^*}(c,\eta)\eta$ is the set of words $\eta \in X^*$ that have non-null associated coefficients. This is, $(c,\eta) \neq 0$. Computationally, we work with power series of a finite support. Specifically, CFS truncated to words of length N denoted as $F_c^N[u](t)$. This is, the formal power series c of the CFS is truncated to words of length N and we have

$$F_c^N[u](t) = \sum_{k\leq N} \sum_{\eta \in X^k} (c,\eta) E_\eta[u](t). \tag{13}$$

Next, to provide the association of the CFS with a nonlinear control-affine system, we give the defintion of a Lie derivative.

The following result gives the conditions under which a nonlinear system has an input-output representation by CFS.

3. Main Results

In the current section, the numerical computation of the Chen-Fliess series is addressed. For this, the iterated integral and the Lie derivative are reformulated to allow for easier algorithm implementations. In Section 3, they are written recursively. In this section, we

Definition 9.

Given the system in (9), the Lie derivative associated with the word $\eta \in X^*$ of the function $h : \mathbb{R}^n \to \mathbb{R}$ is the following:

$$L_\eta h = L_{x_{i_1}} \cdots L_{x_{i_k}} h \tag{14}$$

where we have that $L_{x_j} h = \left(\frac{\partial}{\partial x} h\right) \cdot g_j$

Theorem 1. *Fliess, 1983*

Consider the system in (9). The Chen-Fliess series F_c represents the system if and only if

- The coefficient of the Fliess operator satisfies: $(c,\eta) = L_\eta h(z) \mid_{z_0}$ for $\eta \in X^*$.
- The power series c has finite Lie rank.
- There exist $K, M \geq 0$ such that $\mid (c,\eta) \mid \leq KM^{\mid \eta \mid} \mid \eta \mid !$

changed the direction of the expressions to compute them iteratively. Instead of looking at the definition of the components forwardly, they are rewritten backwardly.

Another aspect of the numerical computation that we describe is how to avoid calculating each component of the Chen-Fliess series word by word. For this, the new definitions are broadcasted to generate batches of a fixed length that will be stacked and multiplied componentwise to obtain the final output. This increases the speed of the algorithms.

3.1. Broadcasting

Consider the partition $\mathcal{P} = \{t_0, \cdots, t_R\}$ of the time interval $[0, T]$ and the input vector function $u = (u_0, \cdots, u_m) \in L_p^{m+1}[0, T]$. To avoid computing each iterated integral one word at the time and, instead, obtain the batch of all words of a fixed length, we need to consider the alphabetical order of the words. First, broadcast the integration of the input function

$$\int_0^{\tau_1} u(\tau)d\tau = \left(\int_0^{\tau_1} u_0(\tau)d\tau, \cdots, \int_0^{\tau_1} u_m(\tau)d\tau\right), \tag{15}$$

then broadcast the multiplication by the first component u_0 and integrate, this is,

$$\begin{aligned}\int_0^{\tau_2} u_0(\tau_1) \int_0^{\tau_1} u(\tau)d\tau d\tau_1 = \\ \left(\int_0^{\tau_2} u_0(\tau_1) \int_0^{\tau_1} u_0(\tau)d\tau d\tau_1, \cdots, \int_0^{\tau_2} u_0(\tau_1) \int_0^{\tau_1} u_m(\tau)d\tau d\tau_1\right).\end{aligned} \tag{16}$$

To make notation lighter, we skip the limits of the integrals and their differentials. Repeating the integration process for each of the components and stacking the result, we obtain the array

$$\int u \int u = \left[\int u_i \int u_0 \cdots \int u_i \int u_m\right]_{i\in\{1,\cdots,m\}}. \tag{17}$$

Notice that $\int u \int u$ is the array of iterated integral $E_\eta[u]$ of words η of length two. Inductively, given the array of iterated integrals of all words in X^k and denote it

$$\left(\int u\right)^k \tag{18}$$

we construct the array of iterated integral of words in X^{k+1} by broadcasting the multiplication of each component u_i and the integration. This is,

$$\left(\int u\right)^{k+1} = \left[\int u_i \odot \left(\int u\right)^k\right]_{i\in\{1,\cdots,m\}} \tag{19}$$

Numerically, this is done as follows. Take the matrix of the partitioned input functions

$$\begin{bmatrix} u_0(t_0) & \cdots & u_0(t_R) \\ \vdots & \ddots & \vdots \\ u_m(t_0) & \cdots & u_m(t_R) \end{bmatrix} \tag{20}$$

integrate by taking the cummulative sum of each component of the matrix and multiplying by a given rectangle size Δ

$$\begin{bmatrix} \sum_{i=-1}^{0} u_0(t_i) & \cdots & \sum_{i=-1}^{R} u_0(t_i) \\ \vdots & \ddots & \vdots \\ \sum_{i=-1}^{0} u_m(t_i) & \cdots & \sum_{i=-1}^{R} u_m(t_i) \end{bmatrix} \Delta \tag{21}$$

broadcast the multiplication of the first component u_0

$$\begin{bmatrix} u_0(t_0) & \cdots & u_0(t_R) \\ \vdots & \ddots & \vdots \\ u_0(t_0) & \cdots & u_0(t_R) \end{bmatrix}_{(m,\, T+1)} \odot \begin{bmatrix} \sum_{i=-1}^{0} u_0(t_i) & \cdots & \sum_{i=-1}^{R} u_0(t_i) \\ \vdots & \ddots & \vdots \\ \sum_{i=-1}^{0} u_m(t_i) & \cdots & \sum_{i=-1}^{R} u_m(t_i) \end{bmatrix} \Delta \tag{22}$$

repeat for each component of the input and stack them as in equation (17) to obtain the list of $E_\eta[u]$ for $\eta \in X^2$. This is,

$$\begin{bmatrix} \sum_{k=-1}^{0} u_0(t_k)(\sum_{i=-1}^{k} u_0(t_i))\Delta & \cdots & \sum_{k=-1}^{R} u_0(t_k)(\sum_{i=-1}^{k} u_0(t_i))\Delta \\ \vdots & \ddots & \vdots \\ \sum_{k=-1}^{0} u_0(t_k)(\sum_{i=-1}^{k} u_m(t_i))\Delta & \cdots & \sum_{k=-1}^{R} u_0(t_k)(\sum_{i=-1}^{k} u_m(t_i))\Delta \\ & \vdots & \\ \sum_{k=-1}^{0} u_m(t_k)(\sum_{i=-1}^{k} u_0(t_i))\Delta & \cdots & \sum_{k=-1}^{R} u_m(t_k)(\sum_{i=-1}^{k} u_0(t_i))\Delta \\ \vdots & \ddots & \vdots \\ \sum_{k=-1}^{0} u_m(t_k)(\sum_{i=-1}^{k} u_m(t_i))\Delta & \cdots & \sum_{k=-1}^{R} u_m(t_k)(\sum_{i=-1}^{k} u_m(t_i))\Delta \end{bmatrix} \Delta. \tag{23}$$

This way of computing the iterated integral assumes and takes advantage of the lexicographical order of words. This avoids having to compute the same iterated integrals several times and instead uses the current block of iterated integrals of certain size N to compute the whole batch of the iterated integrals of size $N+1$. The algorithm is given in the next section.

3.2. Computation of Iterated Integrals

The following provides a different view of Definition 7 that we use later to compute the iterated integrals. Following the notation of the previous section, we have

Algorithm 1 provides the matrix of the stacked iterated integrals. This is the list of iterated integrals associated with words of length less than a certain number N. We denote $S(\cdot)$ the cummulative sum as in equation (21) and use a negative number in the index $\{:, -j\}$ to indicate the removal of column j of the corresponding matrix.

The Python code of Algorithm 1 is the following:

Definition 10.

Consider the word $\eta = x_{i_1} \cdots x_{i_r} \in X^*$, the *backward* iterated integral of $u \in L^p[0,T]$ associated with η is the operator $H_\eta(\cdot)$ described recursively by

$$H_{x_{i_1}}\left(H_{x_{i_2}}\left(\cdots H_{x_{i_r}}\right)\right) \tag{24}$$

where

$$H_{x_{i_j}}(\cdot) = \tag{25}$$

Algorithm 1.

Inputs Given the truncation length N, the inputs u of the system in (9)

Output The matrix $\mathcal{U}$ of the stacking of iterated integrals $E_\eta[u](t)$ for $|\eta| \leq N$

$U_0 \leftarrow 1 \oplus_v u$

$U_1 \leftarrow [\mathbf{0} \mid \ S(U_0)\Delta]$

$\mathcal{U} \leftarrow U_1$

For k in $\{1, \cdots, N-1\}$ do:

1. $V \leftarrow \mathbf{1}_m \otimes U_k$
2. $v \leftarrow \left[I_m \otimes \mathbf{1}_{N_{U_k}}\right] u$
3. $M \leftarrow v \odot V$
4. $U_{k+1} \leftarrow \left[\mathbf{0} \mid \ S(M)_{:,-\hat{T}}\Delta\right]$
5. $\mathcal{U} \leftarrow U_k \oplus_v \mathcal{U}$

```
def iter_int(u,t0, tf, dt, Ntrunc):
    import numpy as np

    # The NumPy package is used to handle matrix operations.

    """
    Returns the list of all iterated integrals of the input u
    indexed by the words of length from 1 to Ntrunc.


    Parameters:
    -----------
    u: array_like
        The array of input functions u_i: [t0, tf] -> IR, for all i in {1, ..., m}
        stacked vertically. Each function has the form u_i = np.array([u_i[0], u_i[1], ...,
u_i[N]])
        where u_i[0] = u_i(t_0), u_i[1] = u_i(t_0+dt), ..., u_i[N] = u_i(tf), N = int((tf-t0)//dt+1)
        and u = np.vstack([u_1, u_2, ..., u_m])

    t0: float
        Initial point of the time-interval domain of the inputs

    tf: float
        Final time of the time-interval domain of the inputs

    dt: float
        The size of the step of the evenly spaced partition of the time-interval domain

    Ntrunc: int
        The truncation length of the words of the Chen-Fliess series
        Sum_{i=0}^Ntrunc Sum_{eta in X^i} (c, eta) E_{eta}[u](t0,tf)



    Returns:
    --------
    list: ndarray
    """
```

Program 1. *Python function to compute iterated integrals.*

```
    # The input function u which is the first parameter
    # is the vertical stacking of the coordinate input functions u_i
    # where each u_i is a zero dimensional numpy array of values defined for
    # each point of the dicretized time interval t = np.linspace(t0, tf, int((tf-t0)//dt+1)))
    # then the length of the partition of time is computed.
    length_t = int((tf-t0)//dt+1)

    # Safety check to see whether the length of u is equal to int((tf-t0)//dt+1))
    if u.shape[1] != length_t:
        raise ValueError("The length of the input, %s, must be int((tf-t0)//dt+1) = %s."
%(u.shape[1], length_t))

    # Generate the discretized time interval
    t = np.linspace(t0, tf, length_t)

    # The input u_0 associated with the letter x_0 is generated.
    u0 = np.ones(length_t)

    # The inputs of the system associated with x_1, ..., x_m with the input associated to x_0
vertically are stacked.
    # [
    #  [u_0(t0), u_0(t0+dt), ..., u_0(tf)],
    #  [u_1(t0), u_1(t0+dt), ..., u_1(tf)],
    #    .
    #    .
    #    .
    #  [u_m(t0), u_m(t0+dt), ..., u_m(tf)]
    # ]
    u = np.vstack([u0, u])
    # The number of rows which are equal to the number of total input functions is obtained.
    num_input = int(np.size(u,0))

    # The total number of iterated integrals of word length less than or equal to the truncation
length is computed.
    # total_iterint = num_input + num_input**2 + ... + num_input**Ntrunc
    total_iterint = num_input*(1-pow(num_input,Ntrunc))/(1-num_input)

    # This is transformed into an integer.
    total_iterint = int(total_iterint)

    # A matrix of zeros with as many rows as the total number of iterated integrals and as many
columns
    # as the elements in the partition of time is computed.
    Etemp = np.zeros((total_iterint,length_t))

    # Starts the list ctrEtemp such that ctrEtemp[k]-ctrEtemp[k-1] = the number of iterated
integrals
    # of word length k
    # ctrEtemp[0] = 0
    # ctrEtemp[k] = num_input + num_input**2 + ... + num_input**k,  1<=k<=Ntrunc
    ctrEtemp = np.zeros(Ntrunc+1)

    # ctrEtemp[k] = num_input + num_input**2 + ... + num_input**k,  1<=k<=Ntrunc
    for i in range(Ntrunc):
        ctrEtemp[i+1] = ctrEtemp[i]+pow(num_input,i+1)

    # The iterated integrals of the words of length 1, E_{x_i}[u](t0, tf) for all i in {0, ..., m},
are computed.

    # First, E_{x_i}[u](t0, tf) for all i in {0, ..., m} are computed
    # for all tf neq t0
    sum_acc = np.cumsum(u, axis = 1)*dt

    # Then the values of E_{x_i}[u](t0, tf) for tf = t0, this is E_{x_i}[u](t0, t0) = 0, are added
    # to have E_{x_i}[u](t0, tf) for all tf>=t0
    Etemp[:num_input,:] = np.hstack((np.zeros((num_input,1)), sum_acc[:,:-1]))

    # The iterated integrals of the words of length k => 1, E_{x_{i_1}...x_{i_k}}[u](t0, tf) for all
i_j in {0, ..., m},
    # are computed at each iteration.
    for i in range(1,Ntrunc):
        # start_prev_block = num_input + num_input**2 + ... + num_input**(i-1)
        start_prev_block = int(ctrEtemp[i-1])
        # end_prev_block = num_input + num_input**2 + ... + num_input**i
```

```
        end_prev_block = int(ctrEtemp[i])
        # end_current_block = num_input + num_input**2 + ... + num_input**(i+1)
        end_current_block = int(ctrEtemp[i+1])
        # num_prev_block = num_input**i
        num_prev_block = end_prev_block - start_prev_block
        # num_current_block = num_input**(i+1)
        num_current_block = end_current_block - end_prev_block


        """
        U_block =
        [
         u_0
         u_0
          .
          .   # u_0 repeats num_input**i times
          .
         u_0
         u_1
         u_1
          .
          .   # u_1 repeats num_input**i times
          .
         u_1
          .
          .   # u_k repeats num_input**i times
          .
         u_m
         u_m
          .
          .   # u_m repeats num_input**i times
          .
         u_m
        ]
        """

        U_block = u[np.repeat(range(num_input), num_prev_block), :]

        """
        prev_int_block =
        [
        E_{x_0...x_0x_0}[u](t0,tf)
        E_{x_0...x_1x_0}[u](t0,tf)
                .
                .                # block with all the num_input**i iterated integrals of words of
length i
                .
        E_{x_m...x_mx_m}[u](t0,tf)
        -------------------------
        E_{x_0...x_0x_0}[u](t0,tf)
        E_{x_0...x_1x_0}[u](t0,tf)
                .
                .                # block with all the num_input**i iterated integrals of words of
length i
                .
        E_{x_m...x_mx_m}[u](t0,tf)
        -------------------------
                .
                .
                .
        -------------------------
        E_{x_0...x_0x_0}[u](t0,tf)
        E_{x_0...x_1x_0}[u](t0,tf)
                .
                .                # block with all the num_input**i iterated integrals of words of
length i
                .
        E_{x_m...x_mx_m}[u](t0,tf)
        ]

        In total there are num_input blocks
        """
        prev_int_block = np.tile(Etemp[start_prev_block:end_prev_block,:],(num_input,1))
```

```
        """
        U_block*prev_int_block =
        [
        u_0(tf)E_{x_0...x_0x_0}[u](t0,tf)
        u_0(tf)E_{x_0...x_1x_0}[u](t0,tf)
                .
                .                       # block with all the num_input**i iterated integrals of
words of length i
                .
        u_0(tf)E_{x_m...x_mx_m}[u](t0,tf)
        -------------------------
        u_1(tf)E_{x_0...x_0x_0}[u](t0,tf)
        u_1(tf)E_{x_0...x_1x_0}[u](t0,tf)
                .
                .                       # block with all the num_input**i iterated integrals of
words of length i
                .
        u_1(tf)E_{x_m...x_mx_m}[u](t0,tf)
        -------------------------
                .
                .
                .
        -------------------------
        u_m(tf)E_{x_0...x_0x_0}[u](t0,tf)
        u_m(tf)E_{x_0...x_1x_0}[u](t0,tf)
                .
                .                       # block with all the num_input**i iterated integrals of
words of length i
                .
        u_m(tf)E_{x_m...x_mx_m}[u](t0,tf)
        ]

        current_int_block integrates U_block*prev_int_block
        """
        current_int_block = np.cumsum(U_block*prev_int_block, axis = 1)*dt
        # Stacks the block of iterated integrals of word length i+1 into Etemp
        Etemp[end_prev_block:end_current_block,:] = np.hstack((np.zeros((num_current_block,1)),
current_int_block[:,:-1]))

    itint = Etemp
    return itint
```

3.3. Computation of Lie Derivatives

As in the previous section, we take advantage of the lexicographical order of words. Similarly to the stacked list of iterated integrals, Algorithm 2 provides the list of the stacked Lie derivatives:

The Python code of Algorithm 2 is the following:

Algorithm 2.

Inputs Given the truncation length N, the inputs u of the system in (9)

Output The matrix $\mathcal{G}$ of the stacking of Lie derivatives $L_\eta h(x)$ for $\mid \eta \mid \leq N$

$G_0 \leftarrow \mathbf{1}_m \otimes h$

$G_1 \leftarrow \frac{\partial}{\partial x} G_0 \cdot g$

$\mathcal{G} \leftarrow G_1$

For k in $\{1, \cdots, N-1\}$ do:

1. $V \leftarrow \mathbf{1}_m \otimes \frac{\partial}{\partial x} G_k$
2. $v \leftarrow \left[I_m \otimes \mathbf{1}_{N_{G_k}} \right] g$
3. $M \leftarrow V \cdot v$
4. $G_{k+1} \leftarrow M$
5. $\mathcal{G} \leftarrow G_{k+1} \oplus_v \mathcal{G}$

```
def iter_lie(h,vector_field,z,Ntrunc):
    import numpy as np
    import sympy as sp

    # The NumPy package is used to handle matrix operations and
    # SymPy to compute the partial derivatives of the Lie derivative.

    """
    Returns the list of all the Lie derivatives indexed by the words of length from 1 to Ntrunc
    Given the system
        dot{z} = g_0(z) + sum_{i=1}^m g_i(z) u_i(t),
        y = h(z)
    with g_i: S -> IR^n and h: S -> IR, S is a subset of IR^n for all i in {0, ..., n}
    The Lie derivative L_eta h of the output function h(z) indexed by the word
    eta = x_{i_1}x_{i_2}...x_{i_k} is defined recursively as
        L_eta h = L_(x_{i_2}...x_{i_k}) (partial/partial z h) cdot g_{i_1}

    Parameters:
    -----------
    h: symbolic
        The symbolic function that represents the output of the system.

    vector_field: symbolic array
        The array that contains the vector fields of the system.
        vector_field =
            sp.transpose(sp.Matrix([[(g_0)_1, ..., (g_0)_n],[(g_1)_1, ..., (g_1)_n], ...,
[(g_m)_1, ..., (g_m)_n]]))

    z: symbolic array
        The domain of the vector fields.
        z = sp.Matrix([z1, z2, ...., zn])

    Ntrunc: int
        The truncation length of the words index of the Lie derivatives

    Returns:
    --------
    list: symbolic array
    """
```

Program 2. *Python function to compute Lie derivatives.*

```
     # The number of vector fields is obtained.
    # num_vfield = m
    num_vfield = np.size(vector_field,1)


    # The total number of Lie derivatives of word length less than or equal to the truncation length
is computed.
    # total_lderiv = num_input + num_input**2 + ... + num_input**Ntrunc
    total_lderiv = num_vfield*(1-pow(num_vfield, Ntrunc))/(1-num_vfield)
    total_lderiv = int(total_lderiv)


    # The list that will contain all the Lie derivatives is initiated.
    Ltemp = sp.Matrix(np.zeros((total_lderiv, 1), dtype='object'))
    ctrLtemp = np.zeros((Ntrunc+1,1), dtype = 'int')

    # ctrLtemp[k] = num_input + num_input**2 + ... + num_input**k,  1<=k<=Ntrunc
    for i in range(Ntrunc):
        ctrLtemp[i+1] = ctrLtemp[i] + num_vfield**(i+1)

    # The Lie derivative L_eta h(z) of words eta of length 1 are computed
    LT = sp.Matrix([h]).jacobian(z)*vector_field

    # Transforms the lie derivative from a row vector to a column vector
    LT = LT.reshape(LT.shape[0]*LT.shape[1], 1)

    # Adds the computed Lie derivatives to a repository
    Ltemp[:num_vfield, 0] = LT


    # The Lie derivatives of the words of length k => 1, L_{x_{i_1}...x_{i_k}}h(z) for all i_j in
{0, ..., m},
    # are computed at each iteration.

    for i in range(1, Ntrunc):
        # start_prev_block = num_input + num_input**2 + ... + num_input**(i-1)
        start_prev_block = int(ctrLtemp[i-1])
        # end_prev_block = num_input + num_input**2 + ... + num_input**i
        end_prev_block = int(ctrLtemp[i])
        # end_current_block = num_input + num_input**2 + ... + num_input**(i+1)
        end_current_block = int(ctrLtemp[i+1])
        # num_prev_block = num_input**i
        num_prev_block = end_prev_block - start_prev_block
        # num_current_block = num_input**(i+1)
        num_current_block = end_current_block - end_prev_block

        """
        LT =
        [
         [L_{x_0...x_0x_0}h(z)],
         [L_{x_0...x_1x_0}h(z)],
                .
                .
                .
         [L_{x_m...x_m}h(z)],
        ]
        these are the Lie derivatives indexed by words of length i
        """
        LT = Ltemp[start_prev_block:end_prev_block,0]

        """
        LT =
        [
         [partial/ partial z L_{x_0...x_0x_0}h(z)],
         [partial/ partial z L_{x_0...x_1x_0}h(z)],
                .
                .
                .
         [partial/ partial z L_{x_m...x_m}h(z)],
        ]
        *
        [g_0, g_1, ..., g_m]

        =
```

```
        [
            L_{x_0x_0...x_0}h(z)    |   L_{x_0x_0..x_0x_1x_0}h(z)   |          |  L_{x_0x_m..x_mx_m}
h(z)
            L_{x_1x_0...x_0}h(z)    |   L_{x_1x_0..x_0x_1x_0}h(z)   |          |  L_{x_1x_m..x_mx_m}
h(z)
            L_{x_2x_0...x_0}h(z)    |   L_{x_2x_0..x_0x_1x_0}h(z)   |          |  L_{x_2x_m..x_mx_m}
h(z)
                .                   |       .                       |    ...   |       .
                .                   |       .                       |    ...   |       .
                .                   |       .                       |    ...   |       .
            L_{x_mx_0...x_0}h(z)    |   L_{x_mx_0..x_0x_1x_0}h(z)   |          |  L_{x_mx_m..x_mx_m}
h(z)
        ]

        """
        LT = LT.jacobian(z)*vector_field
        # Transforms the lie derivative from a row vector to a column vector

        """
        LT =
        [
            L_{x_0x_0...x_0}h(z)
            L_{x_1x_0...x_0}h(z)
            L_{x_2x_0...x_0}h(z)
                .
                .
                .
            L_{x_mx_0...x_0}h(z)
            ------------------------
            L_{x_0x_0...x_1x_0}h(z)
            L_{x_1x_0...x_1x_0}h(z)
            L_{x_2x_0...x_1x_0}h(z)
                .
                .
                .
            L_{x_mx_0...x_1x_0}h(z)
            ------------------------
                .
                .
                .
            ------------------------
            L_{x_0x_m...x_m}h(z)
            L_{x_1x_m...x_m}h(z)
            L_{x_2x_m...x_m}h(z)
                .
                .
                .
            L_{x_mx_m...x_m}h(z)
        ]
        these are the Lie derivatives indexed by words of length i+1
        """
        LT = LT.reshape(LT.shape[0]*LT.shape[1], 1)
        # Adds the computed Lie derivatives to the repository
        Ltemp[end_prev_block:end_current_block,:]=LT

    return Ltemp
```

Notice that the definition of the Lie derivative in Definition 9 does not require a modification since its computation is performed backwards from the last letter of the flipped word to the first letter.

3.4. Numerical Computation of Chen-Fliess Series

In section Section 5, we obtained the iterated integral and in section Section 6, the Lie derivative. Alternatively to Definition 8, the CFS is represented as the inner product of the vector with coordinates equal to all the iterated integrals and the vector with coordinates equal to all the Lie derivatives. This is,

Theorem 2.

The truncated CFS is approximated using the output of Program 1, $\mathcal{U}$, and Program 2, $\mathcal{G}$, as follows

$$F_c^N[u](t) \approx \mathcal{U} \cdot \mathcal{G} \tag{27}$$

$$F_c[u](t) = \left(\sum_{\eta \in X^\star} (c, \eta)\eta\right) \cdot \left(\sum_{\eta \in X^\star} E_\eta[u](t)\eta\right). \tag{26}$$

We obtain a similar version for the truncated CFS. From this and the previous sections, we have the following approximation of the CFS truncated to word length N:

3.5. CFSpy Package

The CFSpy package is a set of tools implemented in the Python programming language to compute Chen-Fliess series and perform reachability analysis of nonlinear control-affine systems. The package is intended to be as *minimal* and *self-contained* as possible as it doesn't require any other package or software except for NumPy [24] and SymPy [25] used in the computation of the series and for the reachability.

The package consists of the following elements in terms of the system (9):

- The function *iter_int* is the implementation of Algorithm 1 and takes as arguments the discretized input function $u : [t_0, t_f] \to \mathbb{R}^m$, the initial and final time t_0, t_f, the step size dt of discretization of the input function and the truncation length N of the series. The input u is entered as a $m \times N_T$ matrix where the $i-th$ row is the $i-th$ input coordinate u_i and the length of the dicretization if N_T. The output of the function is list of iterated integrals of length less or equal to N. The function *single_iter_int* provides one single iterated integral given the index word η as argument.
- The function *iter_lie* is the implementation of Algorithm 2 and takes as arguments the output function $h : \mathbb{R}^n \to \mathbb{R}$, the vector fields $g_i(z)$, the state variable z, and the truncation length N of the series. The functions are entered as symbolic functions.
- The function *pol_inputs* is the symbolic version of iter_int that assumes the input functions are constants. It takes a set of symbolic inputs and the truncation length N of the series. The output is the list of symbolic commutative monomials of degree less or equal to N.

4. Simulations

In the current section, we use the CFSpy package to simulate the output of a system and analyze the reachability by computing the minimum bounding overestimating box of the set of outputs given a set of inputs to the system.

Consider the Susceptible-Exposed-Infectious-Recovered-Susceptible (SEIRS) model for infectious disease dynamics [26]

$$\begin{aligned} \dot{S} &= \mu M - \beta I S/M + \omega R - \mu S \\ \dot{E} &= \beta I S/M - \sigma E - \mu E \\ \dot{I} &= \sigma E - \gamma I - (\mu + \alpha) I \\ \dot{R} &= \gamma I - \omega R - \mu R \end{aligned} \tag{28}$$

where S represents the susceptible population, E, the exposed group, I, the infectious population, and R the recovered population. The parameter β represents the average rate

at which an infected individual can infect a susceptible one, $1/\sigma$ is the period an exposed individual stays in that group before becoming infectious. Recovered individuals are immune for an average protected period of $1/\omega$. The parameter α is the rate at which the infected population dies and μ is the rate of background death of all the population. The total population in the model is M and $1/\gamma$ is the infectious period.

Take the SEIRS model in (28) with control input β and γ and the following parameters: $\mu = 1/76, \omega = 1, \sigma = 1/7, \alpha = 0, M = 1.0$. The system (28) is written as in (9) where $y = S$. Consider $\beta = \sin(t)$ and $\gamma = \cos(t)$, and word truncation length of $N = 6$.

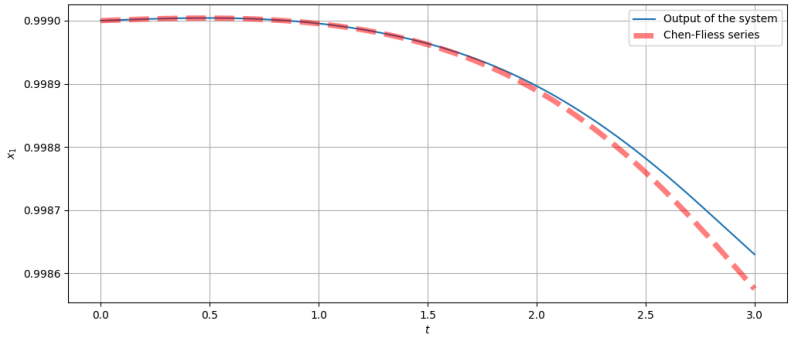


Figure 1. *The picture shows the susceptible population of the SEIRS model and its approximation by CFS when the fluctuating inputs $\gamma = \cos(t)$ and $\beta = \sin(t)$ enter the system. The CFS is truncated to a word length of $N = 6$. The approximation by CFS performs well before $t = 1.5s$ when the series starts to diverge.*

The following block of code uses CFSpy to compute the CFS of the SEIRS system having the susceptible population as the output.

```
from CFS import iter_int, iter_lie, single_iter_int, single_iter_lie

import numpy as np
from scipy.integrate import solve_ivp
import matplotlib.pyplot as plt
import sympy as sp

mu = 1 / 76
omega = 1
sigma = 1 / 7
alpha = 0
N = 1.0

# Define the SEIRS system
def system(t, x, u1_func, u2_func):
    x1, x2, x3, x4 = x
    u1 = u1_func(t)
    u2 = u2_func(t)
    dx1 = mu * N - u1 * x3 * x1 / N + omega * x4 - mu * x1
    dx2 = u1 * x3 * x1 / N - sigma * x2 - mu * x2
    dx3 = sigma * x2 - u2 * x3 - (mu + alpha) * x3
    dx4 = u2 * x3 - omega * x4 - mu * x4
    return [dx1, dx2, dx3, dx4]

# Input 1
def u1_func(t):
    return np.sin(t)

# Input 2
def u2_func(t):
    return np.cos(t)
```

Program 3. *CFS Computation*

```
# Initial condition
x0 = [0.999, 0.001, 0.0, 0.0]

# Time range
t0 = 0
tf = 3
dt = 0.001
t_span = (t0, tf)

# Simulation of the system
solution = solve_ivp(system, t_span, x0, args=(u1_func, u2_func), dense_output=True)

# Partition of the time interval
t = np.linspace(t_span[0], t_span[1], int((tf-t0)//dt+1))
y = solution.sol(t)

# Define the symbolic variables
x1, x2, x3, x4 = sp.symbols('x1 x2 x3 x4')
x = sp.Matrix([x1, x2, x3, x4])

# Define the system symbolically
g = sp.transpose(sp.Matrix([[mu * N + omega * x4 - mu * x1, - sigma * x2 - mu * x2, sigma * x2 - (mu
+ alpha) * x3, - omega * x4 - mu * x4], \
                            [-x3 * x1 / N, x3 * x1 / N, 0, 0], [0, 0, - x3, x3]]))

# Define the output symbolically
h = x1

# The truncation of the length of the words that index the Chen-Fliess series
Ntrunc = 6

# Coefficients of the Chen-Fliess series evaluated at the initial state
Ceta = np.array(iter_lie(h,g,x,Ntrunc).subs([(x[0], x0[0]),(x[1], x0[1]), (x[2], x0[2]), (x[3],
x0[3])]))

# inputs as arrays
u1 = np.sin(t)
u2 = np.cos(t)

# input array
u = np.vstack([u1, u2])

# List of iterated integral
Eu = iter_int(u,t0, tf, dt, Ntrunc)

# Chen-Fliess series
F_cu = x0[0]+np.sum(Ceta*Eu, axis = 0)

# Graph of the output and the Chen-Fliess series
plt.figure(figsize = (12,5))
plt.plot(t, y[0].T)
plt.plot(t, F_cu, color='red', linewidth=5, linestyle = '--', alpha = 0.5)
plt.xlabel('$t$')
plt.ylabel('$x_1$')
plt.legend(['Output of the system','Chen-Fliess series'])
plt.grid()
plt.show()
```

Consider the SEIRS model (28), with control inputs $\beta \in [0.01,\ 0.7]$ and $\gamma \in [0.03,\ 3]$. The following code uses SciPy Optimize to compute the reachable set of the susceptible population as the convex line between its minimum and maximum at time $t = 1s$.

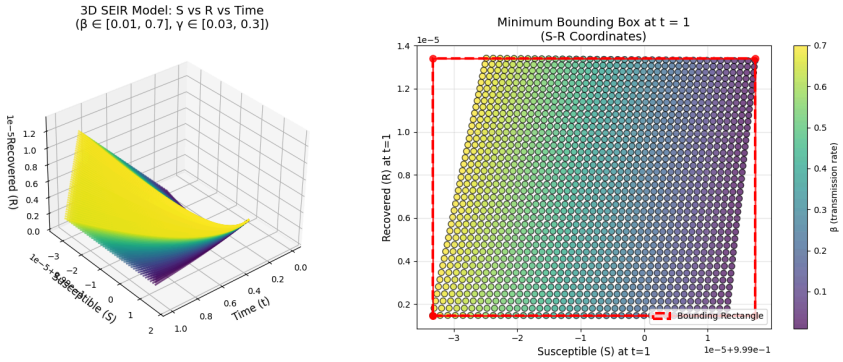


Figure 2. *The picture shows the MBB of the reachable set of system (28) to inputs* $\beta \in [0.01,\ 0.7]$ *and* $\gamma \in [0.03,\ 3]$ *for* $t = 1s$*. The CFS is truncated to a word length of* $N = 6$*.*

```
import numpy as np
import sympy as sp
from CFS import iter_lie, sym_CFS
from scipy.optimize import minimize
from scipy.optimize import Bounds

# Parameters of the model
mu = 1 / 76
omega = 1
sigma = 1 / 7
alpha = 0
N = 1.0

# Initial state
x0 = [0.999, 0.001, 0.0, 0.0]

# Define the symbolic variables
x1, x2, x3, x4 = sp.symbols('x1 x2 x3 x4')
x = sp.Matrix([x1, x2, x3, x4])

# Define the system symbolically
g = sp.transpose(sp.Matrix([[mu * N + omega * x4 - mu * x1, - sigma * x2 - mu * x2, sigma * x2 - (mu
+ alpha) * x3, - omega * x4 - mu * x4], \
                            [-x3 * x1 / N, x3 * x1 / N, 0, 0], [0, 0, - x3, x3]]))

# Define the output symbolically
h = x1

# Truncation length for Chen-Fliess series
Ntrunc = 6

# Coefficients of the Chen-Fliess series evaluated at the initial state (using SymPy Matrix)
Ceta = sp.Matrix(iter_lie(h, g, x, Ntrunc)).subs([(x[0], x0[0]), (x[1], x0[1]), (x[2], x0[2]), (x[3],
x0[3])])

# Define the input symbols
u0, u1, u2 = sp.symbols('u0 u1 u2')
u = sp.Matrix([u0, u1, u2])
t = sp.symbols('t')


# Compute F_cu with the coefficients and product terms
F_cu = x0[0]+np.sum(sym_CFS(u, t, Ntrunc, Ceta), axis = 0)

# Substitute u0 = 1 and t = 1
F_cu_substituted = sp.Matrix(F_cu).subs({u0: 1, t: 1})

# Display the result after substitution

# Convert symbolic expression to a numerical function using lambdify
sum_T_func = sp.lambdify([u1, u2], sp.Matrix(F_cu_substituted), 'numpy')
```

Program 4. *The reachable set of the susceptible population is obtained by computing its optimal values generated by the inputs* β *and* γ *entering the system.*

```
# Define the objective function for scipy.optimize
def objective(x):
    # x is an array [u1, u2]
    return sum_T_func(x[0], x[1])

def objective_max(x):
    # x is an array [u1, u2]
    return -1*sum_T_func(x[0], x[1])

# Initial guess for u1, u2
u_init = np.array([0.1, 0.1])

# Define the bounds: (0.01<u1<0.7, 0.03<u2<3)
bounds = Bounds([0.01, 0.03], [0.7, 3])

# Minimize the function with constraints
result = minimize(objective, u_init, bounds=bounds)
result_max = minimize(objective_max, u_init, bounds=bounds)

# Print the results
print("Optimal values:", result.x)
print("Minimum value:", result.fun)
print("Optimal values:", result_max.x)
print("Maximum value:", -result_max.fun)
```

5. Conclusions

This work presented the CFSpy Python package to numerically compute the Chen-Fliess series and perform reachability analysis. For this, the definition of the iterated integral has been modified similarly to the Lie derivative: iterating the letters of the word backwards instead of forwardly as it is originally defined. The iterated integrals and the Lie derivatives are computed in batches of the same wordth length. Taking advantage of the lexicographical order of the words, the computation of each batch is done recursively. This reduces the computational time compared to the straight forward approach of iterating word by word. The algorithms of the iterated integral and the Lie derivatives are provided as well as their implementation. The functions of the CFSpy package are described. Finally, the SEIRS epidemiological model is used to show the simulations and compare the accuracy of the package.

References

[1] M. Chen, Q. Hu, C. Mackin, J. F. Fisac, and C. J. Tomlin, "Safe Platooning of Unmanned Aerial Vehicles via Reachability.," *54th Conf. on Decision and Control*, pp. 4695–4701, 2015, doi: https://doi.org/10.1109/CDC.2015.7402951.

[2] H. Seo, D. Lee, C. Son, C. Tomlin, and H. Kim, "Robust Trajectory Planning for a Multirotor Against Disturbance Based on Hamilton-Jacobi Reachability Analysis.," *Int. Conf. on Intell. Robot. and Syst.*, 2019, doi: https://doi.org/10.1109/IROS40897.2019.8968126.

[3] S. Bansal, M. Chen, , S. Hebert, and J. Tomlin, "Hamilton-Jacobi Reachability: A Brief Overview and Recent Advances.," *56th Conf. on Decision and Control*, pp. 2242–2253, 2017, doi: https://doi.org/10.1109/CDC.2017.8263977.

[4] I. Mitchell, "A Toolbox of Level Set Methods." p. 6, 2007.

[5] I. Mitchell, A. Bayern, and C. Tomlin, "A Time-dependent Hamilton-Jacobi Formulation of Reachable Sets for Continuous Dynamic Games.." 2005. doi: https://doi.org/10.1109/TAC.2005.851439.

[6] J. Maidens and M. Arcak, "Reachability Analysis of Nonlinear Systems Using Matrix Measures.." pp. 265–270, 2015. doi: https://doi.org/10.1109/TAC.2014.2325635.

[7] F. Bullo, *Contraction Theory for Dynamical Systems, 1.0 ed*. Kindle Direct Publishing, 2022, pp. 265–270. [Online]. Available: http://motion.me.ucsb.edu/book-ctds

[8] J. Scott and P. Barton, "Bounds on the Reachable Sets of Nonlinear Control Systems.." pp. 93–100, 2013. doi: https://doi.org/10.1016/j.automatica.2012.09.020.

[9] J. Meyer, A. Devonport, and M. Arcak, "TIRA: Toolbox for Interval Reachability Analysis.," *22nd ACM International Conference on Hybrid Systems: Computation and Control*, pp. 224–229, 2019, doi: https://doi.org/10.1145/3302504.3311808.

[10] S. Jafapour, A. Harapanahalli, and S. Coogan, "Efficient Interaction-aware Interval Analysis of Neural Network Feedback Loops.." pp. 8706–8721, 2024. doi: https://doi.org/10.1109/TAC.2024.3420968.

[11] S. Coogan, "Mixed Monotonicity for Reachability and Safety in Dynamical Systems.," *59th Conf. on Decision and Control*, pp. 5074–5085, 2020, doi: https://doi.org/10.1109/CDC42340.2020.9304391.

[12] C. Fan, J. Kapinski, X. Jin, and S. Mitra, "Simulation-driven Reachability Using Matrix Measures.." 2017. doi: https://doi.org/10.1145/3126685.

[13] Z. Huang and S. Mitra, "Computing Bounded Reach Sets from Sampled Simulation Traces.," *15th ACM International Conference on Hybrid Systems: Computation and Control, ser. HSCC'12*, pp. 291–294, 2012, doi: https://doi.org/10.1145/2185632.2185676.

[14] C. He, Z. Gong, M. Chen, and S. Hebert, "Efficient and Guaranteed Hamilton–Jacobi Reachability via Self-Contained Subsystem Decomposition and Admissible Control Sets.," *Control Systems Letters, Vol 7.*, pp. 3824–3829, 2023, doi: https://doi.org/10.1109/LCSYS.2023.3344659.

[15] M. Althoff and G. Frehse, "Combining zonotopes and support functions for efficient reachability analysis of linear systems.," *55th Conference on Decision and Control*, pp. 7439–7446, 2016, doi: https://doi.org/10.1109/CDC.2016.7799418.

[16] M. Fliess, "Fonctionnelles causales non linéaires et indéterminées non commutatives.." France, pp. 3–40, 1981. doi: https://doi.org/10.24033/bsmf.1931.

[17] G. Venkatesh, W. Gray, and L. Duffaut Espinosa, "Combining Learning and Model Based Multivariable Control.," *58th Conference on Decision and Control.*, 2019, doi: https://doi.org/10.1109/CDC40024.2019.9028944.

[18] I. Perez Avellaneda and L. A. Duffaut Espinosa, "Reachability of Chen-Fliess series: A Gradient Descent Approach.," *58th Annual Allerton Conference on Communication, Control, and Computing*, pp. 1–7, 2022, doi: https://doi.org/10.1109/Allerton49937.2022.9929346.

[19] I. Perez Avellaneda and L. A. Duffaut Espinosa, "Output Reachability of Chen-Fliess series: A Newton-Raphson Approach," *57th Annual Conference on Information Science and Systems*, pp. 1–6, 2023, doi: https://doi.org/10.1109/CISS56502.2023.10089740.

[20] I. Perez Avellaneda, "CFSpy," *GitHub. Note: https://github.com/iperezav/CFSpy*, 2024.

[21] P. Virtanen *et al.*, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020, doi: https://doi.org/10.1038/s41592-019-0686-2.

[22] A. Salomaa, *Formal Languages*. United Kingdom: Academic Press, 1973.

[23] K. Chen, "Integration of Paths, Geometric Invariants and a Generalized Baker-Hausdorff Formula.," *The Annals of Mathematics*, pp. 163–178, 1957, doi: https://doi.org/10.2307/1969671.

[24] C. R. Harris *et al.*, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, 2020, doi: 10.1038/s41586-020-2649-2.

[25] A. Meurer *et al.*, "SymPy: symbolic computing in Python," *PeerJ Computer Science*, vol. 3, p. e103, 2017, doi: 10.7717/peerj-cs.103.

[26] O. Bjørnstad, K. Shea, M. Krzywinski, and N. Altman, "The SEIRS model for infectious disease dynamics," *Nature Methods*, vol. 17, pp. 557–558, 2020, doi: 10.1038/s41592-020-0856-2.

Open Access | https://doi.org/10.25080/hwcj9957

SciPy 2025

July 7 - July 13, 2025

Proceedings of the 24th Python in Science Conference
ISSN: 2575-9752

Jupyter Book 2 and the MyST Document Stack

A modular, extensible, web-native stack for authoring and publishing computational narratives

Project Jupyter[1], **Evan Bolyen**[2], **J Gregory Caporaso**[1,2], **Rowan Cockett**[1,3,4], **Danny Garside**[5,6], **Chris Holdgraf**[7,1], **Angus Hollands**[7,1], **Julia Kent**[8], **Franklin Koch**[3,1], **Jim Madge**[9,5], **Matthew McKay**[10], **Mike Morrison**[3], **Fernando Pérez**[11,12,13,14,1], **Steve Purves**[1,3,4], **Min Ragan Kelley**[1], **Brian E. J. Rose**[15], **Malvika Sharan**[5], **Brigitta M. Sipőcz**[16,5,17], **Stéfan J. van der Walt**[12,17], and **Kirstie Jane Whitaker**[12,1,5]

[1]Project Jupyter, [2]Northern Arizona University, [3]Curvenote Inc., [4]Continuous Science Foundation, [5]The Turing Way, [6]Digital Research Academy, [7]2i2c, [8]NSF National Center for Atmospheric Research, [9]Alan Turing Institute, [10]QuantEcon, [11]Department of Statistics, UC Berkeley, [12]Berkeley Institute for Data Science (BIDS), [13]Eric and Wendy Schmidt Data Science & Environment Center (DSE), UC Berkeley, [14]Data Science and Technology, Lawrence Berkeley National Lab, [15]University at Albany (State University of New York), [16]Caltech/IPAC - NASA/IPAC Infrared Science Archive, [17]Scientific Python

Abstract

Jupyter Book allows researchers and educators to create books, articles, and collections of connected documents that are reusable, reproducible, and interactive. Over the past three years, Jupyter Book has been entirely rebuilt on top of the MyST Document Engine and its ecosystem of tools and standards. This new foundation introduces a scalable way to publish modern scientific content that enables interactive code and computation, structured metadata, and content reuse across contexts. It allows for a more fluid workflow between computational discovery and communication, treating computation as *first-class* in both authoring and reading. Jupyter Book 2 (JB2) is now a Jupyter sub-project, and already powers several open scientific resources including The Turing Way, QuantEcon, Project Pythia, and the QIIME 2 Framework (Q2F) documentation ecosystem. This paper introduces the principles, architecture, and capabilities of JB2, shares real-world adoption stories, and explores how these tools are shaping the future of open computational publishing.

Keywords open science, jupyter, scientific communication, publishing, authoring, reproducibility

Published Jul 10, 2025

Correspondence to
Rowan Cockett
rowan@curvenote.com

Open Access

Copyright © 2025 Jupyter *et al.*. This is an open-access article distributed under the terms of the Creative Commons Attribution 4.0 International license, which enables reusers to distribute, remix, adapt, and build upon the material in any medium or format, so long as attribution is given to the creator.

1. Introduction

Jupyter Book [1] has become a widely used tool in computational science and open publishing, powering over 18,000 publicly available books[1], lectures, tutorials, and knowledge repositories. It enables clear communication of both exploratory computation and narrative text, making it possible to document, share, and publish computational workflows in accessible formats.

With Jupyter Book 2 (JB2), the project has undergone a major transformation. Jupyter Book

[1]There are 18,152 dependencies of JupyterBook (v1 and v2, combined) based on GitHub dependency graph: https://github.com/jupyter-book/jupyter-book/network/dependents.

2 is now a distribution of the MyST (Markedly Structured Text) Document Engine[2]; every book built with Jupyter Book 2 is also a MyST project, and building a book with Jupyter Book 2 means building a MyST project. Jupyter Book 2 heavily utilizes components from the MyST ecosystem—a collection of open source tools and standards for authoring open computational narratives. This is similar to the relationship between Jupyter Book 1 and Sphinx.

The goal of this overhaul is to make structured scientific content more interoperable, composable, extensible, and machine-readable. This enables researchers to publish high-quality, interactive content with support for standards-based reuse, full-text APIs, executable notebooks, and output formats like Websites, Typst, PDF, LaTeX, Microsoft Word, JATS XML (Journal Article Tag Suite, used in scientific publishing), and more. It is a major step forward towards Jupyter's vision of thinking and storytelling with code and data [2], [3].

In this article, we: (a) document the guiding principles and design thinking behind this major release; (b) present the high-level architecture behind Jupyter Book and the MyST ecosystem; (c) compare our tools to others in the scientific publishing ecosystem; and (d) highlight case studies of notable projects.

2. The challenge: Bridging Exploration and Publication

The mission of Jupyter Book and MyST is to build composable tools and standards for communicating technical and computational narratives. This must encompass the *full lifecycle of discovery and communication* from informal lab notes and collaborative tutorials to formal research articles and textbooks. Our goal is to support authors, educators, and researchers as they share work within local communities and publish across the global scientific commons.

We are focused on two overlapping but distinct communities:

Data-driven communicators Researchers, analysts, and educators who need to share working, reproducible content with collaborators, students, or the public. These users rely on Jupyter, Python, and other computational tools in their daily workflows, but often lack well-integrated systems to communicate what they've done or how to reproduce it.

Scholarly Authors and Publishers Scientists and academics preparing content for peer review, journals, repositories, or formal documentation, and organizations that serve these users. They need structured metadata, citation systems, and outputs compatible with academic and institutional publishing pipelines.

Both of these groups are underserved by the current ecosystem of open source technology. Today's scientific workflows rely on computational tools like Jupyter Notebooks, yet the systems used to communicate and publish these workflows are fragmented and poorly integrated. For example, researchers often do their work in an interactive computing environment designed for exploration, and must then adopt an entirely new toolchain for publishing their work. As a result, researchers must spend time adapting to new workflows, duplicating and reformatting their content just for the sake of publishing. Because output-specific pipelines, such as PDF generation, are fairly constrained, published work is often stripped of its computational context, leaving no mechanism to access software environments and reproduce results.

Over the years, there have been many attempts at improving this workflow with new technologies and standards (several are discussed below), but none have provided the combination of flexibility, standards-based implementation, and community governance

[2]MyST is pronounced like "missed" (see the docs).

needed to cover the entire lifecycle from idea to communication described above. Jupyter Book 2 and the MyST Ecosystem is an attempt to bridge this gap.

2.1. Overview of Jupyter Book 2's architecture

This section describes the architectural structure and components that drive the Jupyter Book 2 stack and the MyST ecosystem. At the heart of JB2 and the MyST ecosystem are four pieces:

- **The MyST Document Engine** (https://mystmd.org), a publication system for authoring computational narratives.
- **The MyST Specification** (https://mystmd.org/spec), a structured and semantically-meaningful way of representing computational content in an Abstract Syntax Tree (AST).
- **MyST Renderers and Themes** (https://mystmd.org/guide/overview#overview-themes), transform MyST ASTs into end-user outputs (websites, PDFs, etc.).
- **MyST Markdown** A lightweight extension of CommonMark Markdown[3] and GiHub Flavored Markdown (GFM[4]) which adds roles and directives (described below).

JB2 combines these and other MyST components into an end-user application meant for community knowledge bases and books. This new architecture is designed to enable dynamic rendering, multi-format output, and composable content workflows—backed by a structured document model that makes metadata, structure, and computation portable and machine-readable.

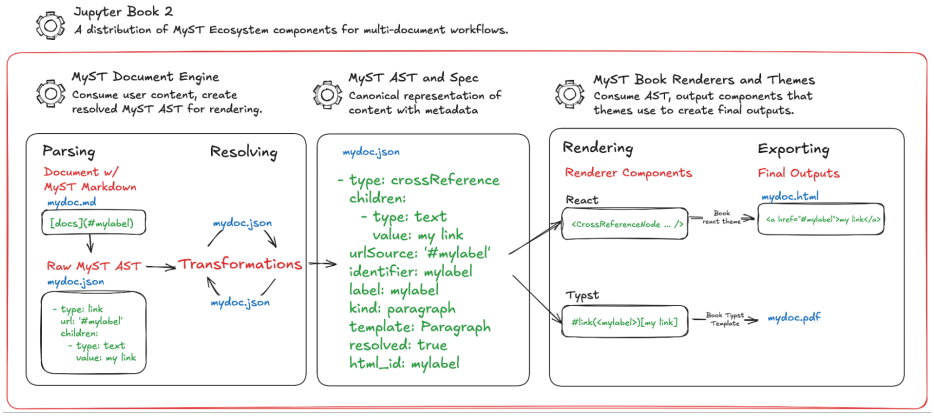


Figure 1. ***High-level overview of Jupyter Book 2 components.*** *JB2 combines several components in the MyST ecosystem, configured for multi-document workflows such as community knowledge bases, websites, and books. The MyST Document Engine parses MyST Markdown (and other flavors of text-based content) into an Abstract Syntax Tree (AST) that follows the MyST Document Specification. It resolves and transforms this AST to enrich it with extra document metadata (such as resolving DOIs). The engine then renders this AST to React components that are bundled with the Book Theme, which makes up the end-user application most-often used with JB2.*

2.2. The MyST Engine: A simple, powerful, and extensible document engine

The MyST Document Engine allows users to **parse** documents into a structured format, **execute** computational content, **transform** them to include metadata and resolve references, and **output** a resolved MyST Abstract Syntax Tree (AST, described in more detail below). It is run via the command-line and is written in TypeScript. It parses many types

[3] CommonMark (https://commonmark.org) is an unambiguous specification for the Markdown language with a formatting test suite.

[4] GitHub Flavored Markdown (https://github.github.com/gfm/), often shortened as GFM, is the dialect of Markdown that is supported for user content on GitHub.

of inputs—Markdown files, Jupyter Notebooks, LaTeX documents, external MyST projects, configuration files, etc—and ensures that all of their content is embedded in this output AST.

2.3. The MyST Specification: A Structured Computational Document Format

The **MyST Specification** defines all of the types of content and metadata that make up a computational narrative. It is meant to be a cross-platform standard that can be used by various applications and parsers in order to facilitate sharing and re-use of content.

The MyST Document Engine outputs a **MyST Abstract Syntax Tree (AST)** that follows the MyST Specification. The AST is a machine-readable representation of each document's content and structure. It preserves not only the narrative content, but also computational blocks, computational outputs from Jupyter Notebooks, citations, figures, and metadata such as author affiliations and license information. These are all built into a structured, versioned AST. It is a **Canonical Document Representation** that can be parsed and rendered into user-facing outputs by renderers and themes.

MyST Markdown is a flavor of Markdown written for the MyST Specification that gives authors quick access to these and other extension points. MyST Markdown supports syntax for all of the content types supported in the MyST Specification. It is used both in Jupyter Book 2 and the MyST Engine, as well as in the Sphinx ecosystem via the myst-parser extension.

As an example, see the example **note directive** written in MyST Markdown below (for more information on directives, see the sections below).

```
:::{note} Admonition title
Admonition **body**.
:::
```

The MyST Document Engine parses this Markdown content into an AST that follows the MyST Specification. It looks like this:

```
- type: admonition
  kind: note
  children:
    - type: admonitionTitle
      children:
        - type: text
          value: Admonition title
    - type: paragraph
      children:
        - type: text
          value: 'Admonition '
        - type: strong
          children:
            - type: text
              value: body
        - type: text
          value: ...
```

Renderers then convert this AST into various output formats.

2.4. MyST Renderers and Themes: Tools to convert MyST AST into user-facing products

MyST Renderers and Themes are tools for converting documents that follow the MyST Specification into user-facing outputs, such as websites and PDFs. Renderers are essentially a collection of rules for converting each type of content (called a "node" in a MyST AST) into

an output. For example, the MyST AST for the Note admonition above would be rendered into a React component, and exported to HTML via the static HTML export command `jupyter book build --html` like so:

```
<-- React component used by the Book or Article theme -->
<Admonition title="Note" kind="{AdmonitionKind.note}" color="blue">
  This is an admonition!
</Admonition>

<-- React component converted to this HTML after static HTML export -->
<aside className="myst-admonition myst-admonition-note">
  <div className="myst-admonition-header">
    <InformationCircleIcon className="myst-admonition-header-icon" />
    <div className="myst-admonition-header-text">Note</div>
  </div>
  <div className="myst-admonition-body">This is an admonition!</div>
</aside>
```

The **Book React Theme** defines the basic structure and styles of a Jupyter Book 2, and is the default result that most users get when they render their books as websites. The React component above is styled by the Book Theme into this final product:

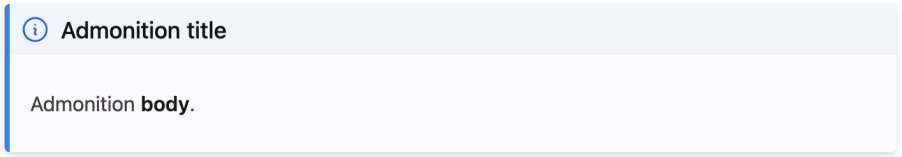


Figure 2. ***Example admonition component.*** *The final product of a MyST Admonition rendered by the Book Theme. Arguments and keywords in the `{note}` directive influence the look and feel of this final product.*

Importantly, **MyST Renderers can be written for any kind of output**. For example, there is a **Typst** renderer and a **LaTeX** renderer, which can themselves be customized with **themes** to generate different types of outputs. Jupyter Book 2 ships with a default Book Theme for each type of renderer, but we imagine our ecosystem of users extending this to include, for example, LaTeX templates for a variety of Journals. The `myst-templates` GitHub organization (github.com/myst-templates) already hosts a collection of community-maintained templates for rendering and theming MyST AST into various kinds of outputs.

2.5. Jupyter Book 2: A distribution of MyST components

Finally, Jupyter Book 2 is a lightweight distribution and configuration layer on top of these and other components of the MyST ecosystem. It is meant to provide an opinionated experience of MyST–for example, by turning on certain features by default, and by providing an out-of-the-box theme meant for multi-document websites. At its core, Jupyter Book 2 functionality is leveraging components in the MyST ecosystem.

This is a pattern that we wish to encourage and grow in the MyST ecosystem. Tools like the MyST Document Engine aim to be unopinionated, modular, configurable, and extendable. They're primarily meant for use by **power users and developers** who are interested in configuring MyST tools for their own needs/community, or building MyST tools into separate applications. Jupyter Book 2 is meant for **end-users like researchers and educators** who want an opinionated system for communicating computational narratives that "just works".

We envision that a broader *ecosystem* of high-level tools and services—*similar to Jupyter Book 2* itself—may be built on top of the MyST Document Engine, Specification, Renderers,

and other components. With that in mind, we now turn to the design principles that drove the architecture of the MyST ecosystem, and the value that they bring to the open science community.

3. Design Principles

Jupyter Book 2 is designed to meet the practical needs of researchers, educators, and publishers who communicate with—and about—data. It also supports developers in the open science ecosystem who build applications on shared standards and tools. We have grounded the design and development of the JB2 stack with the following principles.

Simple to use, easy to extend Users should be able to run common workflows with minimal complexity, dependencies, setup, and configuration. Power-users should be able to extend this functionality with minimal effort.

Use standardized, machine readable formats There should be a canonical, machine-readable representation of all content, allowing it to be re-used and rendered into multiple outputs without losing structure, metadata, or intent.

Content should be modular and composable Canonical content should be made up of small, reusable components to enable reuse, cross-referencing, and re-mixing into diverse outputs.

Computation should be first-class Code, data, and outputs are treated as integral components of the research narrative, for example, integrated and reproducible ways to calculate figures, show inline values, and cache computations.

Open science infrastructure should be governed by an open community Jupyter Book and MyST should be stakeholder-led, with a diverse community of organizations and people from the open science community that guide its development.

Below, we describe each in more detail, how it is implemented in the MyST ecosystem, and the value that it brings to communicating open science with computational narratives.

3.1. Simple and quick to use, easy to extend

First and foremost, it should be simple and straightforward for authors to create and publish computational narratives with Jupyter Book 2, requiring little configuration and customization. If the out-of-the-box functionality of Jupyter Book 2 doesn't fit an author's needs, there should also be simple mechanisms to *extend* Jupyter Book's functionality. The MyST ecosystem should be composed of independent pieces that can work together as a whole, with natural ways to extend for new functionality.

3.1.1. *MyST's plugin ecosystem*

The MyST Documentation engine has several key **extension points** that allow users to add new functionality in a controlled way[5]. Below are a few key extension points:

Roles and directives are mechanisms for adding new types of in-line and block-level content (respectively), allowing developers to define content abstractions (similar to functions in a programming language) for use by authors. Users may define their own roles and directives using JavaScript, Python, or any other language that reads and writes from `stdout`. Building a new role or directive involves taking MyST AST as input, applying transformations, and returning AST.

[5]Examples are available in the appendix

Transforms are a mechanism for transforming the content of one or more MyST ASTs. It is used to enrich content as part of the build process. For example, by collecting `{figure}` nodes throughout the page and aggregating them into a single gallery.

Renderers are a way to export MyST AST into arbitrary types of outputs. They're useful for converting the "Canonical document" into many different formats or styles without changing the original content. They are explained in sections below.

3.2. Content should be canonical and machine readable

The MyST ecosystem emphasizes a **"content-first" model**, where authoring is separated from output format. This reduces the friction and wasted time associated with re-packaging source content in order to build a slightly different output.

3.2.1. *The MyST AST is semantically structured*

The MyST Document Engine produces a machine-readable abstract syntax tree for every page. This well-structured build artifact is a canonical document that describes the underlying structure, content, and metadata of the document. This internal representation captures not just the text of a document, but also its semantic components: citations, headings, roles, directives, code blocks, figures, equations, and embedded metadata like Open Researcher and Contributor IDs (ORCIDs), Digital Object Identifiers (DOIs), or author affiliations via the Research Organization Registry (ROR). A canonical version also allows content to be rendered into multiple outputs—without losing structure, metadata, or intent.

By embedding all of the relevant metadata in a MyST Document, it is possible to gain a more rich understanding of the intent, context, and learning contained within a document. It is also possible to (re)use subsets of this information for various end purposes (for example, to re-render a figure, or to provide metadata for a scientific discovery engine). We give specific effort to support the open-science ecosystem, especially around structured, scholarly metadata that help discoverability (e.g., DOIs, ORCIDs, RORs).

3.2.2. *Many renderers operate on the same AST*

The AST serves as the canonical version for rendering content into multiple target formats: web-native HTML, JATS XML, PDF via Typst or LaTeX, Microsoft Word, and more. Exporting to a target format is done with a **renderer** that consumes the AST and applies format-specific styling, layout, and transformations. Our attention is primarily focused on the React-renderers, Typst PDF templates (a new PDF renderer that is significantly faster than LaTeX), and JATS XML used in scholarly publishing. For example, the SciPy Proceedings for 2024 and 2025 both use the React renderers, the Typst renderer for PDF creation, and JATS XML for all articles [4].

By treating that document AST as a first-class build output, authors can write their content once, and publish in multiple places without manually adapting their content for each target format. For example, a MyST Document can be built once, and the resulting AST can be rendered into an HTML website as well as multiple LaTeX documents for submission to various journals. Paired with the machine readability of the AST (the `.json` file served with websites), it is possible to render multiple "views" of the same underlying content without creating confusion for where the source of truth lies.

Additionally, having a standard and metadata-rich AST structure allows many different workflows to be served from the same document engine. For example, while this article focuses on the multi-document workflow of Jupyter Book 2, the MyST Document Engine has also been used as part of *scholarly article publishing pipelines*, which are composed of independent articles that benefit from the same ability to cross-reference one another.

For example the Proceedings of SciPy have been published with the MyST Document Engine for the last two years, the Notebooks Now! project [5] incorporated MyST-based publishing into the American Geophysical Union proceedings for the last several years, the NeuroLibre project has built a pre-print server using the MyST engine, and Curvenote builds a publishing platform using MyST components [4]. By enabling a variety of publication output types with the same underlying document engine, the MyST ecosystem can reduce duplicated technology and streamline the workflow from data science and documentation to publishing.

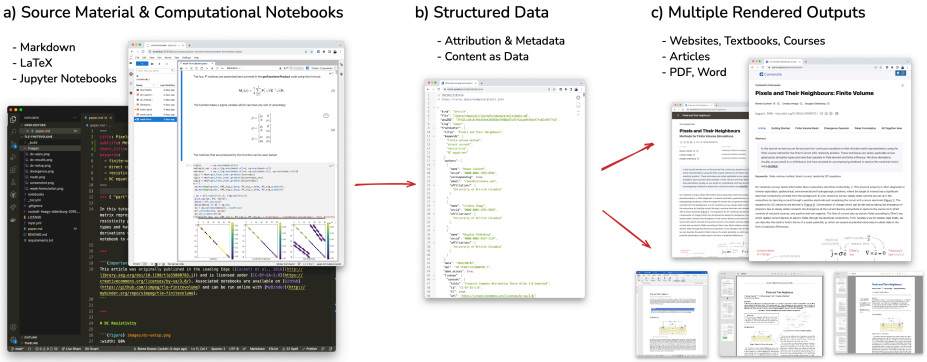


Figure 3. ***Rendering a single AST into many output types.*** *In this workflow, a single collection of source files (a) are built into a MyST AST (b) that serves as the canonical representation of the document. One or more renderers operate on that AST and convert them into outputs (c) such as HTML, PDF, Microsoft Word, etc. (Figure adapted from R. Cockett, S. Purves, F. Koch, and M. Morrison [4], showing content from R. Cockett, L. J. Heagy, and D. W. Oldenburg [6]; CC-BY-SA-4.0).*

3.3. Content should be modular and composable

The MyST ecosystem is designed for composition. Documents are structured into small, addressable components: sections, figures, equations, terms, figures, outputs—that can be reused across books, projects, and publications. This modularity supports new patterns of collaboration and publication: a figure generated in a course textbook can be cited and embedded in a tutorial or a journal article without duplication or loss of provenance. A community policy or database can be re-used throughout many pages in its documentation without duplicating information and creating multiple sources of truth. Additionally, configuration for a collection of projects can be shared and reused.

3.3.1. *The MyST AST modularizes content for easy parsing and re-mixing*

Each page or resource in a Jupyter Book project is assigned a unique slug (a human-readable URL) and corresponding AST. These can be referenced across projects, embedded in-line with other page content, or previewed when users hover over links. This is based on both the MyST Specification and a machine-readable cross-reference manifest `myst.xref.json`[6]. Similar to intersphinx references, a feature for cross-project linking in the Sphinx documentation generator, the `myst.xref.json` gives stable links to any aspect of the content. This makes it easy to programmatically learn the structure of a Jupyter Book, referenceable labels, and the location of the canonical content for each document.

An author can then reference labels in a different Jupyter Book or MyST project with an `xref` link. For example, the MyST Markdown below demonstrates referencing an external project called `other-project` (assuming that the URL of the other project is defined in your MyST configuration).

[6] An example is available in the appendix

```
[Link text](xref:other-project#label-name)
```

This also allows authors to reference *any other Jupyter Book or MyST project* , and use `xref:` links to link or embed the content. These links are either generated at build- or render-time, and allow for dynamic linking between projects with a simple user interface. Paired with structured JSON APIs for document content (described below), it also allows machines to efficiently access data in a Jupyter Book (for example, to enable hover-previews or to identify source files for HTML pages for use by Large Language Models)[7].

3.3.2. *API-Accessible and Federated by Design*

The Canonical AST of each document is *designed to be re-used by others.* It is packaged as a machine-readable `.json` file to encourage programmatic re-use. It is also bundled with web outputs to encourage external services or readers to access and utilize the AST via a machine-readable metadata endpoint (`/pagename.json`).

Machine readable content accessible via an API is one of the most transformative aspects of the JB2 architecture. By making each canonical document accessible via a `.json` API endpoint that exposes its AST and metadata, content can be programmatically indexed, referenced, embedded, transcluded, or transformed without needing to scrape or recompile the site. Readers and developers can query structure, citations, and content across books, enabling new tools like federated search, content previews, and live citation graphs. It also enhances the reading experience by enabling full-text search, granular cross-referencing, and live content embedding across projects.

Moreover, by aligning with community standards (e.g., JATS, DOIs, ORCID, ROR), the system supports today's scholarly publishing workflows, remains grounded in lightweight, Markdown-based authoring, and opens up possibilities for more composable workflows in the future for scientific authoring [7]. This approach lays the groundwork for a **federated publishing network** where content from different lab groups, institutions, or journals can be referenced, reused, and remixed while maintaining proper attribution and structure.

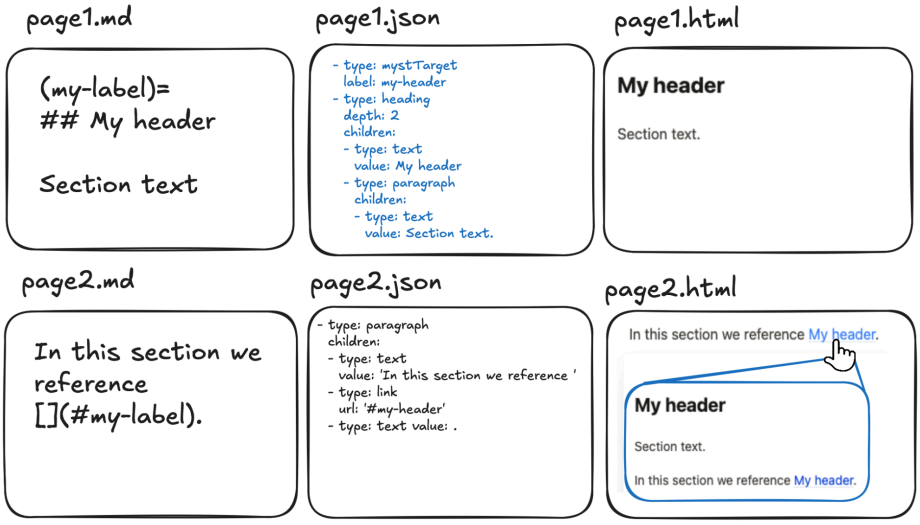


Figure 4. ***Example of how machine-readable AST enables cross-page and cross-project hover references.*** *In this abstract example we demonstrate how the source content of a page (`page1.md`) is parsed into an AST (`page1.json`) with a unique label called `my-label`. A second page (`page2.md`) references this label.*

[7]The hover-xref project in sphinx was an attempt at doing this

At rendering time (`page2.html`) the content of the label is pulled into the page and displayed via a hover-pop-up.

3.3.3. *The MyST Specification is a versioned, community-governed standard*

Just as the Jupyter Notebook format (`.ipynb`) has enabled interoperability across dozens of tools by serving as a stable, well-defined format, the MyST AST aims to serve a similar role for scientific communication. The AST specification is versioned and documented, and efforts are underway to standardize it further via the `myst-spec` initiative. We are also investing in forward-and-backwards interoperability in this specification in the MyST Document Engine.

By creating a community process for defining the evolution of the MyST Document and Markdown standards, we aim to create a participatory process to ensure that this ecosystem continues to serve the needs of its key stakeholders: members of open science, open source, and open knowledge communities who are passionate about communicating with computational narratives. Coupled with the dependability of a stable specification, and a technology stack that allows for graceful upgrading and downgrading, this allows developers, publishers, and downstream platforms to build confidently on top of the MyST ecosystem.

3.4. Computation should be first-class

A key challenge in communicating with data lies in the fact that tools and systems for communicating are *separate* from tools for making discoveries with data and computation. Many researchers begin their communication journey in a computational notebook—exploring data, testing hypotheses, or sharing early-stage insights. Bringing that work into a readable, shareable, and publishable format traditionally requires significant rework: copying plots into a document, cleaning outputs, manually formatting results, re-structuring to suit publication format requirements. These extra steps not only introduce friction, but also weaken reproducibility and create a disconnect between the computation and its communication.

3.4.1. *Execution and caching are built into the MyST Document Engine*

MyST comes with infrastructure to parse computational content, execute it[8], cache the results, and insert them into the MyST AST so that they may be used and referenced. Computational content can come in two forms:

Jupyter Notebooks (`.ipynb` files) This is the standard structure for Jupyter Notebooks, and the MyST Document Engine will use the Jupyter kernel metadata in the notebook to execute each cell and embed its content in the output.

MyST Markdown Notebooks (`.md` files) The MyST Engine allows authors to define the structure of a Jupyter Notebook with a Markdown file, using a special directive called `{code-cell}`. Paired with text-to-notebook conversion tools like Jupytext, this allows for two-directional conversion between `.ipynb` files and MyST Markdown Notebooks.

3.4.2. *Computational outputs are treated like any other content*

Aside from simply executing and inserting outputs into a MyST document, these outputs can also be used similar to any other type of MyST content. For example, authors can **attach a label to a notebook cell** in order to **reference its outputs** like they would any other kind

[8]The Jupyter Book team intends to extend computational capabilities to use kernels and execution services that are hosted elsewhere. This will make it possible for the computation in a Jupyter Book to be executed in a remote environment that may have access to more complex resources or sensitive data (for example, as part of a journal publication pipeline for reproducibility).

of content. For example, the `{code-cell}` directive below creates a plot with `matplotlib` in `computational-page.md`:

```
(full-plot-context)=
This code generates an image:

```{code-cell} python
:label: my-plot
plt.scatter([1, 2, 3], [4, 5, 6])
```
```

In `another-page.md`, the author is able to insert this output into a figure with a caption like so:

```
Below you see the results of my computation:

:::{figure} #my-plot
Here's a caption for the image referenced by `my-plot`, see [the full plot context](#full-plot-context) for details.
:::
```

Our goal with Jupyter Book 2 is to integrate computational content at every level. Code, figures, outputs, and interactive elements are embedded directly in the narrative and treated as primary content—not sidebars or supplemental material. This increases the likelihood that authors will interweave computational content into the reading experience, increases the reproducibility and re-usability of computational ideas, and allows for more powerful storytelling through computation, data, and software [2].

3.4.3. *In-page computation brings computational interactivity to the reading experience*

Beyond static computational outputs, JB2 embraces the web-native workflows of modern data science by allowing computation to power interactive experiences with a live kernel. Executable content can be powered by any Jupyter Kernel by leveraging MyST's native integration with Thebe, which enables static documents to become interactive by linking code cells with a computational kernel. Thebe can leverage kernel providers from sources like Binder [8], JupyterHub, or a local Jupyter Server, allowing readers to rerun computations on demand, right from the page without leaving the context of the narrative.

Providing computational interactivity without leaving the page drastically reduces the friction required to interact with computational ideas. It gives authors the ability to power their computational content with arbitrary software, computational resources, or access to data (for example, authors could leverage a service that operates a BinderHub that provides access to GPUs and a 10 Terabyte dataset, and provide interactive kernels for their readers that are powered by that hub). Moreover, modern efforts to package computational environments in web-native toolchains like WebAssembly (via tools like JupyterLite) will allow for interactive execution directly in a reader's browser.

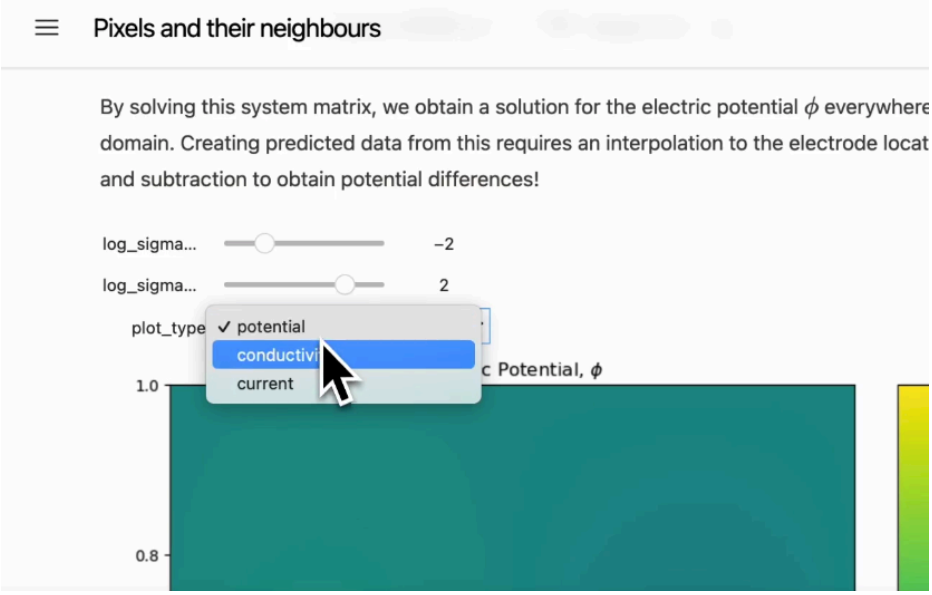


Figure 5. *Embedded notebook cells with live computation directly in an articles with computation backed by Jupyter. These can be running on BinderHub or directly in your browser through JupyterLite. Originally published in R. Cockett, S. Purves, F. Koch, and M. Morrison [4].*

3.5. Open science infrastructure should be governed by an open community

Finally, we believe it is crucial for open science technology and standards to be governed and led by a community of stakeholders, not driven by a single person or organization. Jupyter Book and MyST are community-led projects, developed and governed by an open community of contributors, and hosted under the umbrella of Project Jupyter. The tools in the Jupyter Book 2 stack are maintained by a contributor community that includes open infrastructure organizations, universities, independent developers, and end-users who rely on them in practice.

To facilitate participatory and collaborative design as the Jupyter Book 2 ecosystem evolves, all team practices and policies are defined in the Jupyter Book Team Compass, which inherits its open governance and practices from Project Jupyter. The contributing guide provides detail of the overall stack to help others understand and participate in the development. The MyST Enhancement Proposal process facilitates community feedback to evolve the MyST Specification and Markdown language. These practices aim to allow the MyST ecosystem to remain a community-supported standard while continuing to evolve to meet the needs of its users.

3.6. Next directions for Jupyter Book 2

This article describes the high level architecture, goals, and guiding principles for the JupyterBook project; there is still much work to be done to fully realize our ambitions. We have tried to make it clear where our goals are still aspirational, and we invite others to join our community as we develop these ideas further.

There are several directions where we are particularly excited to make improvements.

We aim to lean further into the idea of modular and composable content by **providing directives and roles that create summaries and views of MyST content**. This could be used to create summary content like galleries and blogs with local or remote MyST documents and Jupyter Books. We wish to **add support for API documentation to enable software development documentation**. We aim to improve **integration of computational work-**

flows with external services and infrastructure such as JupyterHub or Binder to allow authors to leverage extra resources or sources of data as part of generating their books. Finally, we aim to **enable more extension points in the underlying MyST Engine** to give authors and developers more control over the outputs, look and feel, and new kinds of functionality that MyST can support.

4. Comparison to Other Tools

Jupyter Book 2 sits among a number of tools for authoring and publishing computational and scientific content. These include static site generators, document converters, and academic publishing frameworks. Many of these systems are well-suited to particular workflows —whether technical documentation, reproducible research, or educational materials—but often focus on specific output formats, assume linear document structures, or lack integration with browser-based and computational environments. Moreover, most document engines are produced by a single person or organization, rather than a multi-stakeholder community. JB2 is designed to address these gaps, particularly for communities working in open science.

Jupyter Book 1, developed by the Executable Books Project, is built on Sphinx, a Python documentation generator that enabled integration with reStructuredText, MyST extensions, and a range of HTML output themes. While powerful and extensible, Sphinx was originally designed as a documentation engine for software projects. Support for scientific markup and computation has been added by the community, extending its original design. However, its design principles and architecture are not optimized for the browser-based authoring, computation, content re-use, and efficient multi-output exporting workflows that are core to the JB2 stack.

Jupyter Notebooks, developed by Project Jupyter, is a document format and web-based interface for authoring computational narratives with data. Jupyter Notebooks have support for interactive outputs and computation that is similar to Jupyter Book. However, they focus on individual documents rather than multi-page projects with references between them. They are also designed for data science and exploration rather than communicating more complex and re-usable computational narratives. Jupyter Book 2 *complements* the Jupyter Notebook by providing a more rich feature set and workflow that focuses on authoring and reading documents. A Jupyter Book can be written as a collection of *Jupyter Notebooks*, and knows how to parse MyST Markdown in notebook cells. Jupyter Notebooks and Jupyter Lab can also incorporate some of Jupyter Book 2's functionality into their interfaces via the `jupyterlab-myst` extension.

Quarto, developed by Posit PBC, is a flexible publishing system supporting Jupyter notebooks, Markdown, and multiple output formats including HTML, PDF, DOCX, and JATS. Quarto is built on top of Pandoc and is tightly integrated with the R, RStudio, and Positron ecosystems. At this time, Quarto is more polished and further in its development lifecycle, with better support for features like PDF rendering (e.g., sub-figures and tables), and presentation output formats. While Quarto has excellent support for static HTML outputs, it does not focus on the principles of modularity and composability which are core to Jupyter Book 2 (e.g., it does not have content APIs and structured build outputs meant for reuse, nor expose packages that can be used by other tools). Quarto is built directly on Pandoc, which uses a GPLv2 license; the Quarto command line tool is MIT licensed since version 1.4, however, many of the core packages, extensions, and utilities are not permissively licensed. Finally, Quarto is developed by a single company, in contrast to Jupyter Book which is governed and developed by a multi-stakeholder community, and uses a permissive MIT license to maximize the re-use of its technology for individual and commercial purposes.

Other documentation and markup systems like Docusaurus, MkDocs, Hugo, quarkdown, typst, and djot are popular choices for technical documentation, developer blogs, PDF generation, and websites. These tools offer good performance, templating, and theming options, and in some cases also focus on extensibility. They are not designed for scientific publishing or computational narratives and they lack features such as executable content, math, and citation handling, semantic metadata, or integrating with persistent identifiers in the scholarly ecosystem (e.g., DOIs, RRIDs, or RORs) - all of which are built into the JB2 ecosystem.

While these tools each serve their respective audiences well, JB2 is focused specifically on the needs of computational and scientific communities. It supports both narrative and executable content, prioritizes open infrastructure and interoperability, and is designed to integrate with existing Jupyter workflows and the wider open-science ecosystems.

5. Case Studies

Jupyter Book 2 is already being used across a broad range of scientific and educational projects. From national training initiatives to domain-specific research platforms, these case studies demonstrate how JB2 supports reproducible publishing, modular content reuse, and scalable collaboration.

The Turing Way [9] is an open guide to reproducible, ethical, accessible, and inclusive data science. With hundreds of contributors and a highly active community, the project has adopted Jupyter Book 2 to manage its growing library of community-authored chapters and living documents. The structured document model and improved metadata handling in JB2 have made it easier for the Turing Way's community members to learn from, reference, and leverage one another's work. The move from Sphinx to MyST has greatly simplified the technology underlying The Turing Way, which makes it easier for new contributors to build the Turing Way locally and contribute to it. They have been able to reduce the complexity of their own infrastructure by implementing custom features as MyST plugins. This has simplified the book's deployment process and better allowed these features to be shared as modular, generic plugins rather than Turing-Way-specific scripts.

Project Pythia [10] is a community-driven effort to advance computational geoscience education, training, and knowledge-sharing through open-source learning materials. Pythia's collection of cross-referenced Jupyter Book sites draw from over 75 repositories and large numbers of contributors, making the structured content model and cross-repository referencing features of JB2 particularly valuable. Pythia recently transitioned to JB2. There were two standout features to the community: (1) the ability to compose and share configuration between sites, including things like footers, shared references, navigation links; and (2) the ability to cross-reference content between un-related projects. Both of these build on the composability and modularity designs that are at the core of JB2. As an inclusive open-geoscience community effort, Project Pythia aims to build tools and practices that reduce participation barriers and incentivize working geoscientists to share their discipline-specific data fluency. Shared configurations in JB2 substantially reduce the boilerplate code in individual repositories, clearing the way for contributors to focus on their science content while a core team of maintainers efficiently advances the shared computational and web-publishing infrastructure.

The QIIME 2 Framework (Q2F) is a platform for biological data science tools, originally developed for the QIIME 2 microbiome bioinformatics toolkit, but now supporting tools across diverse subdomains of biology including multiplexed serology and pathogen genomics. Q2F has adopted JB2 to support its documentation ecosystem which includes documentation that is general to users of all Q2F-based tools (e.g., creating and using a Q2F data artifact

cache) and documentation that is specific to subdomains (e.g., an end-to-end tutorial for microbiome data analysis). JB2 enables embedding of content across documentation projects, such that general content can be embedded in domain-specific content where relevant, making it easier to hide the software stack complexity from end users while avoiding duplication of source content.

QuantEcon provides open educational resources in economics and quantitative methods, built using Python, Julia, and Jupyter. The transition to JB2 included investment in a custom theme for JupyterBook, which allowed the project to modernize its publishing infrastructure while maintaining interactive notebooks and supporting export to multiple formats including HTML, PDF, and `.ipynb`. The upgraded theme allowed for adoption of new features like hover cross-reference between exercises and solutions using the proof and exercise extensions.

These case studies demonstrate the flexibility of JB2 to meet the needs of both education and research, large and small teams, and varied technical audiences. Each project benefits from the same core strengths: modular content, structured metadata, and tight integration with Jupyter.

6. Conclusion

Jupyter Book 2 is an open-source publishing system for computational and scientific content. It enables authors to combine Markdown, notebooks, structured metadata, and executable code into documents that can be rendered as websites, PDFs, JATS XML, Word documents, and more. By rebuilding the system around a structured document model and a web-native engine, MyST Document Engine, JB2 supports workflows that are modular, reproducible, and extensible. These qualities are essential for modern research and education, and bridge the gap between exploratory computation and formal, published communication, enabling authors to spend more time on data-driven discovery and communicating ideas, and less time re-purposing their workflows for publication.

Unlike general-purpose documentation systems, static website builders, or single-format converters, Jupyter Book 2 is *built by and for* researchers, educators, and institutions working with code, data, and narrative. Each page produced with JB2 has a structured, machine-readable representation that can be queried, embedded, or reused via an API. This enables a federated publishing model, where content from different projects, lab groups, or institutions can be referenced, previewed, and composed together—without duplication or loss of structure. The result is a more connected and discoverable ecosystem of scientific knowledge, where modular content can be maintained in place but surfaced in multiple contexts, with proper attribution and rich metadata preserved.

The Jupyter Book project is openly governed under Project Jupyter, actively maintained, and already powers many high-impact documentation and communication initiatives across domains. From reproducible data science guides like The Turing Way, to educational curricula like QuantEcon and Project Pythia, to domain-specific platforms like the QIIME 2 Framework, Jupyter Book 2 is helping authors publish richer, more reusable, and more transparent scientific outputs.

As scientific publishing continues to evolve, Jupyter Book offers a model for infrastructure that is interoperable, open by design, and responsive to the needs of its users. It is not just a tool for building books—it is a foundation for the next generation of computational publishing.

7. Appendix

7.1. History & Background

In this section, we share a brief history of the Jupyter Book project with the goal of providing context for the path that led to JB2, as well as to share credit and acknowledgement for the hundreds of contributors that have helped the project over the years.

JB2 is the third significant re-write of the Jupyter Book stack in the past decade, each of which involved hundreds of collaborators and users. The original version was built in 2018 as a loose collection of template files utilizing nbconvert for execution and Jekyll for rendering notebooks into websites, originally built for the Data 8 textbook.

In 2020, Jupyter Book was re-written and formally released as Jupyter Book 1 (JB1). This was built on the Sphinx documentation generator, and resulted in the creation of the MyST Markdown syntax and parser for Sphinx. This work was largely funded by the Executable Books Project, a funded by the Sloan Foundation (Grant #9231), Jupyter Meets the Earth (NSF grant #1928406). While the Executable Books Project has completed, maintenance of the Jupyter Book 1 stack continues to this day.

Between 2020 and 2023, the Executable Books Project began a parallel collaboration with the company Curvenote to explore a standards- and web-based workflow for the MyST Markup language (funded in part by Alberta Innovates and the Stanford Doerr School of Sustainability). As a result, Curvenote integrated their document engine as an upstream project in the `executablebooks/`organization and the project developed it together from there. This became the starting point for the `mystmd` stack described in this article.

From 2023 onward, the wider community has invested in `mystmd` and improved the capabilities of the command line tool, parsing capabilities, templates, and web themes. The next version of JupyterBook, Jupyter Book 2 (JB2), is built on top of this engine. In 2024, the project moved from being an independent organization to being incorporated as an official Jupyter sub-project, standardizing on using and stewarding the MyST document engine (See #123).

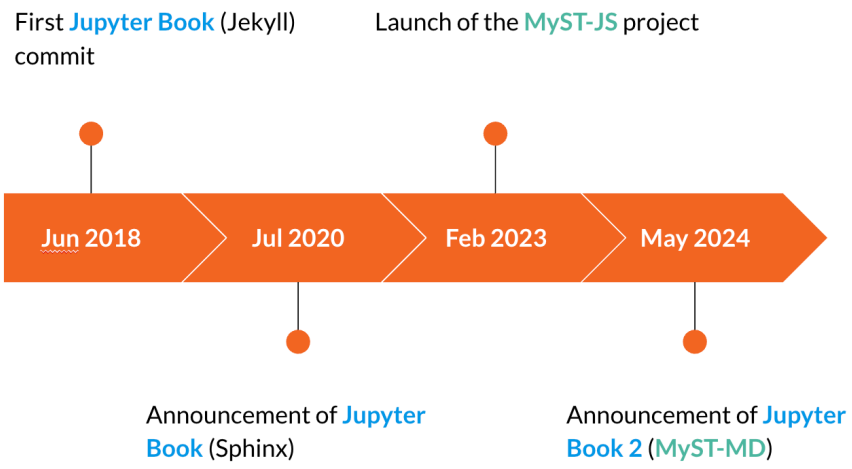


Figure 6. *Major phases of Jupyter Book development.*

7.1.1. *A note on migrating from Jupyter Book 1*

A key goal of Jupyter Book 2 was to leverage the design and standards from Jupyter Book 1 and the Sphinx stack in order to facilitate the upgrade process. As a result JB2 leverages

the same MyST Markdown syntax as JB1 with minimal disruption. It also aims to expose a key subset of the extension points that were available in JB1 and Sphinx (for example, roles, directives, custom transforms, etc). While there is a subset of functionality that is still unique to Sphinx, the JB2 team is focusing their efforts on developing key missing functionality to narrow this gap. See the Jupyter Book 2 migration guide for more information.

7.2. Examples of plugins

Directives: Here is an example of a directive logic that generates an `{image}` node in the AST by pulling a random image from picsum. Note how we define arguments and options, similar to how a function would be defined in a programming language.

```
const picsumDirective = {
  name: 'picsum',
  doc: 'An example directive for showing a nice random image at a custom size.',
  alias: ['random-pic'],
  arg: {
    type: String,
    doc: 'The ID of the image to use, e.g. 1',
  },
  options: {
    size: { type: String, doc: 'Size of the image, for example, `500x200`.' },
  },
  run(data) {
    // Parse size
    const match = (data.options?.size ?? '').match(/^(\d+)(?:x(\d+))?$/);
    let sizeQuery = '200/200';
    if (match) {
      const first = match[1];
      const second = match[2];
      sizeQuery = second ? `${first}/${second}` : first;
    }

    const idQuery = data.arg ? `id/${data.arg}/` : '';
    const url = `https://picsum.photos/${idQuery}${sizeQuery}`;
    const img = { type: 'image', url };
    return [img];
  },
};

const plugin = { name: 'Lorem Picsum Images', directives: [picsumDirective] };
export default plugin;
```

Transforms: Here is an example of transform logic that parses the MyST AST and replaces **strong** styling with *emphasis* styling. It operates on all `node` objects in the document using the `selectAll` utility function.

```
const plugin = {
  name: 'Strong to emphasis',
  transforms: [
    {
      name: 'transform-typography',
      doc: 'An example transform that rewrites bold text as text with emphasis.',
      stage: 'document',
      plugin: (_, utils) => (node) => {
        utils.selectAll('strong', node).forEach((strongNode) => {
          const childTextNodes = utils.selectAll('text', strongNode);
          const childText = childTextNodes.map((child) => child.value).join('');
          if (childText === 'special bold text') {
            strongNode['type'] = 'span';
            strongNode['style'] = {
              background: '-webkit-linear-gradient(20deg, #09009f, #E743D9)',
              '-webkit-background-clip': 'text',
              '-webkit-text-fill-color': 'transparent',
            };
          }
        });
      },
    },
  ],
};

export default plugin;
```

7.3. Example myst.xref.json

Here's an example of references in a `myst.xref.json` file from the MyST Guide, note both page references and "identifier" references (e.g., labels attached to a figure or header):

```
references:
{ kind: "page", data: "/index.json", url: "/" }
{ identifier: "cool-myst-features", kind: "heading", data: "/index.json", … }
{ identifier: "quickstart-tutorials", kind: "heading", data: "/index.json", … }
{ identifier: "cite-mystmd", kind: "heading", data: "/index.json", … }
{ identifier: "project-goals", kind: "heading", data: "/index.json", … }
{ kind: "page", data: "/installing.json", url: "/installing" }
{ identifier: "installing-myst-tabs", kind: "tabSet", data: "/installing.json", … }
```

7.4. Composable Configuration

MyST configuration files can be composed with one another using the `extends:` keyword. This allows configuration to be split across multiple files, or even downloaded from a remote source via the web. For example, an author might put their author affiliation information in a dedicated `authors.yml` file:

```yaml
version: 1
project:
  contributors:
    - id: person_a
      name: First Last
      email: person_a@org_a.org
      orcid: XXXX-XXXX-XXXX-XXXX
      github: person_a
      affiliations:
        - id: org_a
        - id: org_b
    - id: person_b
      name: First Last
      email: person_b@org_b.com
      orcid: XXXX-XXXX-XXXX-XXXX
      github: person_b
      affiliations:
        - id: org_b
  affiliations:
    - id: org_a
      name: Organization A Incorporated
    - id: org_b
      name: Org B Inc.
```

And then re-use this affiliation information across multiple Jupyter Books or MyST Projects by "extending" the configuration like so:

```yaml
version: 1
extends:
  # Our local authors file
  - authors.yml
  # A remote configuration file
  - https://raw.githubusercontent.com/myorg/myrepo/refs/heads/main/funding.yaml
# Any other MyST configuration here
site: ...
project: ...
```

By allowing configuration to be split across multiple files and re-used easily, communities can reduce the duplication and outdated content associated with having a copy of the same information in multiple places. This is particularly useful for communities that need centralized databases of community-wide information (like author information) that they wish to re-use in multiple places. It's also useful to standardize community-specific configuration like branding or links in the site navigation bar.

7.5. Authoring in computational environments with `jupyterlab-myst`

A major motivation for rebuilding Jupyter Book on the MyST ecosystem is to enable a truly **web-native runtime model**. The MyST stack is built in JavaScript, and leverages the `unist` syntax tree ecosystem, a specification for ASTs used in content transformation pipelines, making it directly compatible with modern browser environments and frontend frameworks like React, as well as a host of other plugins and transformations (e.g., `unified-latex`). This also allows us to develop MyST-native authoring applications, such as `jupyterlab-myst`, a plugin that brings MyST rendering capabilities directly into JupyterLab.

With `jupyterlab-myst`, authors can preview how their markdown content elements will appear in the rendered book—including directives, equations, citations, and even code outputs—without leaving the notebook environment. This tight feedback loop reduces friction during authoring, while preserving a single source of truth between exploratory notebooks and published outputs. The plugin is part of our vision to unify **authoring, execution, and publishing** in one environment. Rather than using separate tools for writing, developing, and rendering, the MyST ecosystem will allow interfaces like JupyterLab as a

single full-featured writing and publishing environment. This lowers the barrier to creating high-quality computational narratives, especially for teams already working with Jupyter tools for their day-to-day research and teaching.

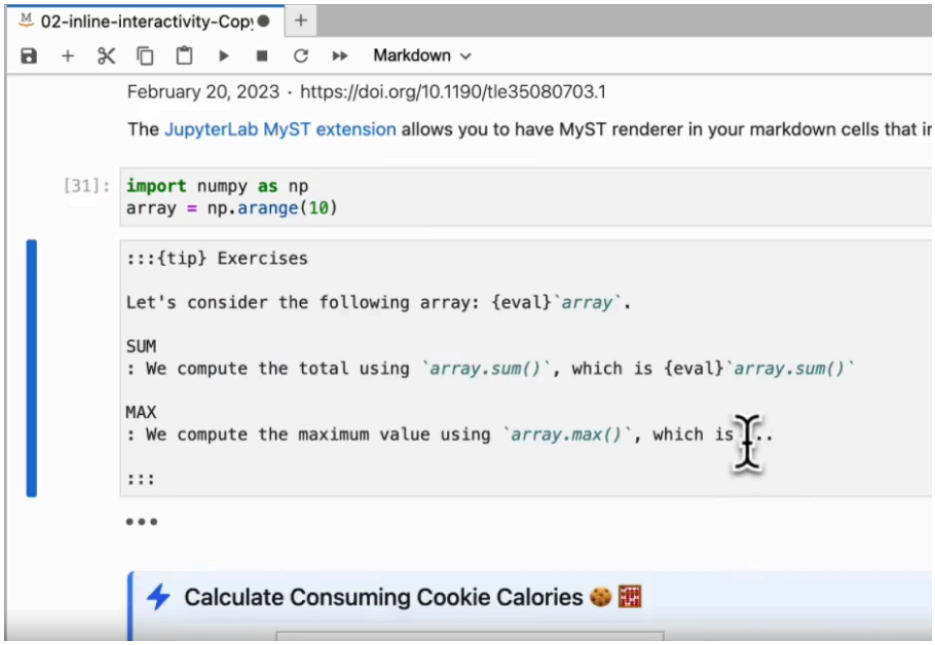


Figure 7. *Using `jupyterlab-myst` to edit MyST content with embedded widgets and interactivity in the markdown cells.*

Acknowledgements

Jupyter Book has been developed over the course of nearly a decade, with contributions from dozens of individuals and organizations. We are grateful to the many collaborators, maintainers, and community members who have helped shape the project's evolution - from its early days as a Jekyll-based tool for compiling notebooks into a textbook, through the Sphinx-based Jupyter Book 1 release, and now with the MyST Markdown and MyST Document Engine-based Jupyter Book 2 architecture. This article did not intend to capture those wide-ranging contributions and ideas or fully document the history of the project, but instead focused on the unique contributions of the Jupyter Book 2 stack.

This project has benefited from key financial support from multiple organizations. The development of MyST Markdown, the Executable Books ecosystem, and Jupyter Book 1 was supported by a grant from the Alfred P. Sloan Foundation (Grant #9231). The development of the MyST Document Engine and the re-architecture of Jupyter Book 2 were made possible in-part by funding from the Alfred P. Sloan Foundation (Grant #9231), Alberta Innovates, The Stanford Doerr School of Sustainability, the Chan Zuckerberg Initiative (2i2c, link), and The Navigation Fund (2i2c, [link] (`https://doi.org/10.71707/r3sy-7371`)). It also included in-kind support from Curvenote, 2i2c, and the broader open-source community. The refactoring of the QIIME 2 Framework documentation ecosystem was supported in part by the NIH National Cancer Institute (Grant 1U24CA248454-01). Project Pythia's transition to Jupyter Book 2 was supported by the National Science Foundation (awards 2324302, 2324303 and 2324304). Notebooks Now! was supported by a grant from the Alfred P. Sloan Foundation (#19361).

We especially acknowledge Chris Sewell, whose foundational work on the MyST Markdown specification, the Sphinx parser (`myst-parser`), and the wider Jupyter Book 1 ecosystem in

Sphinx. Chris's attention to these technical underpinnings laid the groundwork for many of the capabilities now realized in Jupyter Book 2. In addition, we acknowledge John Stachurski for providing key leadership and securing the original funding that created the Jupyter Book 1 ecosystem built on Sphinx.

We also acknowledge the work of the Executable Books team, the contributors to Jupyter Frontends sub-project, Thebe, and the maintainers of MyST-related tooling and plugins. The ongoing development of Jupyter Book 2 is supported and stewarded by the Jupyter Book Community, and is now a subproject of Project Jupyter.

Authors are in alphabetical order after "Project Jupyter" following Project Jupyter's approved process for authoring Jupyter related academic papers and aligned with our shared values of inclusivity and generosity in authorship, following clear, explicit criteria for authorship, openness, and personal accountability.

References

[1] Executable Books Community, "Jupyter Book." [Online]. Available: https://zenodo.org/record/2561065

[2] B. E. Granger and F. Perez, "Jupyter: Thinking and Storytelling With Code and Data," *Computing in Science & Engineering*, vol. 23, no. 2, pp. 7–14, 2021, doi: 10.1109/mcse.2021.3059263.

[3] K. Thomas *et al.*, "Jupyter Notebooks – a publishing format for reproducible computational workflows," in *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, IOS Press, 2016. doi: 10.3233/978-1-61499-649-1-87.

[4] R. Cockett, S. Purves, F. Koch, and M. Morrison, "Continuous Tools for Scientific Publishing: Using MyST Markdown and Curvenote to encourage continuous science practices," in *Proceedings of the 23rd Python in Science Conference*, in SciPy. 2024, pp. 121–136. doi: 10.25080/nkvc9349.

[5] Fernando Pérez, "AGU Notebooks Now! Evaluation.." [Online]. Available: https://zenodo.org/doi/10.5281/zenodo.15061830

[6] R. Cockett, L. J. Heagy, and D. W. Oldenburg, "Pixels and their neighbors: Finite volume," *The Leading Edge*, vol. 35, no. 8, pp. 703–706, 2016, doi: 10.1190/tle35080703.1.

[7] R. Cockett, C. Babott, and S. Purves, "Continuous Science Foundation Workshop — Banff 2025: Shaping the future of scientific communication through Stories, Values, and Movement from Banff," 2025, doi: 10.62329/hytv4378.

[8] P. Jupyter *et al.*, "Binder 2.0 - Reproducible, interactive, sharable environments for science at scale," in *Proceedings of the 17th Python in Science Conference*, in SciPy. 2018, pp. 113–120. doi: 10.25080/majora-4af1f417-011.

[9] The Turing Way Community, "The Turing Way: A handbook for reproducible, ethical and collaborative research." [Online]. Available: https://zenodo.org/doi/10.5281/zenodo.3233853

[10] B. E. J. Rose *et al.*, "Collaborative Research: GEO OSE TRACK 2: Project Pythia and Pangeo: Building an inclusive geoscience community through accessible, reusable, and reproducible workflows," 2023, doi: 10.5281/ZENODO.8184298.

Open Access | https://doi.org/10.25080/jcuj3732

SciPy 2025

July 7 - July 13, 2025

Proceedings of the 24th Python in Science Conference
ISSN: 2575-9752

Eyes in the Sky: Estimating Inland Water Quality Using Landsat Data

Kedar Dabhadkar[1]

[1]Independent Researcher

Abstract

Algal blooms threaten human health and aquatic ecosystems, making monitoring essential. While Chlorophyll-A (Chl-a) effectively indicates algal presence, laboratory analysis is complex. This study utilizes satellite imagery as an alternative, addressing previous research limitations caused by scarce lab data. Moreover, it also demonstrates how openly available Chl-a measurements obtained from the Water Quality Portal (WQP) can enable communities and organizations of all sizes to measure Chl-a in their waters without access to specialized in-situ water sampling skills or laboratory analysis equipment.

By combining the extensive WQP dataset with Landsat satellite imagery, these models estimate Chl-a levels in New York's inland waters. Training with eight years of data demonstrated a strong correlation between satellite-derived and actual measurements (MAPE: 0.96%; RMSE: 3.2 μg/L), enabling improved spatial and temporal monitoring capabilities.

Keywords water quality, landsat, remote sensing, earth observation

1. Introduction

Harmful algal blooms (HABs) can severely impact both the environment and human health. Environmentally, HABs deplete oxygen in water bodies, leading to fish kills and loss of aquatic biodiversity. They can block sunlight, disrupting aquatic plant growth and altering food webs. Many HABs produce toxins that accumulate in the ecosystem, affecting wildlife and domestic animals. For humans, exposure to these toxins—through drinking water, recreation, or consumption of contaminated fish and shellfish—can cause a range of health issues, including skin irritation, respiratory problems, gastrointestinal illness, and, in severe cases, liver or neurological damage. HABs also threaten water supplies and recreational activities, resulting in economic losses for affected communities. Therefore, detecting and mitigating HABs is an area of continued attention.

A key indicator of HABs is the concentration of Chlorophyll a (Chl-a) in water (J. Karmakar *et al.* [1], N. Pahlevan *et al.* [2], P. J. Sahwell, D. Bejar, D. M. Kim, and H. M. Solo-Gabriele [3]). Chl-a is a pigment found in all photosynthetic algae, and its presence is directly correlated with the abundance of algal biomass. Elevated Chl-a levels typically signal increased algal growth, which can indicate the onset or presence of a bloom. Therefore, monitoring Chl-a concentrations provides a reliable proxy for detecting and quantifying HABs in aquatic environments.

Traditionally, Chl-a is detected by in-situ water samples from sources of drinking water. In-situ sampling must be done by experts trained in methods of sample collection and analyzed by professional chemists in the lab with specialized equipment. This makes the process costly and time-consuming. Satellite remotely sensed multispectral or hyperspectral imagery can be used to build statistical models that can predict the amount of Chl-a in sources of drinking water. This reduces the reliance on in-situ sampling and synthesis, but

Published Jul 10, 2025

Correspondence to
Kedar Dabhadkar
kdabhadk@gmail.com

Open Access

Copyright © 2025 Dabhadkar. This is an open-access article distributed under the terms of the Creative Commons Attribution 4.0 International license, which enables reusers to distribute, remix, adapt, and build upon the material in any medium or format, so long as attribution is given to the creator.

some sample collection and analysis are still required to train these models. Individuals and organizations like smaller communities without access to specialized skills and equipment cannot easily access water quality data.

M. R. V. Ross *et al.* [4] first proposed using the Water Quality Portal (WQP) (E. K. Read *et al.* [5]), where organizations can submit surveys of water quality results in the US, along with remotely sensed data, to build water quality measurement models. This study extends Aquasat's approach by using a bigger timeline and focused geometry using the data from the inland waters of New York State. With more than 10,000 inland water bodies, of which nearly 8000 are lakes, ponds and reservoirs, New York is one of the richest states in terms of freshwater supply (Figure 1). Moreover, restricting the geography to a single state allows us to build models that are more specific to the region and its water bodies since other studies have shown that water quality models are not easily transferable across geographies (N. Pahlevan *et al.* [2], S. N. Topp, T. M. Pavelsky, E. H. Stanley, X. Yang, C. G. Griffin, and M. R. V. Ross [6]).

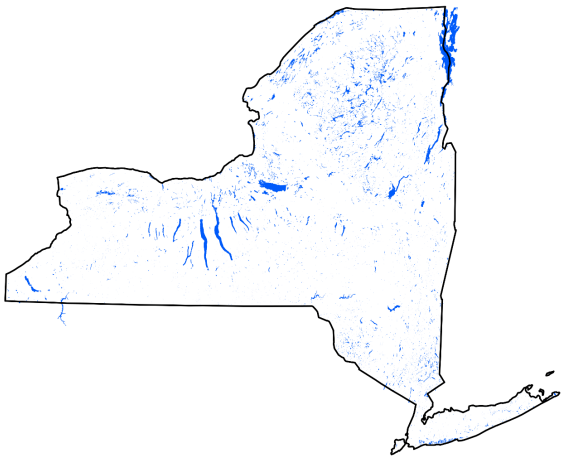


Figure 1. *Map of New York State showing all inland water bodies. There are nearly 10,000 inland water bodies in the state of New York, including lakes, ponds, swamps, and reservoirs.*

2. Data sources

2.1. The Water Quality Portal

The Water Quality Portal (WQP) E. K. Read *et al.* [5] is a collaborative data service developed by the United States Geological Survey (USGS), the Environmental Protection Agency (EPA), and the National Water Quality Monitoring Council (NWQMC). Launched in 2012, the WQP was created to provide a single access point for water quality data collected by federal, state, tribal, and local agencies, as well as non-governmental organizations across the United States. This data is collected using in-situ measurements with probes and lab analytical methods.

The portal aggregates millions of water quality records, including chemical, physical, and biological measurements from rivers, lakes, streams, and other water bodies. By standardiz-

ing and integrating data from multiple sources, the WQP enables researchers, policymakers, and the public to easily search, download, and analyze water quality information. This centralized resource supports environmental monitoring, regulatory compliance, scientific research, and informed decision-making related to water resource management. Due to the complexity of data collection and manual steps involved in in-situ sample collection, this data is very infrequent and sometimes unreliable.

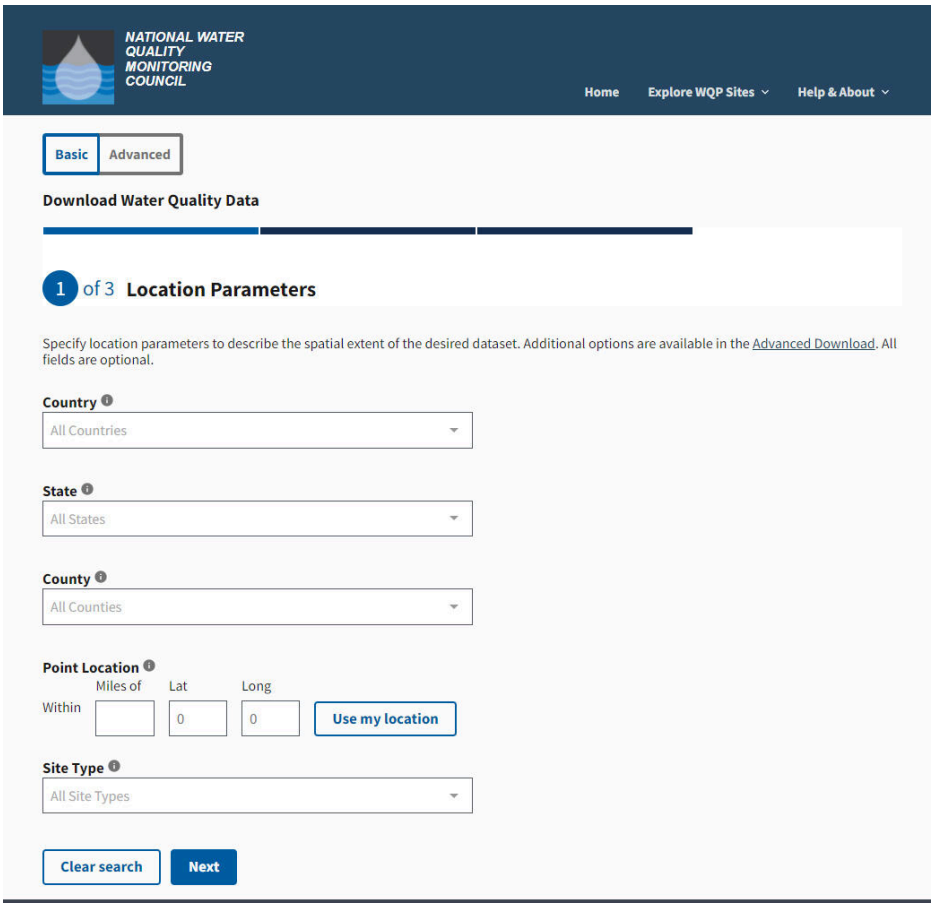


Figure 2. *Water Quality Portal. https://www.waterqualitydata.us/*

2.2. Landsat missions

The Landsat program is a series of Earth-observing satellite missions jointly managed by NASA and the US Geological Survey (USGS). Since its inception in 1972, Landsat has provided the longest continuous space-based record of Earth's land surfaces. The satellites capture multispectral images at regular intervals, enabling the monitoring of environmental changes, land use, agriculture, forestry, and water resources.

Each Landsat satellite is equipped with sensors that detect reflected and emitted energy from the Earth's surface in visible, near-infrared, and shortwave infrared wavelengths. These data are invaluable for tracking changes in vegetation, surface water, urban development, and natural disasters. The Landsat archive is freely available, making it a critical resource for scientific research and environmental management worldwide. Landsat 7, 8, and 9 missions are Sun-synchronous, near-polar orbit satellites that provide near-global coverage, with a swath-width of nearly 185 km and resolution of 15-100 m. Individually, they have a repeat cycle of 16 days, but when two of these are combined, we get a repeat cycle of 8 days. This allows us to evaluate water quality at any given place on Earth once

every 8 days, subject to cloud cover. The revisit times at any given place on Earth are fairly constant.

For water quality studies, Landsat's spatial and spectral resolution allows for the detection of features such as algal blooms, sediment plumes, and changes in water color, which can be correlated with parameters like Chl-a concentration.

Figure 3. *Landsat 9, operational since Sept. 2021, has OLI-2 (visible, NIR, SWIR) and TIRS-2 (thermal) on board. Source: https://landsat.gsfc.nasa.gov/.*

3. Approach

3.1. Data curation

This work follows a sequential process of data curation inspired by M. R. V. Ross *et al.* [4], with the key objective of matching coincident Landsat data with available WQP measurements for Chl-a in the inland waters of New York State. The entire process is laid out in Figure 4.

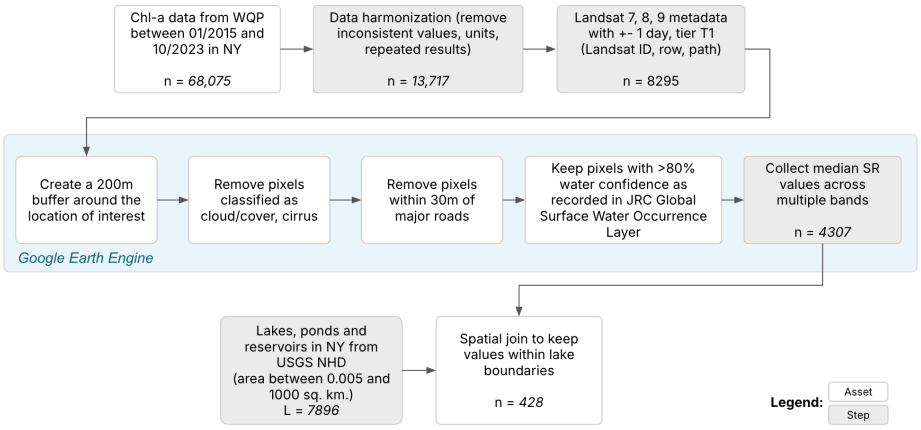


Figure 4. *A sequential data curation process involves querying data for Chl-a measurements from WQP for New York State's inland water bodies. This is followed by querying Google Earth Engine (GEE) for corresponding Landsat measurements, filtering our potential sources of noise (cloud cover and pixels around major sources of roads), and matching this data to keep measurements that fall within the major water bodies.*

Querying the WQP for Chl-a measurements within the bounds of New York State between January 2015 and October 2023 yields a dataset of 68,075 measurements. WQP is not consistent in its nomenclature and data quality checks, so it was essential to specify different ways of referring to Chl-a. Some of those ways are *Chlorophyll*, *Chlorophyll A*, *Chlorophyll a*, *Chlorophyll a (probe relative fluorescence)*, *Chlorophyll a (probe)*, *Chlorophyll a - Periphyton (attached)*, *Chlorophyll a - Phytoplankton (suspended)*, *Chlorophyll a, corrected for pheophytin*, and so on. HyRiver, a Python library published by T. Chegini, H.-Y. Li, and L. Leung [7], was used to connect to the WQP and export data.

To reduce the effect of repeated measurements from the same water body and around the same time, we deduplicate data by removing consecutive values with the same date and time if the relative standard deviation (RSD) of those values was less than 10. Passive remote sensing measurements like Landsat are not sensitive to measuring water quality beyond 100 meters depth. So, we removed all measurements that were reported to be measured beyond 100m in depth. After applying these filters and removing unclean values (where measurement values were not numerical), we are left with 13,717 measurements.

Each record in our WQP dataset is indexed by a unique combination of location (latitude and longitude) and date. For each such unique combination, we query Landsat data to find coincident measurements. We begin by querying Landsat metadata, mainly the scene ID, for each row of our Chl-a WQP dataset. We find all Landsat scenes which included the location of interest and that were imaged within ± 1 day of the WQP measurement date using publicly accessible USGS APIs. If no coincident Landsat scenes existed, we discard the measurement. This condition further reduces the number of usable measurements to 8295.

Before querying the actual Landsat data, we further filter out the unnecessary parts from each of these Landsat scenes since we only care about the part that images the area of the water body where the WQP measurement was recorded. We use Google Earth Engine (N. Gorelick, M. Hancher, M. Dixon, S. Ilyushchenko, D. Thau, and R. Moore [8]) to write a data query and pull Landsat imaged data into a tabular featurized form. To do that, we start by creating a 200-meter buffer around the point of interest (WQP measurement location) and removing the region that falls outside this buffer. Next, we remove all bits classified as cloud, cloud shadow, or cirrus. We also remove all pixels within 30 meters of major national roads and rail routes (TIGER Roads data). To ensure we do not inadvertently consider any land pixels in our data, we remove all pixels corresponding to geographies not marked as water with a confidence value of 80% or more in J.-F. Pekel, A. Cottam, N. Gorelick, and A. S. Belward [9]. 4307 measurements remain after applying all these filters.

Finally, we do a spatial join with the USGS NHD dataset to keep measurements corresponding to lakes, ponds, and reservoirs in NY state with an area of more than 0.005 sq. km but less than 1000 sq. km. That leaves us with 428 readings. Each row in this tabular dataset indicates one reading at a unique location and day. The feature values are surface reflectance from different Landsat bandwidths—Blue, Green, Red, Nir, Swir1, and Swir2.

3.2. Statistical modeling

The cleaned dataset of 428 readings was used to train different statistical models with regression—random forest, gradient boosting, Lasso, AdaBoost, and XGBoost. The best model was chosen based on MAE and RMSE scores in a 10-fold cross-validation experiment. The statistical modeling workflow is shown in Figure 5.

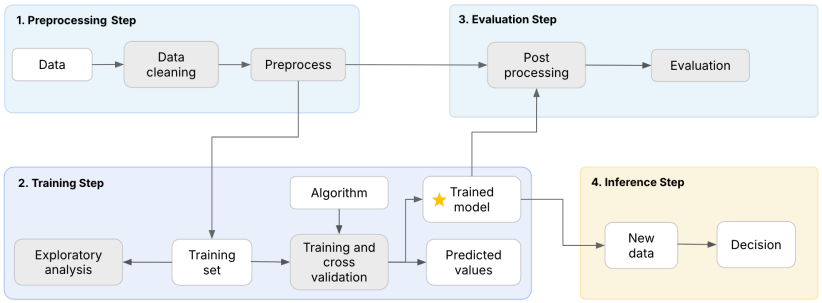


Figure 5. *Training and cross-validation experiment setup*

4. Results

4.1. Model selection

Based on 10-fold cross-validation results for each algorithm, we find the random forest model performed the best (Table 1) with 500 decision tree estimators. Model algorithm and optimization setup available as part of the scikit-learn project [10] in Python was used for model training. We choose this model for further inferencing on other Landsat water body scenes with no matching WQP measurements for Chl-a. Figure 6 shows a comparison of the values predicted by this model and the actual Chl-a measurements from WQP.

Table 1. *Cross-validation results. The regression metrics shown below are for the best set of hyperparameters for the displayed algorithm.*

Algorithm	RMSE (μg/L)	MAE (μg/L)	R2 (adj.)	MAPE (%)
Random forest	**3.18**	**2.22**	**0.90**	**0.93**
Gradient boosting	8.46	6.46	0.29	2.86
Lasso	9.98	7.66	0.013	3.31
AdaBoost	8.28	6.51	0.329	3.072
XGBoost	3.990	2.819	0.842	1.173

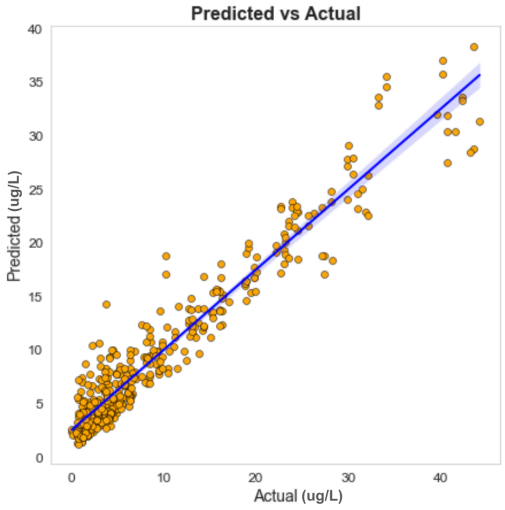


Figure 6. *Values of Chl-a predicted by the best performance model plotted against actual values.*

By training models on in-situ WQP data and applying them to Landsat satellite imagery, we can estimate Chlorophyll-a concentrations for over 30 times more locations and days than those directly measured in the WQP dataset. This expanded coverage is shown in Figure 7.

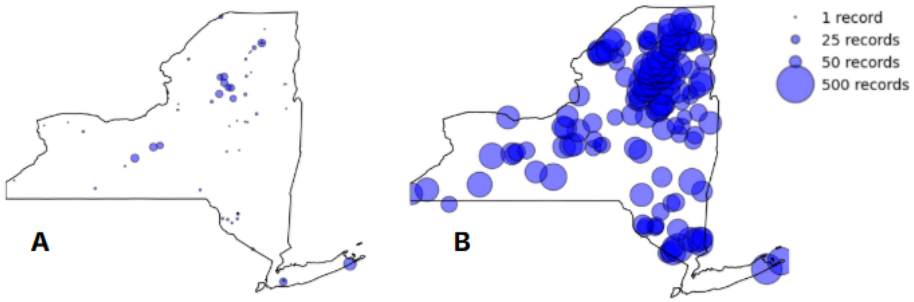


Figure 7. *Panel A shows the number of in-situ Chl-a measurements from the Water Quality Portal across New York's 200 largest lakes between 2015 and 2023. Panel B demonstrates that using remote sensing, we can estimate Chl-a concentrations at over 30 times more locations and dates within the same lakes and time period. Each bubble represents the center of water body and the size of the bubble is proportional to the number of measurements.*

5. Conclusion

This study demonstrates that combining Landsat satellite imagery with in-situ water quality measurements enables accurate estimation of Chl-a concentration in New York's inland waters. The curated dataset and rigorous filtering process ensured high-quality training data, and the random forest regression model achieved strong predictive performance. These results highlight the potential of remote sensing and machine learning to supplement traditional water quality monitoring, offering improved spatial and temporal coverage. Future work could expand this approach to other regions and water quality indicators, further supporting environmental management and public health efforts.

References

[1] J. Karmakar *et al.*, "Analyzing spatio-temporal variability of aquatic productive components in Northern Bay of Bengal using advanced machine learning models," *Ocean & Coastal Management*, vol. 251, p. 107074, 2024, doi: 10.1016/j.ocecoaman.2024.107074.

[2] N. Pahlevan *et al.*, "Seamless retrievals of chlorophyll-a from Sentinel-2 (MSI) and Sentinel-3 (OLCI) in inland and coastal waters: A machine-learning approach," *Remote Sensing of Environment*, vol. 240, p. 111604, 2020, doi: 10.1016/j.rse.2019.111604.

[3] P. J. Sahwell, D. Bejar, D. M. Kim, and H. M. Solo-Gabriele, "Non-traditional abiotic drivers explain variability of chlorophyll-a in a shallow estuarine embayment," *Science of The Total Environment*, vol. 919, p. 170873, 2024, doi: 10.1016/j.scitotenv.2024.170873.

[4] M. R. V. Ross *et al.*, "AquaSat: A Data Set to Enable Remote Sensing of Water Quality for Inland Waters," *Water Resources Research*, vol. 55, no. 11, pp. 10012–10025, 2019, doi: 10.1029/2019wr024883.

[5] E. K. Read *et al.*, "Water quality data for nationalscale aquatic research: The Water Quality Portal," *Water Resources Research*, vol. 53, no. 2, pp. 1735–1745, 2017, doi: 10.1002/2016wr019993.

[6] S. N. Topp, T. M. Pavelsky, E. H. Stanley, X. Yang, C. G. Griffin, and M. R. V. Ross, "Multi-decadal improvement in US Lake water clarity," *Environmental Research Letters*, vol. 16, no. 5, p. 55025, 2021, doi: 10.1088/1748-9326/abf002.

[7] T. Chegini, H.-Y. Li, and L. Leung, "HyRiver: Hydroclimate Data Retriever," *Journal of Open Source Software*, vol. 6, no. 66, p. 3175, 2021, doi: 10.21105/joss.03175.

[8] N. Gorelick, M. Hancher, M. Dixon, S. Ilyushchenko, D. Thau, and R. Moore, "Google Earth Engine: Planetary-scale geospatial analysis for everyone," *Remote Sensing of Environment*, vol. 202, pp. 18–27, 2017, doi: 10.1016/j.rse.2017.06.031.

[9] J.-F. Pekel, A. Cottam, N. Gorelick, and A. S. Belward, “High-resolution mapping of global surface water and its long-term changes,” *Nature*, vol. 540, no. 7633, pp. 418–422, 2016, doi: 10.1038/nature20584.

[10] F. Pedregosa *et al.*, “Scikit-learn: Machine Learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011, [Online]. Available: http://jmlr.org/papers/v12/pedregosa11a.html

Open Access | https://doi.org/10.25080/dvta2583

SciPy 2025

July 7 - July 13, 2025

Proceedings of the 24th Python in Science Conference
ISSN: 2575-9752

Quantum Chemistry Acceleration: Comparative Performance Analysis of Modern DFT Implementations

Kyohei Sahara[1]

[1]Kyoto University

Abstract

This article examines the acceleration of quantum chemistry calculations through modern implementations of Density Functional Theory (DFT). We provide a comparative performance analysis between traditional frameworks and advanced implementations, demonstrating computational efficiency gains. Applications in electrolyte membrane structure analysis illustrate the practical benefits of these performance improvements. Our benchmarking results show how transitioning from traditional implementations to optimized frameworks enables more extensive quantum chemistry simulations within reduced timeframes, with substantial speedups obtained through modern code optimization techniques in Python-based quantum chemistry environments.

Keywords quantum chemistry, density functional theory, performance analysis, computational chemistry

1. Introduction

Quantum chemistry calculations have become indispensable tools for understanding molecular properties, reaction mechanisms, and material behavior at the atomic scale. Density Functional Theory (DFT) has emerged as one of the most widely used quantum mechanical methods due to its balance between computational efficiency and chemical accuracy. However, as research problems become increasingly complex and systems grow larger, the computational demands of traditional DFT implementations often become prohibitive. Modern scientific computing has witnessed advances in optimization techniques, parallel processing, and algorithmic improvements that directly benefit quantum chemistry calculations. The Python ecosystem, in particular, has seen growth in high-performance scientific libraries that enable researchers to leverage these advances more effectively.

This article presents a performance analysis comparing traditional quantum chemistry frameworks with modern optimized implementations. The development of environmentally-friendly materials for electrochemical applications is crucial for sustainable energy storage and electrolysis technologies. This study focuses on chlorinated analogues of phosphoric acid compounds ($CCl_3CCl_2PO_3H$) as alternatives to fluorinated materials for use in electrochemical systems, aiming to reduce environmental impact while maintaining proton conductivity properties for electrolysis processes. Each test system contains 38 atoms with varying water molecule distributions to simulate realistic chemical environments. Our analysis demonstrates substantial computational speedups obtained through modern code optimization strategies while maintaining chemical accuracy.

Published Jul 10, 2025

Correspondence to
Kyohei Sahara
schwalbe10@gmail.com

Open Access

Copyright © 2025 Sahara. This is an open-access article distributed under the terms of the Creative Commons Attribution 4.0 International license, which enables reusers to distribute, remix, adapt, and build upon the material in any medium or format, so long as attribution is given to the creator.

2. Methods

2.1. Target System

Understanding proton exchange membrane structures represents a particularly challenging application for quantum chemistry calculations. The structural analysis involves complex multi-component systems including polymer chains, ionic species, and interfacial interactions. Traditional DFT implementations require substantial computational resources to adequately characterize these structures.

For example, studying environmentally-friendly phosphoric acid structures based on chlorinated analogues of $CF_3CF_2PO_3H$ [1], specifically $CCl_3CCl_2PO_3H$ compounds with water molecules, requires calculations on systems containing 38 atoms total with multiple conformational states. The fluorine-to-chlorine substitution aims to develop non-fluorinated materials for environmental sustainability. To investigate proton conductivity properties, several H_2O molecules are included in the computational model to simulate realistic proton-conducting environments. Using conventional frameworks like SIESTA [2], geometry optimization requires substantial computational time on modern computational clusters, making comprehensive structural studies computationally expensive.

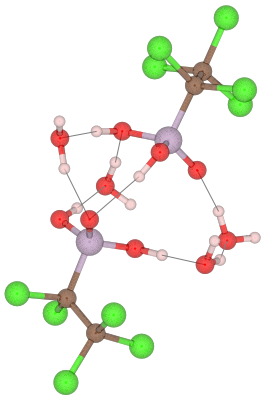


Figure 1. *Chemical structure of the $CCl_3CCl_2PO_3H$ compound with H_2O molecules studied in this work, showing the chlorinated analogue design for environmental sustainability and proton conduction pathway investigation.*

2.2. Computational Frameworks

Traditional quantum chemistry frameworks have provided the foundation for decades of molecular modeling research. Packages such as SIESTA have been widely adopted due to their robust implementations and extensive feature sets. However, traditional implementations like SIESTA often present computational bottlenecks when applied to complex systems requiring extensive parameter space exploration. SIESTA remains valuable for specific applications requiring its specialized basis sets, pseudopotentials, or particular exchange-correlation functionals, but may be less optimal for high-throughput studies.

Modern Python-based quantum chemistry environments offer computational advantages through optimized algorithms, efficient memory management, and leveraging of high-performance numerical libraries. Frameworks such as PySCF [3] and its GPU-accelerated extension GPU4PySCF [4] have emerged as powerful alternatives that can dramatically reduce computation times while maintaining chemical accuracy. PySCF achieves performance improvements through efficient integral computation algorithms and optimized parallel processing using OpenMP and MPI implementations, while GPU4PySCF further

accelerates calculations by leveraging GPU parallelization for computationally intensive operations like two-electron integrals and matrix operations.

3. Experimental Conditions

Our benchmarking study compares computational performance across SIESTA, PySCF, and GPU4PySCF implementations using identical test systems. The benchmark suite consists of 85 individual input files, each being a 38-atom system containing a chlorinated phosphoric acid structure ($CCl_3CCl_2PO_3H_2$) with water molecules. These configurations feature varying water molecule distributions around the phosphoric acid structure to represent different water-membrane configurations, providing realistic system sizes for environmentally-friendly materials design with statistical reliability.

All calculations were performed using identical computational setups to ensure fair comparison:

- **System**: 38-atom systems containing 2 $CCl_3CCl_2PO_3H_2$ structures with 4 H_2O molecules
- **Exchange-correlation functional**: RPBE
- **Basis set**: DZP (Double-Zeta Polarized)
- **Convergence criteria**: SCF tolerance of 10^{-6} for all calculations
- **Computational environment**:
  - SIESTA (CPU): 16 parallel jobs (2 cores per job)
  - PySCF (CPU): 16 parallel jobs (2 cores per job)
  - GPU4PySCF (GPU): 16 parallel jobs
- **Hardware**:
  - CPU: Intel Xeon processors, 32 vCPUs, 256 GiB memory (AWS r7i.8xlarge, $2.117/hr)
  - GPU: 8x NVIDIA A100 80GB SXM4, 240 vCPUs, 1800 GiB RAM (Lambda, $14.32/hr)
- **Performance metrics**: Execution times and costs calculated based on cumulative processing time for all 85 configurations divided by parallelization factor (16) to reflect actual wall-clock time and resource usage

4. Results and Discussion

4.1. Performance Comparison

Figure 2 shows the performance differences between the three implementations. SIESTA represents the traditional approach with longer execution times, while PySCF provides acceleration through modern optimization techniques. GPU4PySCF shows the best performance by leveraging GPU acceleration, showing improvement in execution time while maintaining lower computational costs. These performance differences enable researchers to conduct more extensive conformational sampling and statistical analysis within available computational resources.

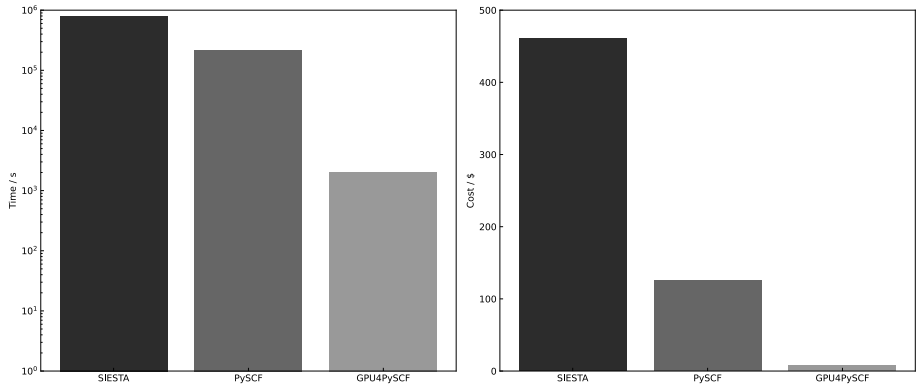


Figure 2. *Computational performance comparison between SIESTA, PySCF, and GPU4PySCF for 38-atom systems containing phosphoric acid structures, showing both execution time (left) and computational cost (right).*

4.2. Impact on Research Methodology

The performance improvements demonstrated in this study have broader implications beyond phosphoric acid structure analysis. The 3.7× speedup observed with PySCF and 390× speedup with GPU4PySCF can be expected to apply to similar molecule systems with comparable atom counts and electronic structure characteristics.

These computational advantages enable:

- **Expanded parameter space exploration**: Systematic investigation of molecular modifications, conformational changes, and environmental conditions across diverse chemical systems through sampling of structural configurations
- **Enhanced statistical reliability**: Larger sample sizes enabling more robust conclusions and confident predictions in materials design studies through increased sampling of molecular configurations and property distributions
- **Integration with modern workflows**: Seamless compatibility with machine learning approaches and automated high-throughput computational screening pipelines, facilitating development of predictive models for materials discovery

The methodology established here provides a foundation for accelerating quantum chemistry calculations across various molecular systems, particularly those involving compounds with heteroatoms (atoms other than carbon and hydrogen) and solvation effects (interactions between solutes and surrounding solvent molecules).

5. Conclusions

This study demonstrates computational advantages of modern PySCF implementations over traditional SIESTA for DFT calculations.

Key findings include:

1. **Performance improvement**: PySCF shows 3.7× speedup and GPU4PySCF demonstrates 390× speedup compared to SIESTA while maintaining chemical accuracy
2. **Cost efficiency**: Computational costs reduced from $460.50 (SIESTA) to $125.35 (PySCF) and $7.99 (GPU4PySCF), representing 73% and 98% cost reductions respectively

3. **Broader applicability**: The methodology extends to similar molecular systems, enabling systematic parameter exploration and integration with modern computational workflows

These results provide a foundation for accelerating quantum chemistry calculations across various molecular systems. The demonstrated efficiency gains illustrate the potential for making quantum chemistry calculations more accessible for complex materials science applications.

6. Code Availability

The computational scripts, input files, and data processing code used in this study are available at:

https://github.com/schwalbe10/quantum-chemistry-acceleration

The repository includes:

- SIESTA, PySCF and GPU4PySCF calculation scripts
- 85 input configurations for phosphoric acid structures
- Data processing and visualization code
- Detailed instructions for reproducing the benchmark results

References

[1] T. Ogawa, H. Ohashi, G. M. Anilkumar, T. Tamaki, and T. Yamaguchi, "Suitable Acid Groups and Density in Electrolytes to Facilitate Proton Conduction," *Physical Chemistry Chemical Physics*, vol. 23, no. 41, pp. 23778–23786, 2021, doi: 10.1039/D1CP00718A.

[2] J. M. Soler *et al.*, "The SIESTA Method for Ab Initio Order-N Materials Simulation," *Journal of Physics: Condensed Matter*, vol. 14, no. 11, pp. 2745–2779, 2002, doi: 10.1088/0953-8984/14/11/302.

[3] Q. Sun *et al.*, "PySCF: The Pythonbased Simulations of Chemistry Framework," *WIREs Computational Molecular Science*, vol. 8, no. 1, p. e1340, 2018, doi: 10.1002/wcms.1340.

[4] X. Wu *et al.*, "Enhancing GPU-Acceleration in the Python-Based Simulations of Chemistry Framework," *WIREs Computational Molecular Science*, 2024, doi: 10.1002/wcms.70008.

Open Access | https://doi.org/10.25080/ahrt5264

SciPy 2025

July 7 - July 13, 2025

Proceedings of the 24th Python in Science Conference
ISSN: 2575-9752

Imprecise uncertainty management with uncertain numbers to facilitate trustworthy computations

Yu Chen[1], and **Scott Ferson**[1]

[1]University of Liverpool

Abstract

Scientific computations of complex systems are surrounded by various forms of uncertainty, requiring appropriate treatment to maximise the credibility of computations. Empirical information for characterisation is often scarce, vague, conflicting and imprecise, requiring expressive uncertainty structures for trustworthy representation, aggregation and propagation. Current practices may present two undesired extremes in terms of uncertainty management, with one endpoint being the total ignorance of uncertainty, whereas the other suggesting overconfidence through the introduction of assumptions unjustified by empirical information. In response to these challenges, this paper demonstrates the framework of `uncertain number`, a unified construct for expressive uncertainty representation at different imprecision. This framework, embedded in the library `pyuncertainnumber`, allows for a closed computation ecosystem whereby trustworthy computations can be conducted intrusively or non-intrusively, hence accomplishing faithful management of uncertainty throughout the computational pipeline. This paper presents an overview of the main capabilities and features of `pyuncertainnumber`.

Keywords uncertain number, imprecise probability, probability bounds analysis, optimisation under uncertainty

1. Introduction

Large-scale scientific computing applications for complex physical and engineered systems are often approximate and uncertain, notably due to limited knowledge of the physical processes, approximate representation of the model, and sparse experimental data for calibration, etc [1], [2], [3]. Besides, engineered systems are often required to operate safely and robustly under varying environments or uncertain operational conditions, requiring uncertainty to be comprehensively considered in order to maximise the credibility of predictions, designs and decisions [4].

It is vital to *know what you do not know* in terms of trustworthy modelling and predictions, suggesting that *all models are wrong but some are useful* on the condition of knowing their assumptions and applicability, hence building the credibility of the computational results. It is often a challenge, in modelling complex physical phenomena, to construct mathematical models in a quantitative manner, on one hand, without ignoring significant information and, on the other hand, without introducing unwarranted assumptions [5], [6]. The bottleneck is usually the limited information in terms of both knowledge and experimental data.

The increasing awareness of the differentiation of aleatory and epistemic uncertainty arises the need for more expressive mathematical frameworks to reason with various forms of uncertainty [7], [8]. Imprecise uncertainty frameworks [9], such as evidence theory, random set, possibility distributions, credal set, capacities are proposed to reflect the situations

Published Jul 10, 2025

Correspondence to
Yu Chen
lesliechen666@outlook.com

Open Access

Copyright © 2025 Chen & Ferson. This is an open-access article distributed under the terms of the Creative Commons Attribution 4.0 International license, which enables reusers to distribute, remix, adapt, and build upon the material in any medium or format, so long as attribution is given to the creator.

when information is scarce, vague, conflicting or imprecise whereby precise distribution are hard to be defined. Given the available information, there exist two untenable extreme (exclusive) endpoints that prescribe an interval of trustworthiness. The lack of uncertainty quantification presented in many deterministic numerical simulations constitutes the lower bound, whereas the overconfidence of the computation, through introduction of unwarranted assumptions about uncertainties not faithful to the state of knowledge, as the upper bound.

We aim at a faithful management of uncertainty throughout the computational pipeline using `uncertain number`, a unified construct for uncertainty characterisation at different imprecision. This paper demonstrates the framework of `uncertain number` which allows for a closed computation ecosystem whereby trustworthy computations can be conducted in a rigorous manner. This paper presents an overview of the main capabilities of the library `pyuncertainnumber`[1]. To facilitate clearer illustration and reproducibility of the workflow, essential code snippets are included in the main text, while a comprehensive tutorial is available in the Supporting Documents section.

2. Expressive power of uncertain number

When characterising input parameters to a simulation model, while probability distribution has long been used to reflect uncertainty (*variability*[2]), there is still uncertainty (*incertitude*[3]) about the distribution shape and interdependencies. Empirical information varies in quality and nature, which may be quantitative, qualitative and even linguistically vague. Multiple sources or elicitation could vary in credibility and even be conflicting. Empirical data, if any, may be scarce or imprecise due to the inaccuracy of experimental measurements, or prohibitive cost of collecting data. It is therefore challenging to formulate suitable uncertainty models given partial information without introducing unwarranted assumptions. To this end, more expressive mathematical constructs are needed.

`Uncertain number` stands for a generalised representation that unifies several uncertainty constructs including intervals, probability distributions, probability boxes (p-boxes) and Dempster-Shafer structures (DSS), plus real numbers. These constructs are closely related to each other: an interval indicates a value imprecisely known within a range where no assumptions of likelihood about the enclosed values are made; P-boxes can be considered as interval bounds on cumulative distributions and a DSS can be deemed as a discrete distribution with interval quantiles. Parametric p-boxes $F_X(x \mid \theta^I)$ are probability distributions whose parameters θ^I and samples are intervals, and an interval ($I = [a, b]$) can be identified as a p-box $[H_a(x), H_b(x)]$ whose bounds are unit step functions denoted by $H(x)$; a p-box can be discretised into a DSS with pairs of intervals (focal elements) and probability masses $\left\{([a_i, b_i], p_i)_1^N\right\}$, and conversely a DSS can be stacked with a list of intervals. Importantly, all of these constructs are special cases of free p-boxes which effectively represents a set of distributions.

Figure 1 visually illustrates the notion of an `uncertain number`, which is underpinned by a probability bounding approach [10], [11], [12] that allows for a faithful representation of the state of empirical information. For example, it can be characterised as a real number (a *degenerate* of an interval) when there is no uncertainty, a precise distribution (a *degenerate*

[1] `pyuncertainnumber` is a research-grade open-source library, with its source code available on GitHub and comprehesive documentation available online.

[2] also called randomness, aleatory uncertainty, objective uncertainty, dissonance, or irreducible uncertainty arised from natural stochasticity, environmental or structural variation across space or through time.

[3] also called ignorance, epistemic uncertainty, non-specificity, or reducible uncertainty arised from incompleteness of knowledge.

of a p-box) when there is abundant data, and a set of distributions (e.g. a p-box) when there is partial information.

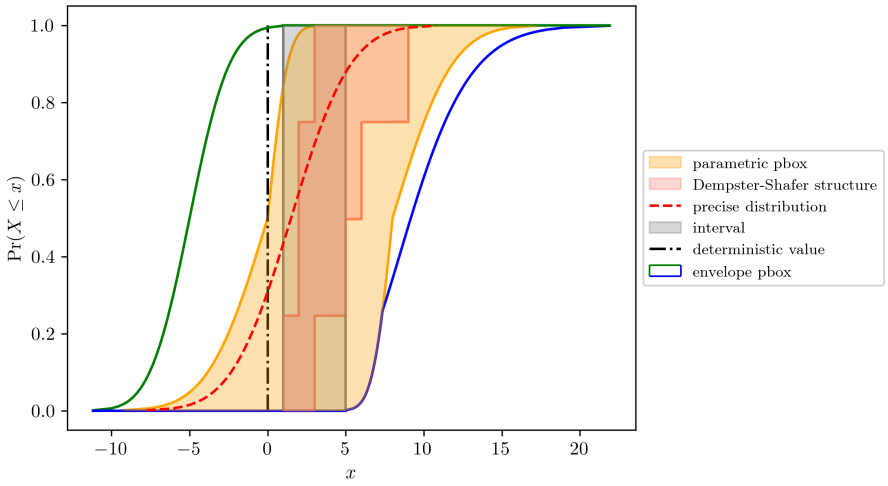


Figure 1. *Uncertain number refers to a generalised representation that unifies several constructs including intervals, probability distributions, probability boxes (p-boxes) and Dempster-Shafer structures, plus real numbers.*

2.1. A probability bounding approach on information constraints

Often there is little empirical information pertaining some parameters of a mathematical model. A faithful characterisation entails that all of the available statistical information should be utilised but without introducing any extra assumptions beyond what are empirically justified.

The bounding approach presents as a natural reflection of the state of epistemic uncertainty, which tightens the bounds with extra empirical information. As shown in Figure 2, the level of information specifies constraints to accordingly construct the uncertain number, which starts as little as a coarse estimated range, to moment information, further to the knowledge of shape such that uncertainties are pinched to a precise distribution. These bounds are best-possible[4] in the sense that they could not be any tighter without excluding CDF satisfying the specified constraints, or without additional information [7].

Conveniently, `pyuncertainnumber` provides a bespoke constructor to facilitate the faithful characterisation of uncertain quantities based on known information, which includes limits on quantiles, information about summary statistics such as mean, mode or variance, and qualitative information about distribution shape, such as whether it is symmetric or unimodal.

Figure 2. *Illustration of the idea of level of information specified as constraints*

[4] the uncertain number could not be any tighter without more information.

```
import pyuncertainnumber as pun

# specify available empirical information as constraints
pun.known_constraints(minimum=0, maximum=2., mean=1, var=0.25)
```

2.2. *Aggregation of uncertainty*

One of the controversial subject in uncertainty analysis is the aggregation of multiple (imperfect) sources of information, evidence, or expert elicitations. Information varies in quality and could be conflicting. Experts may have different degrees of subjectivity and different representations for the uncertain quantity, leading to epistemic uncertainty in the various educated estimates.

`Uncertain number` presents one advantage that it provides a unified structure to enclose the aggregation operation on a set of expert elicitations, whatever the forms they may be, whether a distribution or an interval. Figure 3 shows an example of the elicitation from a couple of experts with their credibility shown as probability masses. This explains further the provenance of the constructs in Figure 1. `pyuncertainnumber` supports several aggregation rules (e.g. envelope, mixture, and intersection, etc) enabling diverse level of information to be pooled stochastically or conservatively.

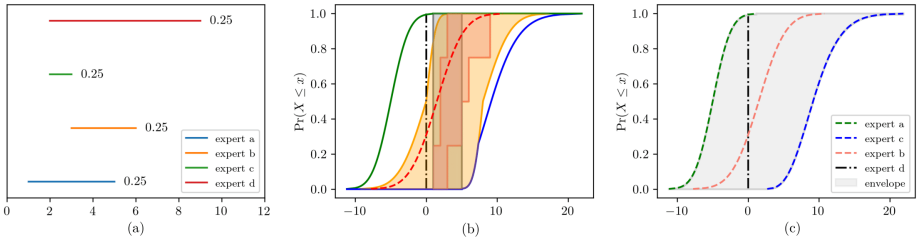


Figure 3. *Uncertainty aggregation:* ***(a)*** *expert knowledge expressed as intervals coupled with their credibility;* ***(b)*** *the same uncertain number in Figure 1 of which the DSS is composed by the aggregation in (a), and the circumscribed p-box is composed by the envelope in (c);* ***(c)*** *expert elicitation as precise distributions whereby an envelope operation is taken*

```
from pyuncertainnumber import stochastic_mixture, envelope

""" note aggregation operations apply to all constructs of uncertain numbers """

# mixture aggregation of interval elicition into a Dempster-Shafer structure
a = pun.I(1, 5)  # b, c, d ...
mix = stochastic_mixture(a, b, c, d, masses=[0.25, 0.25, 0.25, 0.25])

# envelope aggregation of distributional eclicitation into a p-box
a = pun.D('gaussian', (-5, 2))  # b, c, d ...
env = envelope(a, b, c, d)
```

2.3. *Measurement imprecision*

Empirical data rarely come in perfect forms, especially for *in situ* measurements. Practical computations frequently deal with poor measurements with different imprecision, possibly arising in recording, transmission, communication or manipulation, etc [13]. Intervals turn out to be natural constructs for representing *incertitude* in imprecise measurements, manifested either in a direct interval or a plus-or-minus form [14]. When interval-valued measurements are present in a data set, a single probability distribution is inadequate to characterise the epistemic uncertainty. Rather, the bounding strategy applies whereby classical inference methods are extended to both characterise the sampling uncertainty and also data imprecision. For example, as shown in Figure 4, a set of maximum likelihood

estimates $\lambda^I = \left[\underline{\lambda}, \overline{\lambda}\right]$ are yielded for a single datum where λ indicates the parameter of exponential distribution. Collectively, the dataset leads to an interval bound of the fitted exponential distribution shown in the shaded area in Figure 4, along with the true data generating mechanism (DGM) in red dotted curve and the empirical CDF in gray dotted curve. Further, as a nonparametric comparison, the Kolmogorov Smirnov confidence bands is also extended [15], as shown in blue dashes below. These are both rigorous uncertain numbers that enclose the true yet unknown data generating distribution.

$$\left[\underline{F}(x), \overline{F}(x)\right] = \left[\min\left(1, \hat{F}_L(x) + D_N^\alpha\right), \max\left(0, \hat{F}_R(x) - D_N^\alpha\right)\right] \tag{1}$$

where $\hat{F}_L$ and $\hat{F}_R$ represent the empirical CDF on endpoints of the imprecise dataset, and D_N^α denotes the Smirnov critical value at significance level α.

Figure 4. *Characterisation of imprecise measurements.* ***(a)*** *fitting an exponential distribution to an interval datum by maximum likelihood estimation;* ***(b)*** *Parametric and nonparamettric characterisation of the whole imprecise data set $\{x_i\}$ which includes 15 data points i.i.d (independent and identically distributed) sampled from an exponential distribution (i.e. $\mathrm{Exp}(0.4)$) contaminated by a margin of error $\Delta = 1.4$. That is, $x_i = [\underline{x}_i, \overline{x}_i] = [m_i - \Delta, m_i + \Delta]$ where $m_i \sim \mathrm{Exp}(0.4)$.*

```
import scipy.stats as sps
from pyuncertainnumber import pba

# synthetic generation of the imprecise data
precise_sample = sps.expon(scale=1/0.4).rvs(15)
imprecise_data = pba.I(lo = precise_sample - 1.4, hi=precise_sample + 1.4)

# parametric distributional estimator using method of matching moments
pun.fit('mom', family='exponential', data=imprecise_data)

# nonparametric estimator using Kolmogorov Smirnov confidence bands at 95% confidence level
pun.KS_bounds(imprecise_data, alpha=0.025, display=True)
```

2.4. *Linguistic numerical hedges for uncertainty interpretation*

Minimally, qualitative linguistic description may be used to express the estimates over numerical input values. Those are called numerical hedges, which may include colloquial words such as “about”, “around”, “almost” etc. [16]. With the focus on NLP in many deep learning applications in recent years, the interpretation of hedged words and their quantitative implication of uncertainty is vital for downstream safety-related applications. `pyuncertainnumber` provides support to interpret these hedges in an effort to build a consistent basis for consistent uncertainty elicitation and communication in situations where empirical information is almost minimal. Importantly, real numbers such as “7” or “7.0” can also be interpreted as an interval based on significant digits [16], potentially leading to a complete system of rigorous numerical numbers.

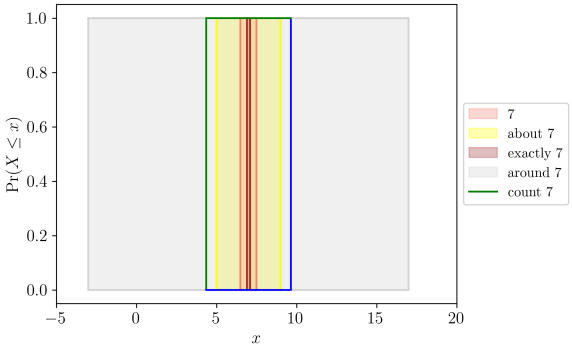


Figure 5. *Illustration of numerical hedges.*

```
pun.hedge_interpret('about 7')  # similar usage for other hedges, such as '7.0'
```

2.5. *Dependency structure: fully specified, partially known or unknown*

Neglecting the incertitude about the dependency structure and assuming independence anyway constitutes a methodological bad practice. Maybe the most notable example of misuse of dependency structure is the 2008 financial crisis [17].

Similar as the bounding approach applies onto marginal distributions in the face of various levels of information, the inter-variable dependency information can also be known, partially known or unknown within the probability bounding framework. Collectively, it manifests the imprecise specification of a joint distribution. As an example, Figure 6 illustrates the interval bounds of a joint CDF of a bivariate p-box. In this case, sampling operations will produce bivariate interval constructs, shown as the rectangles at the bottom of Figure 6. These constructs can all be consistently represented as uncertain numbers. Additional details regarding the imprecise dependency specification will be discussed in Section 6.

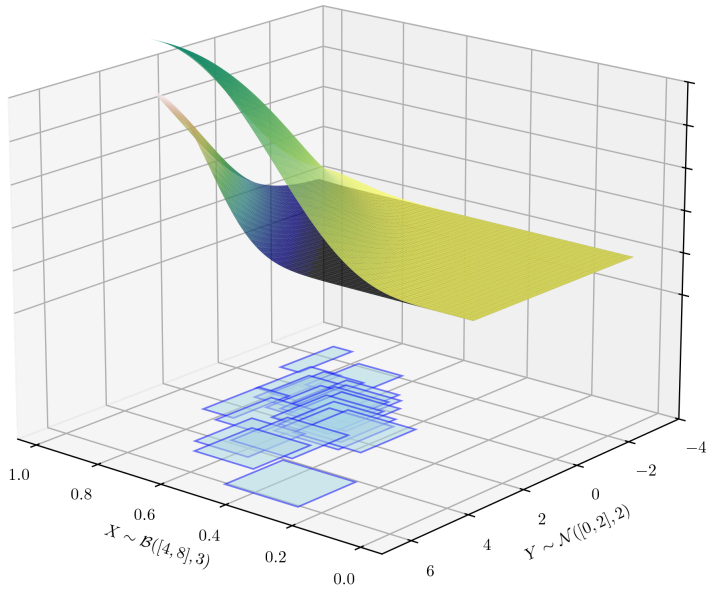


Figure 6. *A bivariate p-box with marginals $X \sim \mathcal{B}([4,8],3)$ and $Y \sim \mathcal{N}([0,2],2)$, and Gaussian copula parameterised by $\rho_{XY} = -0.8$. The upper of the plot displays the joint CDF with upper and lower surface and the bottom shows the sampled deviates which are bivariate intervals*

3. Non-deterministic computations and uncertainty propagation

Scientific computations are desired to be *rigorous* and *best-possible* when subject to uncertainties.

3.1. *Probability bounds analysis*

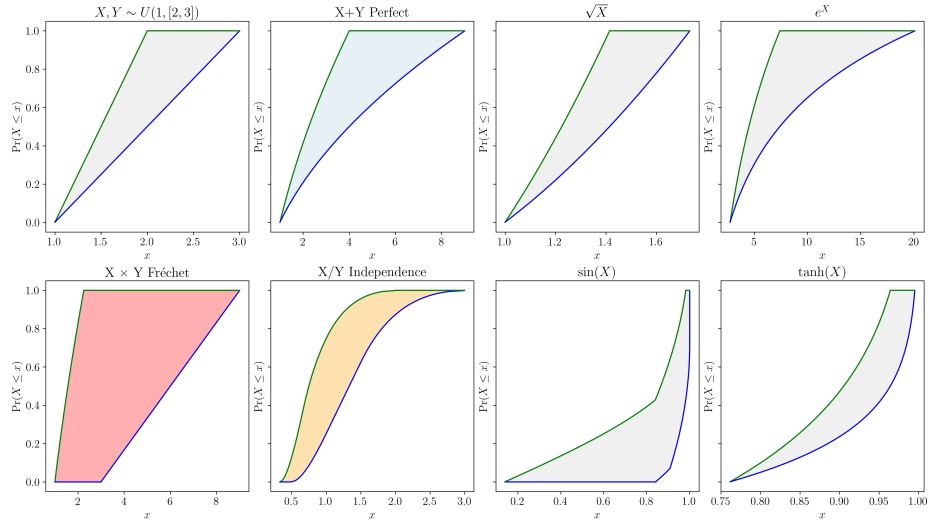


Figure 7. *Illustration of p-box arithmetic under various dependency structure. The left four show different dependency structures in colors with different binary operations, the right four show differnt unary operations.*

Probability bounds anlaysis (PBA) combines both interval analysis and probability theory, allowing rigorous bounds of (arithmetic) functions of random variables to be computed even with partial information [7], [11], [12]. Intuitively, as interval arithmetic enables rigorous calculation for sets of real numbers, PBA accomplish the same for sets of distributions. P-box arithmetic is built upon generalised probability convolutions, which through further extensions cover a wide spectrum of arithmetic operations: unary transformations, binary operations between p-boxes or Dempster-Shafer structures, and general functions composed of a series of base operations. It also covers a wide spectrum of dependency structures which could be fully known (specified copula C), partially known (lower bound copula $\underline{C}$), or even unknown (Fr ' e chet bounds [18]).

Importantly, we provide a unified interface and implementation through `pyuncertainnumber`, enabling rigorous calculations regardless of the representation of the uncertain variables or their dependency structure. This interface has abstracted out much of the arcane details in terms of conversion, discretisation, condensation, and stacking such that analysts can benefit the consistency and focus more on the effects the calculations.

Notably, the Fr ' e chet bounds provides a trustworthy default when no dependency information is known, in that the resulting bounds rigorously enclose the result no matter what correlation or nonlinear dependency may exist between these variables. This faithfully translates the situation of unknown into making no assumption at all, as opposed of an unjustified independence assumption. Equation (2) gives the Fr ' e chet expression [18] for the addition of uncertain numbers and multiplication and division have similar forms, where $\left[\overline{F}_X(x), \underline{F}_X(x)\right]$ denotes the interval bounds of CDF of the composing uncertain number.

$$\left[\ \overline{F}_{X+Y}(z), \underline{F}_{X+Y}(z)\ \right] = \left[\inf_{z=x+y} \min\left(\ \overline{F}_X(x) + \overline{F}_Y(y), 1\ \right), \sup_{z=x+y} \max(\underline{F}_X(x) + \underline{F}_Y(y) - 1, 0) \right] \quad (2)$$

Figure 7 displays several common arithmetic examples. These arithmetic operations can be performed in the same signature in `pyuncertainnumber` as real number arithmetics in Python via operator overloading. Fr ' e chet are employed as default except the dependency can be otherwise justified. Importantly, now these calculations yield regirous results that are guaranteed to enclose all possible distributions of the output variable so long as the input uncertain number were all sure to enclose their respective distributions.

3.2. *Non-deterministic propagation*

Scientific computing typically involves a mathematical model, for example a coupled system of nonlinear partial differential equations, to simulate the behaviour of natural or engineered systems. Many such applications involve high-fidelity numerical solutions as a complicated black-box model (e.g. CFD) in a non-intrusive setting.

For a comprehensive uncertainty analysis, all uncertainty sources (e.g. model inputs, initial or boundary conditions, model form assumptions, numerical approximation, and model extrapolation) should be appropriately characterised and and have their contributions to the total uncertainty of the system response quantity of interest (QoI) elucidated, such that efficient uncertainty reduction or management can be conducted by decision makers. A notable example that embodies the idea of such uncertainty framework is the NASA UQ challenge [19].

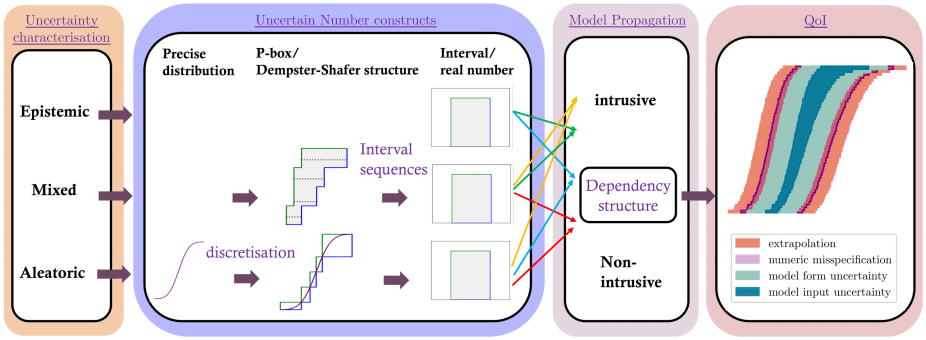


Figure 8. *Flowchart of a comprehensive uncertainty analysis pipeline. Various dependency structures are denoted in colored arrows.*

Figure 8 outlines a general computational pipeline typical for scientific numerical computations with a black-box model, which features the probability bounding approach during the stages of characterisation, propagation and finally the representation of QoI, and echos the idea that aleatory and epistemic uncertainty should be treated differently. Pairs of bounds constitute layered colour-shaded regions reflecting the sources of uncertainties involved. This provides a straightforward and intuitive way to account for the contributions of uncertainties in a comprehensive framework.

It should be noted that *interval analysis* plays a central role in the propagation of epistemic uncertainty and further mixed uncertainty or mixture of uncertainty. The *slicing* algorithm discretises a p-box into intervals and effectively propagate intervals. The same rigorous strategy also applies to precise distributions through outward-rounding.

Regarding the propagation methods for different types of uncertainty, various methods have been proposed, intrusive or nonintrusive, depending on the characteristic of the performance function such as linearity, monotonicity, or the accessibility to gradients. Table 1 lists the characteristics of many methods. Similar to the cases of information constraints for uncertainty characterisation, the more knowledge about the mathematical model, the higher chances of finding an efficient method. Notably, enriched sampling methods such as nested Monte Carlo or interval Monte Carlo allows the outward (rigorous) implementation for a sampling signature consistent with aleatory uncertainty. This demonstrates the universal applicability and compatibility of sampling-based approaches for generic deterministic black-box models in scientific computing. `pyuncertainnumber` provides high-level API to these methods and will automatically inspect the appropriateness of the propagating method given the characteristics of the function and uncertainty.

```
""" a typical workflow of uncertainty characterisation and propagation """
from pyuncertainnumber import Propagation

# constructions of uncertain number
a = pun.I(2, 3)
b = pun.normal(4, 1)
c = pun.uniform([4,5], [9,10])

# specify a response function
def foo(x): return x[0] ** 3 + 5 * x[1] + x[2]

# intrusive signature where uncertain numbers are drop-in replacements
response = foo([a, b, c])

# alternatively, to use a generic high-level propagation API
p = Propagation(vars=[a,b,c],
      func=foo,
      method='slicing',
      interval_strategy='direct'
)

# heavy-lifting of running propagation
response = p.run(n_slices=50)
```

Table 1. *Characterisitics of uncertainty propagation methods*

uncertainty	Methods	cost	characteristics	Non-intrusive applicability
Aleatory	Monte Carlo methods	***	Brute-force sampling	Yes
	Taylor expansion	*	linearity assumption; gradient required	No
Epistemic	interval arithmetic	*	vulnerable to dependency problem	No
	Endpoints	**	monotonicity over intervals; exponential growth 2^d	Yes
	Subinterval reconstitution	***	computationally intensive with exponential growth $(m+1)^d$	Yes
	Cauthy-Deviate method	***	linearity assumption	Yes
	gradient-based optimisation	*	gradients required	No
	Gradient-free optimisation	***	general applicability	Yes
Mixed	Slicing	***	transformed into interval propagation; cartesian product required	Yes
	interval Monte Carlo	***	general applicability with heavy cost; cartesian product required	Yes
	nested Monte Carlo	***	double-loop sampling conditioned on epistemic variables; mixture of uncertainty	Yes

3.3. *Uncertainty-aware surrogate models*

Modern advanced numerical simulations are often computationally expensive, making abundant model evaluations required by uncertainty analysis impractical or even intractable. Surrogate models are therefore utilised to learn and generalise from observed data (a limited subset of DOE). Epistemic uncertainties exist not only in the model form (i.e. parameter and structure) but also in the extrapolation of predictions.

To account for such uncertainty, a probabilistic take of machine learning brings models that represent parameters in probability distributions (Bayesian neural networks), and models that represent a distribution of function structures (Gaussian Process). On the other hand, an intervalised or distribution-free view employs models that consider interval-valued parameters (Interval Predictor Model). These models have the prospects of efficiently propagating the input uncertainties to fully characterise the tail probability of the QoI for example in the imprecise reliability analysis (will be discussed in Section 9).

With many machine learning frameworks available in the Python ecosystem (e.g. Tensorflow, GPflow, PyIPM, etc), we provide the algorithms and the interface in `pyuncertainnumber` to extend those models to be combined within our framework to further propagate uncertain numbers for a efficient and comprehensive uncertainty management.

4. Risk, reliability, and design optimisation under uncertainty

Uncertainty is often a means to an end for scientific computations. For example, risk assessment of environmental hazards (e.g. earthquakes), evaluation of performance and reliability of engineered system under uncertain environmental conditions, and supports of decision-makings for optimal engineering designs. An increasing acknowledgement of the mixture of uncertainty encountered in real world problems has led to the transition from the deterministic safety factor approach to probabilistic risk assessment, further to performance-based or reliability-based design protocols in several safety-critical industries [5]. A comprehensive and expressive uncertainty framework that reflects the contributions of various uncertainties is essential, and an accessible toolkit with universal uncertainty representations is beneficial in facilitating the adoption of comprehensive uncertainty management.

4.1. *Imprecise reliability analysis*

Conventionally, in the analysis of a probabilistic safety framework, the probability of failure is given as $p_f = \boldsymbol{P}[g(\boldsymbol{X}) \leq 0]$ which involves an integration of the joint probability distribution over the failure domain. Given the presence of epistemic uncertainty, the failure probability presents as an interval bound as opposed of precise probability measure. Consider a generic simulation in the face of mixed uncertainties such that the system variables $\boldsymbol{X} = (X_1, ..., X_n)^T$ cannot be precisely characterised but to be more appropriately modelled as an `uncertain number`, as suggested in Expressive power of uncertain number. The interval of failure probability can be given as [20]:

$$\left[\underline{p}_f, \overline{p}_f\right] = \left[\sum_{\eta_i : \geq \inf(g(\eta_i))} \alpha_i, \sum_{\eta_i : \geq \sup(g(\eta_i))} \alpha_i\right] \tag{3}$$

in which $\inf$ and $\sup$ denote the infimum and supremum function; Each uncertain number X_i is discretised as pairs of focal elements and probability masses $\left\{(\eta_i, \alpha_i)_1^N\right\}$. The computation of $g(\boldsymbol{X})$ can be straightforwardly done by the methods discussed in Section 7 and `pyuncertainnumber` provides a simple function to easily evaluate the probability interval.

Figure 9 illustrates the conceptual comparison of expressing the probability of failure for both probabilistic and imprecise frameworks. Conventionally, the p_f can be estimated using

a Monte Carlo estimator that reads: $p_f = \frac{1}{N}\sum_i^N \boldsymbol{I}_{g(\boldsymbol{X})\leq 0}(\boldsymbol{x}_i)$, where $\boldsymbol{I}$ denotes an indicator function, as shown in different colours in the histogram. This summation also depicts the blue cross in the empirical CDF in the top figure, where an interval of $p_f \in \left[\underline{p}_f, \overline{p}_f\right]$ is shown in red.

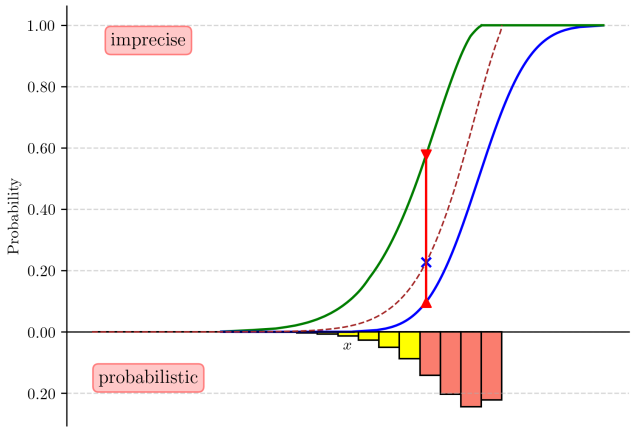


Figure 9. *Illustration of failure probability $\mathbb{P}[g(\boldsymbol{X}) \leq 0]$ for both probabilistic and imprecise frameworks where the probabilistic input is enclosed in the imprecise input*

4.2. Robust designs under epistemic and mixture of uncertainty

Considerable progress has been done towards design optimisation under uncertainty with a probabilistic framework. Typical formulations include robust optimisation aiming for designs less sensitive to variation and reliability-based optimisation whose constraints are less prone to failure under variability. However, real world scenarios commonly involve lack-of-knowledge uncertainties and, more often, mixture of uncertainties, which brings additional subtleties in the formulation. In recognition of the epistemic uncertainty, optimisation is directed with respect to an interval bound. As an example, Figure 10 shows a simple exemplar objective function assumed to represent the cost of manufacture, see (4), in the face of pure epistemic uncertainty.

$$C(x1, x2) = x_1 x_2 - \sin(x_1)x_2^2 + x_1^2 + 300 \tag{4}$$

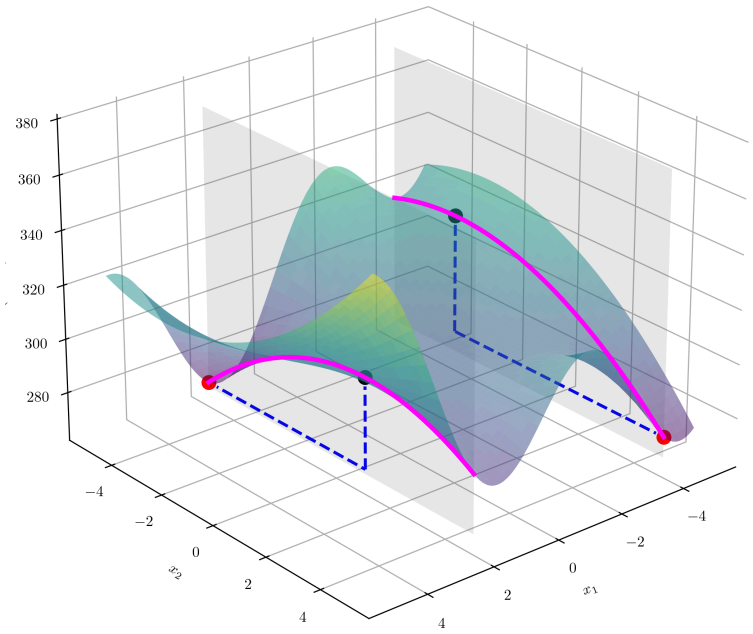


Figure 10. *An examplar objective function. Two vertical planes corresponding to design variables $x_1 = -4$ and $x_1 = 2$ are shown, in which the intersecting curves as well as the maximum and minimum are specifically marked.*

where x_1 denotes design variables of the range of $[-5, 5]$, while x_2 denotes uncertain parameters. Given the pure epistemic uncertainty, a performance-oriented objective can be intuitively formulated as minimising the upper bound: $\min \overline{C}$, or a robustness-oriented objective which minimises the width the cost interval: $\min w\big(\big[\underline{C}, \overline{C}\big]\big)$.

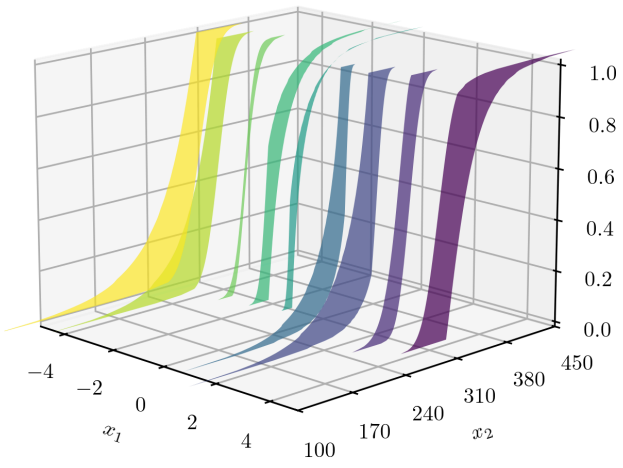


Figure 11. *Illustration of the cost objective represented in uncertain numbers with respect to designs (x_1) in the face of mixed-type uncertain parameter (x_2).*

In mixed uncertainty cases, the cost w.r.t a design is effectively presented by p-boxes , as seen in Figure 11. The existence of epistemic domain results in a bounded objective such as an expected performance indicator or a probability measure, for which both operations will average out the aleatory space. Intuitively, in this example, one would desire to find the design accomplishing the minimum of the expected cost $\left[\underline{\mathbb{E}}(C), \overline{\mathbb{E}}(C)\right]$. Alternatively, it is also possible to prescribe the performance requirement in a probability statement as below, which optimises with the probability of having a cost higher than a prescribed budget c_j, where $p_f(C) = \mathbb{P}\left[C(x_1, x_2) > c_j\right]$.

$$x_1^* = \arg\min \overline{\mathbb{E}}(C) \text{ or } x_1^* = \arg\min \overline{p}_f(C) \tag{5}$$

This effectively introduces a constraint as to the budget. Note that upper bounds are used in the current formulation to take into effect of the epistemic uncertainty and to reflect a sense of performance guarantee. Additionally, to be more pragmatic, the objective can be compounded with another constraint which could for example reflects a cost-production tradeoff that a lower cost is aimed on the condition that a certain production requirement is probabilistically satisfied. An examplar formation can be given as follows where g_i stands for extra constraint functions:

$$\arg\min\left\{\overline{\mathbb{E}}(C) : \mathbb{P}[g_i(x1, x2) \leq 0] \geq 1 - \varepsilon_i\right\} \tag{6}$$

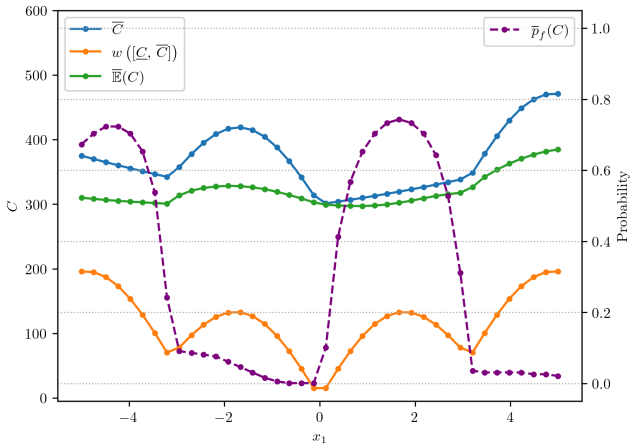


Figure 12. *Optimisation tasks of different formulations in the face of mixed uncertainty*

5. Conclusion and Outlook

To know what you do not know suggests the importance to realise the assumptions and applicability of scientific computations to maximise the credibility for the resulting predictions, designs and decisions. Given the various sources of uncertainties and the often limited empirical information, how to appropriately represent, aggregate, propagate uncertainties is a critical challenge for trustworthy reliability and risk assessments, especially for safety critical applications. Current practices tend to go overboard with unjustified assumptions of Gaussianality and independence, mostly due to the computational simplicity and the lack of tools for a comprehensive uncertainty analysis. This leads to a need of a computational framework that effectively balances representational expressiveness and computational feasibility.

This paper presents the framework of `uncertain number` which fills this gap and presents several advantages: (i) it is highly expressive enabling faithful characterisation of an uncertain quantity given various scenarios of partial knowledge where conventional probability theory struggles to cope; (ii) it provides a closed environment where various characterisation, aggregation and propagation operations can be consistently conducted based on

the unified structure, `uncertain number`; (iii) it is underpinned by a probability bounding mechanism which intuitively showcases the notion of epistemic uncertainty — wherein increased knowledge leads to progressively tighter bounds; (iv) the bounding mechanism enables an explicit differentiation of aleatory and epistemic uncertainties, allowing their respective contributions to be accounted for during both characterisation and propagation.

The developed Python library, `pyuncertainnumber`, facilitates trustworthy management of uncertainty through faithful representation and rigorous propagation. Unlike other established Python-based general purpose uncertainty quantification (UQ) tools that primarily focus on probabilistic modelling —– often requiring precise specification of distribution shape and dependency structures, which is challenging in practice in the face of partial information, `pyuncertainnumber`, however, relaxes such specification and is capable of rigorous treatment even with unknown knowledge states. It explicitly addresses both aleatory and epistemic uncertainty using appropriate methods without making unjustified assumptions, advancing uncertainty management to handle both variability and incertitude. Consequently, it enables a more robust framework for addressing real-world UQ challenges such as mixed uncertainty propagation and characterisation under partial knowledge.

As probabilistic programming provides support for automatic inference, we aim for an imprecise uncertainty analysis framework, where variables are consistently represented by uncertain numbers in the face of both variability and incertitude, allowing for extensions of deterministic functions to be computed in an automatic, comprehensive, and rigorous manner. Our next focus will be extending the framework of probabilistic programming, which has a heavy focus on Bayesian inference, into an imprecise realm where more comprehensive uncertainty structures can be integrated into the learning and inference of machine learning models.

Acknowledgement

This work has been supported by the DAWS2 (Development of Advanced Wing Solution 2) project funded by Innovate UK.

References

[1] C. J. Roy and W. L. Oberkampf, “A comprehensive framework for verification, validation, and uncertainty quantification in scientific computing,” *Computer methods in applied mechanics and engineering*, vol. 200, no. 25–28, pp. 2131–2144, 2011, doi: https://doi.org/10.1016/j.cma.2011.03.016.

[2] W. L. Oberkampf, J. C. Helton, C. A. Joslyn, S. F. Wojtkiewicz, and S. Ferson, “Challenge problems: uncertainty in system response given uncertain parameters,” *Reliability Engineering & System Safety*, vol. 85, no. 1–3, pp. 11–19, 2004, doi: 10.1016/j.ress.2004.03.002.

[3] R. C. Smith, *Uncertainty quantification: theory, implementation, and applications*. SIAM, 2024. doi: 10.1137/1.9781611973228.

[4] A. Gray *et al.*, “From inference to design: A comprehensive framework for uncertainty quantification in engineering with limited information,” *Mechanical Systems and Signal Processing*, vol. 165, p. 108210, 2022, doi: 10.1016/j.ymssp.2021.108210.

[5] M. Beer, S. Ferson, and V. Kreinovich, “Imprecise probabilities in engineering analyses,” *Mechanical systems and signal processing*, vol. 37, no. 1–2, pp. 4–29, 2013, doi: 10.1016/j.ymssp.2013.01.024.

[6] E. Patelli, M. Broggi, S. Tolo, and J. Sadeghi, “COSSAN SOFTWARE: A MULTIDISCIPLINARY AND COLLABORATIVE SOFTWARE FOR UNCERTAINTY QUANTIFICATION,” in *Proceedings of the 2nd International Conference on Uncertainty Quantification in Computational Sciences and Engineering (UNCECOMP 2017)*, in UNCECOMP 2017. 2017, pp. 212–224. doi: 10.7712/120217.5364.16982.

[7] S. Ferson and J. G. Hajagos, “Arithmetic with uncertain numbers: rigorous and (often) best possible answers,” *Reliability Engineering & System Safety*, vol. 85, no. 1–3, pp. 135–152, 2004, doi: 10.1016/j.ress.2004.03.008.

[8] S. Ferson and L. R. Ginzburg, “Different methods are needed to propagate ignorance and variability,” *Reliability Engineering & System Safety*, vol. 54, no. 2–3, pp. 133–144, 1996, doi: 10.1016/s0951-8320(96)00071-3.

[9] S. Destercke, D. Dubois, and E. Chojnacki, “Unifying practical uncertainty representations–I: Generalized p-boxes,” *International Journal of Approximate Reasoning*, vol. 49, no. 3, pp. 649–663, 2008, doi: 10.1016/j.ijar.2008.07.003.

[10] R. C. Williamson and T. Downs, “Probabilistic arithmetic. I. Numerical methods for calculating convolutions and dependency bounds,” *International Journal of Approximate Reasoning*, vol. 4, no. 2, pp. 89–158, 1990, doi: 10.1016/0888-613x(90)90022-t.

[11] S. Ferson, V. Kreinovich, L. Ginzburg, D. S. Myers, and K. Sentz, *Constructing probability boxes and Dempster-Shafer structures*, no. 4015. Sandia National Laboratories, 2003. doi: 10.2172/809606.

[12] N. Gray, S. Ferson, M. De Angelis, A. Gray, and F. B. de Oliveira, “Probability bounds analysis for Python,” *Software Impacts*, vol. 12, p. 100246, 2022, doi: https://doi.org/10.1016/j.simpa.2022.100246.

[13] S. Ferson, V. Kreinovich, J. Hajagos, W. L. Oberkampf, and L. Ginzburg, “Experimental uncertainty estimation and statistics for data having interval uncertainty.,” 2007. doi: 10.2172/910198.

[14] M. de Angelis, “intervals.” [Online]. Available: https://doi.org/10.5281/zenodo.6205624

[15] K. Tretiak and S. Ferson, “Should data ever be thrown away? Pooling interval-censored data sets with different precision,” *International Journal of Approximate Reasoning*, vol. 156, pp. 114–133, 2023, doi: 10.1016/j.ijar.2023.02.007.

[16] S. Ferson *et al.*, “Natural language of uncertainty: numeric hedge words,” *International Journal of Approximate Reasoning*, vol. 57, pp. 19–39, 2015, doi: 10.1016/j.ijar.2014.11.003.

[17] C. Donnelly and P. Embrechts, “The devil is in the tails: actuarial mathematics and the subprime mortgage crisis,” *ASTIN Bulletin: The Journal of the IAA*, vol. 40, no. 1, pp. 1–33, 2010, doi: 10.2143/ast.40.1.2049222.

[18] W. L. Oberkampf *et al.*, “Dependence in probabilistic modeling, Dempster-Shafer theory, and probability bounds analysis.,” 2004. doi: 10.2172/919189.

[19] L. G. Crespo and S. P. Kenny, “The NASA langley challenge on optimization under uncertainty,” *Mechanical Systems and Signal Processing*, vol. 152, p. 107405, 2021, doi: 10.1016/j.ymssp.2020.107405.

[20] H. Zhang, R. L. Mullen, and R. L. Muhanna, “Interval Monte Carlo methods for structural reliability,” *Structural Safety*, vol. 32, no. 3, pp. 183–190, 2010, doi: 10.1016/j.strusafe.2010.01.001.

Open Access | https://doi.org/10.25080/ppgk2467

SciPy 2025

July 7 - July 13, 2025

Proceedings of the 24th Python in Science Conference
ISSN: 2575-9752

Redist, a python tool for model-agnostics binned-likelihood fits in High Energy Physics

Marco Colonna[1], **Johannes Albrecht**[1], **Florian Bernlochner**[2], **Lorenz Gärtner**[3], **Abhijit Mathad**[4], and **Biljana Mitreska**[1]

[1]Technische Universität Dortmund, [2]University of Bonn, [3]Ludwig Maximilian University of Munich, [4]CERN

Abstract

Current High Energy Physics measurements rely on analysing the data by using underlying theoretical assumptions in model building that do not account for possible deviations from the nominal model on which the assumptions are based. This approach can introduce potential bias in parameter extraction and complicate the reinterpretation of measurements without fully reanalyzing the data. Redist is a package for building model-agnostic binned-likelihood fits, allowing combination and enhancing the reinterpretation of datasets. Redist allows direct inference of theoretical parameters of any beyond the Standard Model scenario through histogram reweighting, properly taking into account changes in kinematic distributions. By leveraging truth-level observables and capturing correlations, Redist can be interfaced with different theoretical backends to connect theoretical parameters with the *pyhf* environment for fitting. We present a Redist-HAMMER interface that has been developed to address direct Beyond the Standard Model measurements by reinterpreting existing datasets collected by various High Energy Physics collaborations, using the increasingly popular theoretical backend HAMMER. This enables model-agnostic interpretation of data—a crucial step for advancing precision flavor physics. Further applications of the Redist package in different fields, along the lines of the Redist-HAMMER interface in High Energy Physics, to extend *pyhf* models are possible and straightforward to implement.

Keywords High Energy Physics, Flavour Physics, Model agnostic, Data reinterpretation

1. Introduction

As High Energy Physics (HEP) is entering a high-precision measurements era, minimizing assumptions about theoretical models in the data analysis becomes necessary for testing alternative interpretations on the same data. Usually HEP analyses rely on theoretical assumptions, such as the Standard Model (SM) predictions or the shapes of simulated samples produced based on a set underlying kinematic assumptions. These inherent dependencies make reinterpretation of datasets collected by the the LHCb [1] and Belle II [2] collaborations, leading the field of flavour physics, challenging. To avoid necessarily generating simulated samples for each Beyond the Standard Model (BSM) scenario, tools like the HAMMER [3] or the EOS [4] packages have been developed, giving access to frameworks to adjust simulation-derived models to arbitrary BSM scenarios by reweighting the kinematic distributions of the particles to any assumption differing from the SM, also allowing to vary theoretical parameters of the chromodynamics of the events, known as Form Factors (FFs). Handling FFs is very important specifically in the study of beauty-meson decays involving neutrinos in the final state. The HAMMER interface, in particular, makes use of truth-level kinematic information, meaning the real momentum components and particle identity of the simulated particles, in the initial and final states to apply an event-by-event weighting.

Published Jul 10, 2025

Correspondence to
Marco Colonna
marco.colonna@cern.ch

Open Access

Copyright © 2025 Colonna *et al.*. This is an open-access article distributed under the terms of the Creative Commons Attribution 4.0 International license, which enables reusers to distribute, remix, adapt, and build upon the material in any medium or format, so long as attribution is given to the creator.

It allows to vary the Form Factor parametrization, from the one that has been used in the simulation production to the one that is desired for the analysis. The HAMMER interface also allows testing different BSM scenarios by assigning weights to the events on the basis of the different NP contributions associated with the Effective Field Theory (EFT) [5] operators listed in Table 1. In general for each operator two contributions to the Wilson Coefficients are defined respectively representing the real and imaginary parts of the Wilson Coefficient.

Table 1. *List of the EFT operators implemented in HAMMER. The symbols u and d represent the up-like and the down-like quarks involved in the vertex, as the symbols l and ν represent the lepton and the neutrino. The pedestals R and L represent the chiral-state of the particles (right-, or left-handed). Any operator involving right-handed neutrinos implemented in HAMMER has been excluded from this list.*

Current type	WC Name	NP operator
Standard Model	SM	$[\bar{u}\gamma^\mu P_L d]\left[\bar{l}\gamma^\mu P_L \nu\right]$
Vector	V_{qLlL}	$[\bar{u}\gamma^\mu P_L d]\left[\bar{l}\gamma^\mu P_L \nu\right]$
	V_{qRlL}	$[\bar{u}\gamma^\mu P_R d]\left[\bar{l}\gamma^\mu P_L \nu\right]$
Scalar	S_{qLlL}	$[\bar{u}P_L d]\left[\bar{l}P_L \nu\right]$
	S_{qRlL}	$[\bar{u}P_R d]\left[\bar{l}P_L \nu\right]$
Tensor	T_{qLlL}	$\left[\bar{u}\sigma_{\mu\nu}P_L d\right]\left[\bar{l}\sigma_{\mu\nu}P_L \nu\right]$

Python is becoming increasingly popular in High Energy Physics (HEP) analysis due to its flexibility, ease of use and the growing number of scientific tool implemented on it. At the same time, the HAMMER package is gaining traction in the community as a powerful framework to reinterpret previously analyzed datasets and explore BSM scenarios through FFs reweighting. The Redist-HAMMER (github repository) is the first full Pythonic interface between the HAMMER package and a fitting environment, by integrating it with the *pyhf* framework [6], [7]. This allows HAMMER-processed samples to be used directly within *pyhf*'s binned-likelihood models [8], via the Redist modifiers of the likelihood, encoding the BSM and FF dependence of the shapes as defined by the HAMMER theoretical backend. Other efforts, such as the RooHammerModel [9], have been spent to address similar functionalities in the C++-based RooFit-HistFactory framework [10]. Redist-HAMMER builds on this idea by bringing it into the Python ecosystem, providing a flexible reinterpretation tool that supports combination and fitting workflows entirely in Python allowing for an easy to start interface to apply reinterpretation of HEP datasets.

The structure of this document is as follows: The *pyhf* and the Redist package describes the Redist environment and the custom modifier functionalities implemented as an extension of *pyhf*, The Redist-HAMMER interface describes the Redist-HAMMER interface to *pyhf* with a specific focus on the methods to start coding with it and perform simple fits, Using HAMMER in *pyhf* shows a set of proof of usage of the package discussing advantages of the methods that are used, finally Outlook provides an outlook on further implementation of the Redist package in HEP and in multiple different fields of science.

2. The *pyhf* and the Redist package

The *pyhf* package implements a very popular model building method for binned-likelihood fits. The most generic binned-likelihood can be written as the product of three components:

$$L = L_{data} \cdot C_{exp} \cdot C_{th} \tag{1}$$

defined respectively as the data likelihood, L_{data}, and two components introducing the experimental and theoretical constraints, C_{exp} and C_{th}. L_{data} is defined, according to the binned nature of the model, as the product of Poisson probabilities of the yields observed in data, n, and the expected number of events, ν:

$$L_{data}(n;\eta,\chi) = \prod_{c\in \text{ channel}} \prod_{b\in \text{ bins}} Pois(n_{cb} \mid \nu_{cb}(\eta,\chi)) \tag{2}$$

where each channel represents a different and disjoint binned distribution and bins correspond to the histogram bins. The rates ν are functions of constrained and unconstrained parameters, respectively defined as η and χ. The constraints applied on the χ parameters might come from both experimental or theoretical priors, whose definition in the likelihood is contained in the C_{exp} and C_{th} terms:

$$C_i = \prod_{\chi\in\chi_i} c_\chi(a_\chi \mid \chi) \tag{3}$$

where the index i represents the two types of constraints, and a represents the auxiliary data enforcing the constraints.

The *pyhf* package requires the rates ν of the different contributions of the model to be defined as the expected number of events taken from simulation, allowing the definition of a set of modifiers which will change the rates in the fit on the depencence of the given degrees of freedom. Different types of modifiers are already defined in *pyhf*, so that the event rate for a given channel c and bin b is defined as:

$$\nu_{cb} = \sum_{s\in samples} \prod_{k\in k} k_{scb}(\eta,\chi) \times \left(\nu^0_{scb}(\eta,\chi) + \sum_{\Delta\in\Delta} \Delta_{scb}(\eta,\chi)\right) \tag{4}$$

where samples represents the set of contributions appearing in the same channel, ν^0 is the initial event rate, and k and Δ are respectively multiplicative and additive modifiers. One challenge with defining the rates in terms of the modifiers is that it may be complicated to capture more intricate dependencies, particularly if the degrees of freedom η and χ are expressed as the NP coefficients from the HAMMER package or as FF parameters.

The Redist package [11] addresses this problem with a reinterpretation method where the binned event counts are updated on the basis of the new kinematic assumptions. The method has been implemented in the *pyhf* fitting interface through the definition of a custom modifier of the likelihood: a modifier varying event rates, computed by custom functions or external theoretical backends. The Pythonic interface of the HAMMER package has been used to build an interface between HAMMER and a generic Python-based fitting interface, while the Redist package has been implemented with a new custom modifier to use the HAMMER-weighted histograms in the definition of the *pyhf* models.

This implementation of Redist-HAMMER allows us to make use of the HAMMER package directly in our fits and is a valuable example of how the Redist weighting method can be applied easily to specific cases by interfacing HEP theory packages to the *pyhf* fitting environment.

3. The Redist-HAMMER interface

The Redist-HAMMER follows a deliberately nested class structure, where each class is responsible for a narrowly defined task, adhering to the principle of single responsibility. Higher-level classes delegate specific subtasks to lower-level components, which are composed as objects within them. This design, while seemingly convoluted, promotes clarity, testability and separation of concerns by ensuring that each layer handles only its designated function and relies on contained objects for more granular operations. This modular, layered structure enables to effectively apply both unit and integration testing to validate each class's behavior both in isolation and in collaboration with the other components. The structure of the package and its purpose, with respect to the HAMMER interface and *pyhf*, are summarized in Figure 1.

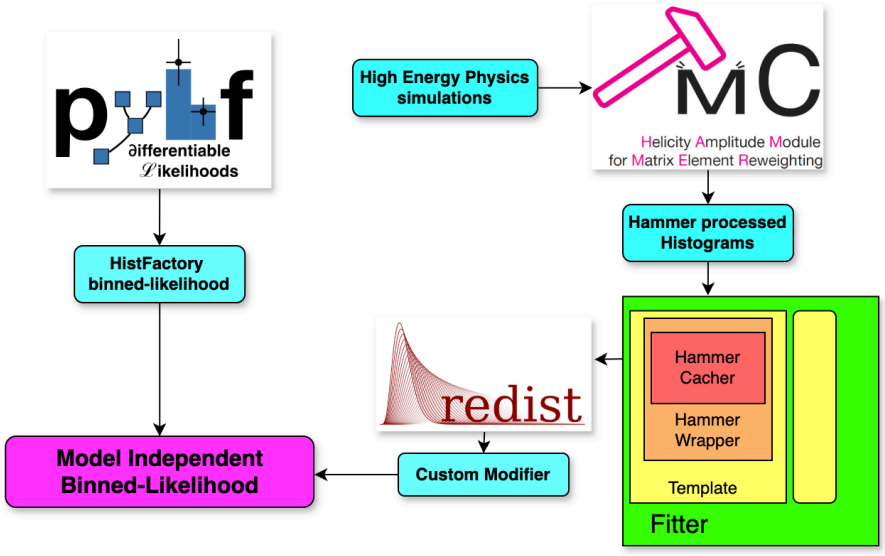


Figure 1. *System diagram of the Redist package allowing HAMMER processed samples to be used in* pyhf *inference. The nested structure of the Redist-HAMMER interface is summarized.*

The most inner object in the nested structure is the HammerCacher, a class that uses the Python HAMMER interface functions to read from a file a HAMMER-processed histograms, which are created applying HAMMER to the simulations and contain the HAMMER weighting parameters (generally ∼10-20 new degrees of freedom for describing the FF parametrization and BSM injection) and the mathematical tools to extrapolate the variations of the shape as function of them. The HammerCacher stores the bin content of the HAMMER-processed histogram and updates it on the basis of any set of NP and FF parameters that were defined at HAMMER processing time. To avoid useless iterations with the HAMMER weighting, a caching procedure for the values of the parameters has been used: the values of the NP and FF parameters are stored, and the histogram bin content is updated only when the parameters are effectively varied. Multiple HammerCacher instances can be embedded in the same object and their parameters varied at the same time using the MultiHammerCacher. In addition the HammerNuisWrapper is defined as the one that contains a HammerCacher and attaches a set of multiplicative parameters to vary the bin content of the HAMMER histogram. A similar structure has been built to include non-HAMMER-weighted contributions in the model, meaning regular histograms produced from HEP simulations; the shapes of these contributions are defined by reading a file, and from the set of multiplicative parameters—representing, for example, the yields. Each NuisWrapper is further contained in a Template class, which defines an object allowing access to the bin content of the contributions and enabling changes to the parameters. The Template class also contains options to generate toy samples or to return the whole template histogram for a specific injection of parameters. Multiple Templates are stored in a single Fitter object, which is used to represent what in *pyhf* would be defined as a Channel, and it is used to interface the preceding nested structure of classes to Redist.

An additional class named Reader is taking care of model construction using the configurations included in a single JSON file:

```
{
  "B02DstTauNu": {
    "fileNames": ["File_containing_a_histogram.dat"],
    "histoname": "Name_of_the_histogram",
    "ffscheme": {
      "name":"SchemeBLPRXP",
      "Process":"BtoD*",
      "SchemeVar":"BLPRXPVar"
    },
    "wcscheme": "BtoCMuNu",
    "scalefactor": 1.0,
    "formfactors": {
      "delta_RhoSq": 1.5,
      "delta_cSt": 2.0
    },
    "wilsoncoefficients": {
      "SM": [1.0,0.0],
      "S_qLlL": [0.0,0.0]
    },
    "nuisance": {
      "Luminosity": 1.
    },
    "ishammerweighted": true,
    "injectNP":true,
    "axistitles": [
      "title"
    ],
    "strides":[1],
    "binning": [[0.0, 10.0]]
  }
}
```

The path to the file containing the HAMMER-processed histogram is specified, together with all the information required to read the histogram—namely, the name and the FF scheme used in the HAMMER processing. Other options are included to define the HAMMER parameters to be handled and to organize a multi-channel model, such as “injectNP” to tag some of the contributions as purely SM, or “ishammerweighted” to distinguish contributions that are not HAMMER-processed. A scalefactor is attached to the contributions to apply a fixed pre-scaling to the yield of the contributions. Finally, information useful at plotting time, such as axis titles and binnings, is included.

Given a JSON file structured as the one above, a Fitter can be easily produced simply by using the Reader object, letting the machinery compose the nested structure on top of which the Fitter sits:

```
import pyhf
import json
from redist import modifier
from redist import modifier_hammer

reader = modifier_hammer.Reader("config.json")
fitter = reader.createFitter(True) # True enables a Verbose creation of the Fitter
```

then the Fitter object allows easy handling of the HAMMER-processed contributions and their degrees of freedom—for example, generating pseudo-data where, for a given injection of NP and FF parameters, a binned distribution is produced according to a Poissonian distribution for each bin:

```
params_0 = {"SM" : 1.,"Re_S_qLlL" : 0.,"Im_S_qLlL" : 0. ,
            "delta_RhoSq" : 0.,"delta_cSt" : 0. ,
            "Luminosity": 1.}
toy_data = fitter.get_template(0).generate_toy(**params_0)
# generate_toy() produces according to a Poisson distribution
# generate_template() to produce the exact value without fluctuations
fitter.upload_data(toy_data) # store a toy_data to be accessed as fitter._data
```

Given these very simple ingredients, it is now straightforward to operatively apply the Redist method by defining the custom modifier_hammer class, inheriting from the original modifier, interfacing it with the Fitter. The original modifier class can be easily adapted in the same way to any other interface.

4. Using HAMMER in *pyhf*

Finally it is possible to operatively apply the Redist weighting method in *pyhf*. A static reference distribution needs to be defined, together with an alternative distribution that is a function of the parameters of interest:

```
SM_distribution = fitter.get_template(0).generate_toy(**params_0)

def SM_B02DstTauNu():
  return SM_distribution

def NP_B02DstTauNu(**kwargs):
  params_NP = {"SM": 1.,
               "Re_S_qLlL": **kwargs["Re_S_qLlL"],
               "Im_S_qLlL": **kwargs["Im_S_qLlL"],
               "delta_RhoSq": **kwargs["delta_RhoSq"],
               "delta_cSt": **kwargs["delta_cSt"],
               "Luminosity": 1.}
  NP_distribution = fitter.get_template(0).generate_template(**params_NP)
  return NP_distribution
```

these functions will effectively be used to create the custom modifier and vary the shape of the histogram by multiplying the initial distribution by the ratio of the two functions:

$$M(\vec{p}) = M(\vec{p_0}) \times \frac{f(\vec{p})}{f_0(\vec{p_0})} \tag{5}$$

where $M(\vec{p})$ is the shape of the template as a function of the parameters $\vec{p}$, f_0 is the reference hypothesis distribution, and f is the alternative distribution. A set of parameters is then defined and the custom modifier created. The custom modifier is assigned a name to distinguish it from other types of standard modifiers and is added later to an already existing, but not custom, *pyhf* model through the *expand_pdf* attribute:

```
new_params = {
  "Re_S_qLlL": {
      "inits": (0.0,),
      "bounds": ((-3.0, 3.0),),
      "paramset_type": "unconstrained"
  },
  "Im_S_qLlL": {
      "inits": (0.0,),
      "bounds": ((-3.0, 3.0),),
      "paramset_type": "unconstrained"
  },
  "delta_RhoSq": {
      "inits": (0.0,),
      "bounds": ((-3.0, 3.0),),
      "paramset_type": "unconstrained"
  },
  "delta_cSt": {
      "inits": (0.0,),
      "bounds": ((-3.0, 3.0),),
      "paramset_type": "unconstrained"
  }
}

cmod = modifier_hammer.Modifier_Hammer(
  new_params,
  NP_B02DstTauNu,
  SM_B02DstTauNu,
  name="mod_B02DstTauNu"
)

spec = {
  "channels": [
      {
          "name": "RDsMuonic",
          "samples": [
              {
                  "name": "B02DstTauNu",
                  "data": SM_distribution.tolist(),
                  "modifiers": [
                      {
                          "name": "mu",
                          "type": "normfactor",
                          "data": None,
                      }
                  ],
              },
          ]
      }
  ]
}
model = pyhf.Model(spec)
custom_mod_B02DstTauNu = {
  "name": "mod_B02DstTauNu_theory",
  "type": "mod_B02DstTauNu",
  "data": {"expr": "mod_B02DstTauNu_weight_fn"},
}
expanded_pyhf = {**cmod_B02DstTauNu.expanded_pyhf}
model = modifier.add_to_model(
  model,
  ["RDsMuonic"],
  ["B02DstTauNu"],
  expanded_pyhf,
  custom_mod_B02DstTauNu
)
```

The *pyhf* model contains and handles degrees of freedom associated with HAMMER-processed histograms for model-agnostic fits and reinterpretation. More generically, following the same prescription, the *pyhf* model is able to handle arbitrarily complex degrees of freedom where the effect of changing the templates is completely determined by the initial and alternative hypothesis we define as simple python functions.

This method has direct application in HEP analysis, in particular for the reinterpretation of semileptonic analyses. Taking, for example, the semileptonic modes of the beauty mesons $\overline{B^0} \to D^{*+}\mu^-\overline{\nu_\mu}$ and $\overline{B^0} \to D^{*+}\tau^-\overline{\nu_\tau}$, notably, tension exists between the expected ratio of the abundance of the two decay modes and the measured ratio of the respective Branching Fractions (BF):

$$R(D^*) = \frac{\mathrm{BF}\left(\overline{B^0} \to D^{*+}\tau^-\overline{\nu_\tau}\right)}{\mathrm{BF}\left(\overline{B^0} \to D^{*+}\mu^-\overline{\nu_\mu}\right)} \tag{6}$$

has been observed [12]. The Redist method, implemented with the theoretical HAMMER backend, allows to disentangle the tension and study the BSM processes in these decays in an innovative and model-independent way. The HAMMER-weighted sample can be accessed, and a template fit can be applied to study the sensitivity on the parameters of interest given a certain statistical power, defined by the number of events of the template dataset used in the fit. The phase space of the real and imaginary parts of the Wilson Coefficient associated with a scalar-like NP contribution, coupling particles and antiparticles in the same chiral state, has been inspected using a template generated with the RapidSim tool [13] and containing $B^0 \to D^*\tau\nu_\tau$ events. The fit has been done using a pseudodataset identical to the template itself, with different injections of NP, to demonstrate the application of the Redist-HAMMER interface in a small and simple sensitivity study context. This represents a simple and practical example of a use case in flavour physics data analysis, demonstrating the functionalities of maximum-likelihood-fits, generation of pseudodatasets and phase space scanning for confidence level determination.

A Negative-Logarithmic-Likelihood (NLL) fit is performed to find the optimal point, then further fits are applied over a grid of points in the phase space of the real and imaginary parts of the NP Wilson Coefficients. For each point, the NP Wilson Coefficients are fixed, while the rest of the parameters, for instance the SM contributions, are free to float to maximize the NLL. If the grid of phase space is defined with sufficiently fine granularity, this method allows inferring 2-Dimensional (2D) Confidence Intervals (CI) correctly, taking into account correlations among the parameters as shown in Figure 2.

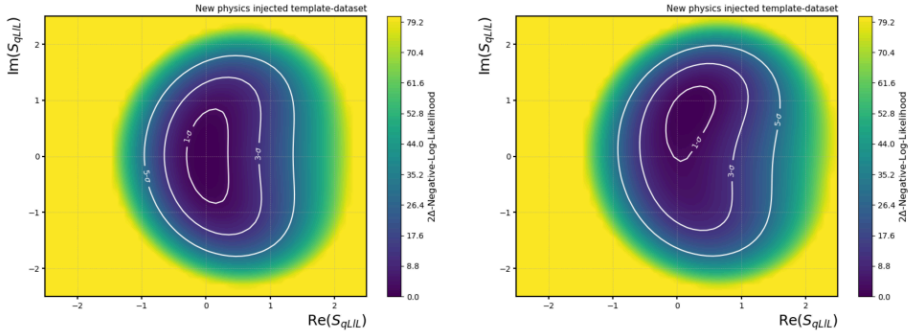


Figure 2. *Negative-Log-Likelihood phase-space scan of the scalar Wilson Coefficient on a template-dataset with no NP injection (left) and on a template dataset with the NP injection of* $Re(S_{qLlL}) = 0.2$ *and* $Im(S_{qLlL}) = 0.8$ *(right). The confidence levels correspontent to 1, 3 and 5 standard deviations are overlaid to both the scans. Both the fits correctly identify the injected New Physics values showing the effectiveness of the method.*

In fact, the custom modifiers applying the weighting can be mixed with the already existing shape modifiers in *pyhf*. This is particularly important as, generically speaking, multiple templates will compose a single channel, and some of them might not be derived from HAMMER-processed histograms, and might require other parameters for the fit. Shape modifiers like the standard histosys of *pyhf* can also be assigned to the HAMMER-weighted derived templates, for example to represent the variation of a different parameter, or to

embed the impact of one of the HAMMER parameters that is not explicitly set free to float in the fit by defining the model as:

```
spec = {
  "channels": [
      {
          "name": "RDsMuonic",
          "samples": [
              {
                  "name": "B02DstTauNu",
                  "data": SM_distribution.tolist(),
                  "modifiers": [
                      {
                          "name": "mu",
                          "type": "normfactor",
                          "data": None,
                      },
                      {
                        "name": "my_histosys",
                        "type": "histosys",
                        "data": {"hi_data": hi_temp.tolist(), "lo_data": lo_temp.tolist()},
                      }
                  ],
              },
          ]
      }
  ]
}
```

where the hi_temp and lo_temp are obtained from the shapes of the contributions with different FFs or WCs injected. An example of this usage, where the two alternative shapes hi_temp and lo_temp have been defined with respect to different injections of a FF parameter ($\Delta\rho^2$), is shown in Figure 3.

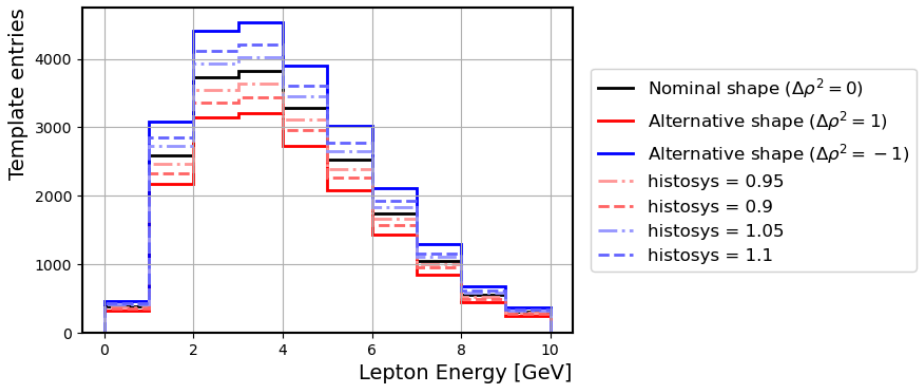


Figure 3. *Distributions of a 1-dimensional* pyhf *template for different values of the* histsys *modifier, defined with respect to two different injection of values of the ρ^2 FF parameter in HAMMER. Redist-HAMMER allows the integration of the theoretical uncertainty on the Form Factor parameter as a standard* histosys *nuisance parameter in* pyhf.

5. Outlook

We present the Redist-HAMMER interface, an extension of the already existing Redist module to interface HAMMER with the fitting environment of *pyhf*. The package has been developed to address the challenge of direct NP measurements through the reinterpretation of the LHCb and Belle datasets, which will be a crucial step for getting a further and deeper understanding of tensions of HEP measurements with SM predictions. The module offers the possibility of easily handling very complex parameters, such as the Wilson Coefficients and the Form Factors, using the HAMMER package as theoretical backend.

Allowing for a correct reinterpretation of data also opens the possibility of combining different datasets in a single coherent fitting environment. The Redist-HAMMER interface has been validated, and studies on the benefits of a coherent combination in a simultaneous fit of different datasets, representing different decay modes, are currently ongoing. By combining different datasets in a simultaneous fit, the degrees of freedom that are shared lead to an enhancement of the sensitivity and possibly to the cancellation of biases that might arise in a post-fit average of different results.

Redist-HAMMER offers a simple and fully pythonic approach to otherwise very complex problems and has already been used to validate other studies on the Wilson Coefficients, such as the recent, and currently in development, simulation-based interface of ROOT.

What has been shown in this document represents one of the many applications, in HEP and beyond, of the Redist weighting method for model building. Future efforts will integrate other popular theoretical backends in flavour physics like Flavio [14], finally allowing data analysts effortless use of, and possibly toggling between, all the different packages.

Fields outside HEP can benefit from the Redist method for building model-independent fits, enabling interpretation of observed patterns in complex systems without relying on predefined models.

References

[1] L. Collaboration, "The LHCb Detector at the LHC," *JINST*, vol. 3, p. S8005, 2008, doi: 10.1088/1748-0221/3/08/S08005.

[2] J. Brodzicka *et al.*, "Physics achievements from the Belle experiment." [Online]. Available: https://doi.org/10.1093/ptep/pts072

[3] F. U. Bernlochner, S. Duell, Z. Ligeti, M. Papucci, and D. J. Robinson, "Das ist der HAMMER: consistent new physics interpretations of semileptonic decays," *The European Physical Journal C*, vol. 80, no. 9, 2020, doi: 10.1140/epjc/s10052-020-8304-0.

[4] D. van Dyk and others, "EOS: a software for flavor physics phenomenology," *Eur. Phys. J. C*, vol. 82, no. 6, p. 569, 2022, doi: 10.1140/epjc/s10052-022-10177-4.

[5] A. Pich, "Effective Field Theory." [Online]. Available: https://arxiv.org/abs/hep-ph/9806303

[6] L. Heinrich, M. Feickert, and G. Stark, "pyhf: v0.7.6." [Online]. Available: https://doi.org/10.5281/zenodo.1169739

[7] L. Heinrich, M. Feickert, G. Stark, and K. Cranmer, "pyhf: pure-Python implementation of HistFactory statistical models," *Journal of Open Source Software*, vol. 6, no. 58, p. 2823, 2021, doi: 10.21105/joss.02823.

[8] K. Cranmer, G. Lewis, L. Moneta, A. Shibata, and W. Verkerke, "HistFactory: A tool for creating statistical models for use with RooFit and RooStats," 2012, doi: 10.17181/CERN-OPEN-2012-016.

[9] J. García Pardiñas, S. Meloni, L. Grillo, P. Owen, M. Calvi, and N. Serra, "RooHammerModel: interfacing the HAMMER software tool with HistFactory and RooFit," *Journal of Instrumentation*, vol. 17, no. 4, p. T4006, 2022, doi: 10.1088/1748-0221/17/04/t04006.

[10] W. Verkerke and D. P. Kirkby, "The RooFit toolkit for data modeling," *eConf*, p. MOLT7, 2003, doi: https://doi.org/10.48550/arXiv.physics/0306116.

[11] L. Gärtner *et al.*, "Constructing model-agnostic likelihoods, a method for the reinterpretation of particle physics results," *The European Physical Journal C*, vol. 84, no. 7, 2024, doi: 10.1140/epjc/s10052-024-13038-4.

[12] S. Banerjee *et al.*, "Averages of b-hadron, c-hadron, and \tau-lepton properties as of 2023". [Online]. Available: https://arxiv.org/abs/2411.18639

[13] G. Cowan, D. Craik, and M. Needham, "RapidSim: An application for the fast simulation of heavy-quark hadron decays," *Computer Physics Communications*, vol. 214, pp. 239–246, 2017, doi: 10.1016/j.cpc.2017.01.029.

[14] D. M. Straub, "flavio: a Python package for flavour and precision phenomenology in the Standard Model and beyond." [Online]. Available: https://arxiv.org/abs/1810.08132

Open Access | https://doi.org/10.25080/nwuf8465

SciPy 2025

July 7 - July 13, 2025

Proceedings of the 24th Python in Science Conference
ISSN: 2575-9752

Reproducible Machine Learning Workflows for Scientists with Pixi

Matthew Feickert[1], **Ruben Arts**[2], and **John Kirkham**[3]

[1]University of Wisconsin–Madison, [2]Prefix.dev, [3]NVIDIA

Abstract

Scientific researchers need reproducible software environments for complex applications that can run across heterogeneous computing platforms. Modern open source tools, like Pixi, provide automatic reproducibility solutions for all dependencies while providing a high level interface well suited for researchers. Combined with the recent emergence of the entire CUDA software stack — from compilers to development libraries — being supported on conda-forge, researchers are now able to easily specify their exact hardware acceleration requirements and software dependencies and get portable computational environments locked down to the digest level.

Keywords reproducible, machine learning, hardware acceleration, CUDA, conda-forge, Pixi

1. Introduction

A critical component of research software sustainability is the reproducibility of the software and computing environments software operates and "lives" in. Providing software as a "package" — a standardized distribution of all source or binary components of the software required for use along with identifying metadata — goes a long way to improved reproducibility of software libraries. However, more researchers are consumers of libraries than developers of them, but still need reproducible computing environments for research software applications that may be run across multiple computing platforms — e.g. scientific analyses, visualization tools, data transformation pipelines, and artificial intelligence (AI) and machine learning (ML) applications on hardware accelerator platforms (e.g. GPUs). While workflow engines and Linux containers offer a gold standard for scientific computing reproducibility, they require additional layers of training and software engineering knowledge. Modern open source multi-platform environment management tools, e.g. Pixi [1], provide automatic multi-platform digest-level lock file support for all dependencies — down to the compiler level — of software on public package indexes (e.g. PyPI [2] and conda-forge [3]) while still providing a high level interface well suited for researchers. Combined with the arrival of the full CUDA [4] stack on conda-forge, it is now possible to declaratively specify a full CUDA accelerated software environment. We are now at a point where well supported, robust technological solutions exist, even for applications with highly complex software environments. What is currently lacking is the education and training by the broader scientific software community to adopt these technologies and build community standards of practice around them, as well as an understanding of what are the most actionably useful features of adopting computational reproducibility tools.

2. Reproducibility

"Reproducible" research is a term that can mean multiple things across various fields. Some fields may view work as "reproducible" if the full process is documented, and other may

Published Jul 10, 2025

Correspondence to
Matthew Feickert
matthew.feickert@cern.ch

Open Access

Copyright © 2025 Feickert *et al.*. This is an open-access article distributed under the terms of the Creative Commons Attribution 4.0 International license, which enables reusers to distribute, remix, adapt, and build upon the material in any medium or format, so long as attribution is given to the creator.

view "reproducible" as meaning that all computations will give the same numerical outputs barring entropy variations. As there are multiple levels of reproducibility, we will restrict "reproducibility" to software environment reproducibility. We define this as be limited to the ability to define and programmatically create a software environment composed of packages that specifies all software, and its dependencies, with exact URLs and binary digests ("hashes"). Reproducible environments need to be machine agnostic in that for a specified computing platform in the environment they must be installable without modification across multiple instances.

2.1. Hardware accelerated environments

Software the involves hardware acceleration on computing resources like GPUs requires additional information to be provided for full computational reproducibility. In addition to the computer platform, information about the hardware acceleration device, its supported drivers, and compatible hardware accelerated versions of the software in the environment (GPU enabled builds) are required. While this information is straightforward to collect, traditionally this has been difficult to make use of in practice given software access restrictions and the lack of declarative human interfaces for defining relationships between system-level drivers and user software. Multiple recent technological advancements (made possible by social agreements and collaborations) in the scientific open source world now provide solutions to these problems.

3. Conda packages

Conda packages (`.conda` files) are language agnostic file archives that contain built code distributions and metadata. This is quite powerful, as it allows for arbitrary code to be built for any target platform and then packaged with its metadata. When a conda package is downloaded and then unpacked with a conda package management tool (e.g. Pixi, conda, mamba) it is then "installed" by copying the package's file directory tree to the base of the environment's directory tree. Package contents are also simple; they can only contain files and symbolic links.

3.1. conda-forge

Conda packages can be distributed on package indexes that support the concept of "channels" which redirect URLs to directory trees of conda packages. Channel names serve as the base path for hosting packages. The most broadly used community channel for conda-packages is the `conda-forge` channel, which hosts the conda packages generated from builds on the global conda-forge community cyberinfrastructure. The conda-forge community operates in a GitHub organization that hosts "feedstock" Git repositions that contain conda package build recipes as well as automation infrastructure and continuous integration (CI) and continuous delivery (CD) workflows. This allows for conda-forge community members to submit and maintain recipes — instructions for conda package build systems — to build and distribute conda packages for multiple variants of computing platforms — combinations of operating systems and hardware architectures — for Linux, macOS, and Windows.

4. CUDA

CUDA (Compute Unified Device Architecture) is a parallel computing platform and programming model developed by NVIDIA for general computing on graphical processing units (GPUs) [4], [5]. The CUDA ecosystem provides Software Development Kits (SDKs) with APIs to CUDA that allow for software developers to write hardware accelerated programs with CUDA in various languages for NVIDIA GPUs. CUDA has official language support for C++, Fortran, and Python, with community support for Julia and other languages. While there

are other types of hardware acceleration development platforms, as of 2025 CUDA is the most abundant platform for scientific computing that uses GPUs and effectively the default choice for major machine learning libraries and applications.

CUDA is closed source and proprietary to NVIDIA, which means that NVIDIA has historically limited the download access of the CUDA toolkits and drivers to registered NVIDIA developers (while keeping the software free (monetarily) to use). CUDA then required a multi-step installation process [6] with manual steps and decisions based on the target platform and particular CUDA version. This meant that when CUDA enabled environments were setup on a particular machine they were powerful and optimized, but brittle to change and could easily be broken if system wide updates (like for security fixes) occurred. CUDA software environments were bespoke and not many scientists understood how to construct and curate them.

5. CUDA PACKAGES ON CONDA-FORGE

5.1. Initial implementation

After discussion in late 2018 [7] to better support the scientific developer community, the CUDA packaging community agreed to use the Anaconda `defaults` channel's [8] `cudatoolkit` package. Initially the `cudatoolkit` package was designed around Numba's CUDA needs [9], though it evolved to a bundle of redistributable CUDA libraries. In 2019, NVIDIA began packaging the `cudatoolkit` package in the `nvidia` conda channel. With help from the broader community, the `cudatoolkit` package was added to `conda-forge` in 2020 [10]. For the first time, this provided users the *ability to specify different versions of CUDA libraries* and download them in newly created conda environments.

Supporting initial conda-forge CUDA builds required additional components:

- A conda-forge Docker image using the NVIDIA CUDA Docker images, which provided the NVIDIA build tools for compiling packages.
- A shim package to leverage the NVIDIA build tools within a conda package build.
- A CUDA build matrix in conda-forge's global pinnings, which tied these two pieces together.

These ideas were tied together in the first package build on September 20, 2019 [11], and the initial implementation of this work was completed later in 2019. In 2020, support was expanded to Windows CUDA builds. Lots of iteration on this work happened after, all using the same basic foundation.

5.2. Revised implementation

After some time using these packages and build process, a few observations became clear. First, some packages used only a subset of the libraries, like the driver, the CUDA runtime library, or particular library components like cuBLAS. However, the `cudatoolkit` package shipped considerably more than that, so having finer specifications of dependencies would provide a better package maintainer and end-user experience. Second, some packages needed components that were not part of the `cudatoolkit` bundle like other libraries or parts of the build toolchain. Having some way to depend on these components would improve usability. Third, the infrastructure management overhead of custom Docker images and their integration into the conda-forge build matrix was cumbersome for conda-forge maintainers. Being able to install and use the build tools directly would simplify maintenance and benefit end-users wishing to use these build tools.

To address these issues, NVIDIA began working on a revised set of packages. These more closely matched packages in other distribution channels (like Linux distribution package

managers) and were adapted to the conda user experience. For example, Linux distributions often install packages at the system level, which differs from the first-class userspace environment experience that conda package environments provides. As a result, some distinctions that a Linux distribution provides are unneeded in conda. There are additional differences around behavior in pinning versions of dependencies or how compilers are packaged and expected to work in their installed environments. Initial production of the packages were made on the `nvidia` channel, however, all of this work was being done internally in NVIDIA and published to a separate channel. This made the packages less visible and required additional knowledge to use.

In 2023, NVIDIA began adding the releases of CUDA conda packages from the `nvidia` channel to conda-forge, making it easier to discover and allowing for community support. Given the new package structure, NVIDIA added the packages for CUDA `12.0` to indicate the breaking change. Also with significant advancements in system driver specification support, CUDA `12` became the first version of CUDA to be released as conda packages through conda-forge and included all CUDA libraries from the CUDA compiler `nvcc` to the CUDA development libraries. CUDA metapackages were also released, which allow users to easily describe the version of CUDA they require (e.g. `cuda-version=12.5`) and the CUDA conda packages they want (e.g. `cuda`). This significantly improved the ability for researchers to easily create CUDA accelerated computing environments.

6. Pixi

6.1. Conceptual overview

Pixi is a cross-platform package and environment manager that can handle complex development workflows [1], [12]. Importantly, Pixi automatically and non-optionally will produce or update a lock file — a structured file that contains a full list of all environments defined with a complete list of all packages, as well as definition of each packages with digest information on the binary — for the software environments defined by the user whenever any actions mutate the environment. Pixi is written in Rust, and leverages the language's speed and technologies to solve environments fast.

Pixi addresses the concept of computational reproducibility by focusing on a set of main features

1. **Virtual environment management**: Pixi can create environments that contain conda packages and Python packages and use or switch between environments easily.
2. **Package management**: Pixi enables the user to install, update, and remove packages from these environments through the `pixi` command line.
3. **Task management**: Pixi has a task runner system built-in, which allows for tasks with custom logic and dependencies on other tasks to be created.

These features become powerful when combined with robust behaviors

1. **Automatic lock files**: Any changes to a Pixi workspace that can mutate the environments defined in it will automatically and non-optionally result in the Pixi lock file for the workspace being updated. This ensures that any state of a Pixi project is trivially computationally reproducible.
2. **Solving environments for other platforms**: Pixi allows the user to solve environment for platforms other than the current user machine's. This allows for users to solve and share environment to any collaborator with confidence that all environments will work with no additional setup.

3. **Pairity of conda and Python packages**: Pixi allows for conda packages and Python packages to be used together seamlessly, and is unique in its ability to handle overlap in dependencies between them. Pixi will first solve all conda package requirements for the target environment, lock the environment, and then solve all the dependencies of the Python packages for the environment, determine if there are any overlaps with the existing conda environment, and the only install the missing Python dependencies. This ensures allows for fully reproducible solves and for the two package ecosystems to compliment each other rather than potentially cause conflicts.
4. **Efficient caching**: Pixi uses an efficient global cache shared between all Pixi projects and globally installed tools on a machine. The first time Pixi installs a package it will download the files to the global cache and link the files into the environment. When Pixi has to reinstall the same package in a different environment, the package will be linked from the same cache, making sure internet bandwidth for downloads and disk space is used as efficiently as possible.

Pixi users declaratively specify their project dependencies which are recorded in a Pixi manifest `pixi.toml` file (which for Python projects can optionally be embedded in a `pyproject.toml [pixi]` table) and automatically resolved in the `pixi.lock` lock file. This declarative nature allows for users to efficiently specify their project requirements while being guaranteed a static and reproducible environment from the lock file.

6.2. CUDA hardware accelerated environment creation

Combining the features of modern CUDA `12` conda packages with Pixi's environment management, it is now possible to efficiently manage multiple software environments that can include both hardware accelerated and CPU environments. An example Pixi workspace is presented in Program 1

```toml
[workspace]
channels = ["conda-forge"]
name = "ml-example"
platforms = ["linux-64", "osx-arm64", "win-64"]
version = "0.1.0"

[tasks]

[dependencies]
python = ">=3.13.7,<3.14"

[feature.cpu.dependencies]
pytorch-cpu = ">=2.7.1,<3"
torchvision = ">=0.22.0,<0.23"

[feature.cpu.tasks.train-cpu]
description = "Train a PyTorch CNN on MNIST on CPU"
cmd = "python ./src/torch_MNIST.py --epochs 2 --save-model --data-dir data"

[feature.gpu.system-requirements]
cuda = "12"

[feature.gpu.target.linux-64.dependencies]
pytorch-gpu = ">=2.7.1,<3"
torchvision = ">=0.22.0,<0.23"

[feature.gpu.target.win-64.dependencies]
pytorch-gpu = ">=2.7.1,<3"
torchvision = ">=0.22.0,<0.23"

[feature.gpu.tasks.train-gpu]
description = "Train a PyTorch CNN on MNIST on GPU"
cmd = "python ./src/torch_MNIST.py --epochs 14 --save-model --data-dir data"

[feature.inference.dependencies]
matplotlib = ">=3.10.3,<4"

[feature.lab.dependencies]
notebook = ">=7.4.5,<8"
jupyterlab = ">=4.4.7,<5"

[feature.lab.tasks.start]
description = "Launch JupyterLab"
cmd = "jupyter lab"

[environments]
cpu = ["cpu"]
gpu = ["gpu"]
inference = ["gpu", "inference"]
lab = ["gpu", "inference", "lab"]
```

Program 1. *Example of a multi-platform and multi-environment Pixi manifest with all required information and constraints to resolve and install CUDA accelerated conda packages.*

where the definition of multiple platforms allows for solving the declared environments for all platforms while on other platforms

```toml
[workspace]
channels = ["conda-forge"]
name = "ml-example"
platforms = ["linux-64", "osx-arm64", "win-64"]
version = "0.1.0"
```

the `cpu` feature defines `dependencies` and `tasks` that are accessible from the `cpu` environment

```
...

[feature.cpu.dependencies]
pytorch-cpu = ">=2.7.1,<3"
torchvision = ">=0.22.0,<0.23"

[feature.cpu.tasks.train-cpu]
description = "Train a PyTorch CNN on MNIST on CPU"
cmd = "python ./src/torch_MNIST.py --epochs 2 --save-model --data-dir data"

...

[environments]
cpu = ["cpu"]
```

The `gpu` feature does the same for the `gpu` environment, but it also importantly defines a `system-requirements` table that define the system specifications needed to install and run a Pixi workspace's environments.

```
...

[feature.gpu.system-requirements]
cuda = "12"

[feature.gpu.target.linux-64.dependencies]
pytorch-gpu = ">=2.7.1,<3"
torchvision = ">=0.22.0,<0.23"

[feature.gpu.target.win-64.dependencies]
pytorch-gpu = ">=2.7.1,<3"
torchvision = ">=0.22.0,<0.23"

[feature.gpu.tasks.train-gpu]
description = "Train a PyTorch CNN on MNIST on GPU"
cmd = "python ./src/torch_MNIST.py --epochs 14 --save-model --data-dir data"

...

[environments]
...
gpu = ["gpu"]
```

`system-requirements` build upon the concept of conda "virtual packages", allowing for the dependency resolver to enforce constraints declared by defining compatibility of the system with virtual packages, like `__cuda`. In the particular case of CUDA, the `system-requirements` table specifies the CUDA version the workspace expects the host system to support, as detected through the host system's NVIDIA driver API. While the `system-requirements` field values do not correspond to lower or upper bounds, specifying that the workspace is expected to work on systems that support CUDA 12

```
...

[feature.gpu.system-requirements]
cuda = "12"

...
```

ensures that packages depending on `__cuda >= 12` are resolved correctly. This effectively means that declaring the system requirement will cause the Pixi dependency resolver to find CUDA enabled packages that are compatible with CUDA 12, disallowing for incompatible package builds to be resolved. Once these package dependencies have been resolved and locked, this ensures that any system capable of meeting the system requirement will get working CUDA accelerated conda packages installed.

Not all machines will have an NVIDIA GPU on them to allow for the system requirements to be resolved correctly. To allow for non-CUDA-supported-machines to still resolve Pixi workspace requirements, shell environment overrides exist through the `CONDA_OVERRIDE_CUDA` environmental variable. Setting `CONDA_OVERRIDE_CUDA=12` on a machine that doesn't meet the CUDA version requirements, will override the supported virtual packages and set a value of `__cuda=12` for the system. This can be clearly understood from setting the override and then querying the workspace summary with `pixi info`, as seen in Program 2. This is a powerful functionality as it allows for environment specification, resolution, and locking for target platforms that users might not have access to, but can be assured are valid.

```
% pixi info
System
------------
       Pixi version: 0.54.1
           Platform: osx-arm64
   Virtual packages: __unix=0=0
                   : __osx=15.3.2=0
                   : __archspec=1=m2
...

% CONDA_OVERRIDE_CUDA=12 pixi info
System
------------
       Pixi version: 0.54.1
           Platform: osx-arm64
   Virtual packages: __unix=0=0
                   : __osx=15.3.2=0
                   : __cuda=12=0
                   : __archspec=1=m2
...
```

Program 2. *Demonstration of using the `CONDA_OVERRIDE_CUDA` environmental variable on a system with no CUDA support (an Apple silicon machine) to allow dependency resolution as if it supported CUDA 12.*

Pixi also allows for feature composition to efficiently create new environments. Program 1's `gpu` and `inference` features are combined and resolved collectively to provide a new CUDA accelerated `inference` environment that does not affect the `gpu` environment. The same applies for the `lab` feature and environment, which additionally provides JupyterLab for interactive programming with notebooks.

```
...

[feature.inference.dependencies]
matplotlib = ">=3.10.3,<4"

[feature.lab.dependencies]
notebook = ">=7.4.5,<8"
jupyterlab = ">=4.4.7,<5"

...

[environments]
...
gpu = ["gpu"]
inference = ["gpu", "inference"]
lab = ["gpu", "inference", "lab"]
```

Composing multiple environments from Pixi features allows for separating conceptual steps of scientific analysis into bespoke software environments that contain only the necessary dependencies. This allows for each step's environment to be better defined, potentially with radically different or conflicting dependencies from other steps, and for clean separation between interactive and non-interactive ("batch") computing models.

6.3. Locked environments

Once the workspace has been defined, any Pixi operation on the workspace will result in all environments in the workspace having their dependencies resolved and then fully specified ("locked") at the digest ("hash") level in a single `pixi.lock` Pixi lock file, as seen in Program 3. The lock file is a YAML file that contains two definition groups: `environments` and `packages`. The `environments` group lists every environment in the workspace for every platform with a complete listing of all packages in the environment. The `packages` group lists a full definition of every package that appears in the `environments` lists, including the package's URL and digests (e.g. sha256, md5). These groups provide a full description of every package described in the Pixi workspace and its dependencies and constraints on other packages. Versioning the lock file along with the manifest file in a version control system allows for workspaces to be fully reproducible to the byte level indefinitely into the future, conditioned on the continued existence of the package indexes the workspace pulls from (e.g. conda-forge, PyPI, the nvidia conda channel). In the event that long term preservation and reproducibility are of importance, there are community projects [13] that allow for downloading all dependencies of a Pixi environment and generating a tar archive containing all of the packages, which can later be unpacked and installed.

```
version: 6
environments:
  cpu:
    channels:
    - url: https://conda.anaconda.org/conda-forge/
    packages:
      linux-64:

...

      - conda: https://conda.anaconda.org/conda-forge/linux-64/python-3.13.7-h2b335a9_100_cp313.conda
      - conda: https://conda.anaconda.org/conda-forge/noarch/python_abi-3.13-8_cp313.conda
      - conda: https://conda.anaconda.org/conda-forge/linux-64/pytorch-2.7.1-cpu_mkl_py313_h58dab0e_
103.conda
      - conda: https://conda.anaconda.org/conda-forge/linux-64/pytorch-cpu-2.7.1-cpu_mkl_hc60beec_
103.conda

...

  gpu:
    channels:
    - url: https://conda.anaconda.org/conda-forge/
    packages:
      linux-64:

...

      - conda: https://conda.anaconda.org/conda-forge/linux-64/cuda-nvcc-tools-12.9.86-he02047a_2.
conda
      - conda: https://conda.anaconda.org/conda-forge/linux-64/cuda-nvdisasm-12.9.88-hbd13f7d_0.conda
      - conda: https://conda.anaconda.org/conda-forge/linux-64/cuda-nvrtc-12.9.86-h5888daf_0.conda
      - conda: https://conda.anaconda.org/conda-forge/linux-64/cuda-nvtx-12.9.79-h5888daf_0.conda
      - conda: https://conda.anaconda.org/conda-forge/linux-64/cuda-nvvm-tools-12.9.86-h4bc722e_2.
conda
      - conda: https://conda.anaconda.org/conda-forge/noarch/cuda-version-12.9-h4f385c5_3.conda
      - conda: https://conda.anaconda.org/conda-forge/linux-64/cudnn-9.10.1.4-hbcb9cd8_1.conda

...

      - conda: https://conda.anaconda.org/conda-forge/linux-64/python-3.13.7-h2b335a9_100_cp313.conda
      - conda: https://conda.anaconda.org/conda-forge/noarch/python_abi-3.13-8_cp313.conda
      - conda: https://conda.anaconda.org/conda-forge/linux-64/pytorch-2.7.1-cuda129_mkl_py313_h1e53
aa0_304.conda
      - conda: https://conda.anaconda.org/conda-forge/linux-64/pytorch-gpu-2.7.1-cuda129_mkl_h43a4b0
b_304.conda

...

packages:

...

- conda: https://conda.anaconda.org/conda-forge/linux-64/pytorch-gpu-2.7.1-cuda129_mkl_h43a4b0b_304.
conda
  sha256: af54e6535619f4e484d278d015df6ea67622e2194f78da2c0541958fc3d83d18
  md5: e374ee50f7d5171d82320bced8165e85
  depends:
  - pytorch 2.7.1 cuda*_mkl*304
  license: BSD-3-Clause
  license_family: BSD
  size: 48008
  timestamp: 1753886159800

...
```

Program 3. *Example structure of a `pixi.lock` Pixi lock file showing the definition of the environments as well as a full description of each package used in each environment.*

7. Deploying environments to remote compute

Often researchers are running scientific and machine learning workflows on remote computational resources that use batch computing systems (e.g. HTCondor, SLURM). For systems

with shared filesystems (e.g. SLURM) it is possible to use Pixi workspaces in workflows in a similar manner to local machine (e.g. laptop or workstation). Other systems (e.g. HTCondor) do not have a shared filesystem (e.g. HTCondor), requiring that each worker node receive its own copy of the software environment. While locked Pixi environments significantly help with this, it is often advantageous to distribute the environment in the form of a Linux container image to the compute resources. These systems are able to mount Linux container images to worker nodes in ways that reduce the disk and memory cost to the user's session, compared to installing Pixi and then downloading all dependencies of the software environment from the package indexes used. This also reduces the bandwidth use as the Linux container image can be cached at the compute resource host and efficiently replicated to the worker nodes, paying the bandwidth cost of download once. While Linux container technology historically has presented additional engineering and design overhead to researchers, Linux container construction of Pixi environments is simple and can be reduced to templated format. An example in the form of a templated Dockerfile is seen in Program 4.[1] The template requires user input to define the target CUDA version (`CUDA_VERSION`) and the name of the Pixi environment to install (`ENVIRONMENT`). As the Pixi environment is already fully defined and locked it can be directly installed as normal in the `build` stage of the container image build, along with an entrypoint shell script that will activate the environment, and then copied from the `build` stage into the `final` stage to reduce the total image size by removing the cache and reducing the total number of layers in the final image.

```
/ml-example/Dockerfile
ARG CUDA_VERSION="12"
ARG ENVIRONMENT="gpu"

FROM ghcr.io/prefix-dev/pixi:noble AS build

# Redeclaring ARGS in a stage without a value inherits the global default
ARG CUDA_VERSION
ARG ENVIRONMENT

WORKDIR /app
COPY . .
ENV CONDA_OVERRIDE_CUDA=$CUDA_VERSION
RUN pixi install --locked --environment $ENVIRONMENT
RUN echo "#!/bin/bash" > /app/entrypoint.sh && \
    pixi shell-hook --environment $ENVIRONMENT -s bash >> /app/entrypoint.sh && \
    echo 'exec "$@"' >> /app/entrypoint.sh

FROM ghcr.io/prefix-dev/pixi:noble AS final

ARG ENVIRONMENT

WORKDIR /app
COPY --from=build /app/.pixi/envs/$ENVIRONMENT /app/.pixi/envs/$ENVIRONMENT
COPY --from=build /app/pixi.toml /app/pixi.toml
COPY --from=build /app/pixi.lock /app/pixi.lock
# The ignore files are needed for 'pixi run' to work in the container
COPY --from=build /app/.pixi/.gitignore /app/.pixi/.gitignore
COPY --from=build /app/.pixi/.condapackageignore /app/.pixi/.condapackageignore
COPY --from=build --chmod=0755 /app/entrypoint.sh /app/entrypoint.sh

ENTRYPOINT [ "/app/entrypoint.sh" ]
```

Program 4. *The template structure of a Dockerfile for a locked Pixi environment with CUDA dependencies. The only values that need user input are the CUDA version and the name of the target environment.*

[1] As many compute facilities do not allow for use of Docker directly given security concerns, Apptainer container image formats are more common. Apptainer definition files are similarly easy to write as compared to Dockerfiles and Docker container images can be converted into a format that Apptainer can use. As Docker is a more common format in the broader computing world, including commercial settings, it has been used for this example. These workflows are not limited to a single container image format.

The Dockerfile can then be built into a Linux container image binary file which can be distributed to a container image registry. Batch computing system workflow definition files can use these container images to provide the software environment for the computing jobs, which pull the images from the container image registry when requested by the job.

8. Summary

As hardware accelerated code becomes more common across scientific computing, especially CUDA accelerated software for machine learning, the need for simple but powerful solutions for software environment management has grown too. The simple and flexible structure of conda packages allows for complex projects to be packaged as directory trees of built binaries on a platform specific level. This has allowed for the complexity of the CUDA software stack to be efficiently built as conda packages using the conda-forge cyberinfrastructure and then distributed on the conda-forge conda channel for public use. Distribution of CUDA conda packages on conda-forge additionally allows for other conda-forge projects to use CUDA conda packages in their builds, resulting in a wide selection of CUDA enabled projects, including many machine learning packages. Through use of Pixi's declarative specification of dependencies in the project manifest and non-optional digest level lock file generation, software environments can now be declaratively and rapidly constructed, resolved, and locked using semantic operations well designed for scientific researchers. With these powerful technologies and abstractions, researchers can now construct machine learning and data science environments for multiple platforms at once and use trusted patterns to develop locally and deploy to remote computational resources.

In addition to the long term reproducibility provided by the combination of these technologies, the maintenance burden and complexity reduction should not be overlooked. With the CUDA v12 distributions on conda-forge, researches no longer need to have experience in CUDA internals and distribution installation to accelerate their software projects. They need only know the supported versions of CUDA by the NVIDIA drivers on their target machines. Researchers also no longer need to use multiple tools to build bespoke workflows for constructing and maintaining lock files for multiple environments and platforms, while keeping environment definition files and lock files synced. Pixi provides a single tool and unified interface to achieve the same results faster while using high level abstractions — removing most of the work of software environment reproducibility from the user workflow.[2] Having the full specification of the software environment including the CUDA dependencies also removes runtime failures due to missing, unspecified, or incompatible system-level requirements on remote compute resources. Most importantly, reducing cognitive overhead and the latency to reach a usable software environment reduces the time to insight for researchers, transferring the problems of scientific computing back into their domains of expertise.

Acknowledgements

Matthew Feickert is supported by the U.S. National Science Foundation (NSF) under Cooperative Agreement PHY-2323298 (IRIS-HEP) and by the US Research Software Sustainability Institute (URSSI) via grant G-2022-19347 from the Sloan Foundation. Ruben Arts is supported by prefix.dev GmbH. John Kirkham is supported by NVIDIA. The described work [14] was created in association with [15].

[2] It is worth reflecting on that the auspicious interplay of these technologies is a recent advancement that could not have happened significantly earlier. Conda-forge was created as a project in 2015 (at the SciPy 2015 conference), and the first distributions of CUDA v12 were in 2022, with Pixi being created in 2023. The advancements that have occurred in less than two years after their coexistence is a reflection of the power of strong design standards and collaboration across the conda-forge community.

References

[1] R. Arts, B. Zalmstra, W. Vollprecht, T. de Jager, N. Morcotilo, and J. Hofer, "pixi." [Online]. Available: https://github.com/prefix-dev/pixi/releases/tag/v0.54.1

[2] "The Python Package Index (PyPI)." [Online]. Available: https://pypi.org/

[3] conda-forge community, "The conda-forge Project: Community-based Software Distribution Built on the conda Package Format and Ecosystem." [Online]. Available: https://doi.org/10.5281/zenodo.4774216

[4] J. Nickolls, I. Buck, M. Garland, and K. Skadron, "Scalable parallel programming with CUDA," in *ACM SIGGRAPH 2008 Classes*, in SIGGRAPH '08. Los Angeles, California, 2008. doi: 10.1145/1401132.1401152.

[5] M. Fatica, "CUDA toolkit and libraries." [Online]. Available: https://doi.org/10.1109/HOTCHIPS.2008.7476520

[6] "CUDA Installation Guide for Linux." [Online]. Available: https://docs.nvidia.com/cuda/cuda-installation-guide-linux/

[7] "conda-forge.github.io Issue 687: How to specify CUDA version in a conda package?." [Online]. Available: https://github.com/conda-forge/conda-forge.github.io/issues/687

[8] Anaconda, Inc., "Anaconda Documentation: Default channels." (https://www.anaconda.com/docs/tools/working-with-conda/reference/default-channels, 2025.

[9] Continuum Analytics, Inc., "numba/conda-recipe-cudatoolkit." [Online]. Available: https://github.com/numba/conda-recipe-cudatoolkit

[10] "conda-forge/staged-recipes Pull Request 12882: Cudatoolkit." [Online]. Available: https://github.com/conda-forge/staged-recipes/pull/12882

[11] "conda-forge/ucx-split-feedstock Pull Request 14: Add GPU builds." [Online]. Available: https://github.com/conda-forge/ucx-split-feedstock/pull/14

[12] R. Arts, B. Zalmstra, W. Vollprecht, T. de Jager, N. Morcotilo, and J. Hofer, "Pixi Documentation." [Online]. Available: https://pixi.sh/v0.54.1/

[13] P. Zwerschke, D. Elsner, and B. Stoyan, "pixi-pack." [Online]. Available: https://github.com/Quantco/pixi-pack/releases/tag/v0.7.2

[14] M. Feickert, R. Arts, and J. Kirkham, "SciPy 2025 Tutorial: Reproducible Machine Learning Workflows for Scientists with Pixi." [Online]. Available: https://github.com/matthewfeickert-talks/reproducible-ml-for-scientists-with-pixi-scipy-2025/releases/tag/scipy-2025

[15] M. Feickert, "Reproducible Machine Learning Workflows for Scientists." [Online]. Available: https://github.com/carpentries-incubator/reproducible-ml-workflows

Open Access | https://doi.org/10.25080/cehj4257

SciPy 2025

July 7 - July 13, 2025

Proceedings of the 24th Python in Science Conference
ISSN: 2575-9752

NeuroConv: Streamlining Neurophysiology Data Conversion to the NWB Standard

Heberto Mayorquin[1], **Cody Baker**[2], **Paul Adkisson-Floro**[1], **Szonja Weigl**[1], **Alessandra Trappani**[1], **Luiz Tauffer**[1], **Oliver Rübel**[3], and **Benjamin Dichter**[1]

[1]CatalystNeuro, [2]Dartmouth College, Center for Open Neuroscience, [3]Lawrence Berkeley National Laboratory

Abstract

Modern neurophysiology generates increasingly complex, multimodal datasets that require standardized formats for effective sharing and reuse. The Neurodata Without Borders (NWB) format has emerged as a solution for data standardization, but data conversion remains a significant bottleneck due to format heterogeneity, metadata complexity, and required technical expertise. We present NeuroConv, an open-source Python library that automates the conversion of neurophysiology data from 47 distinct formats into NWB through a unified, modular architecture. Developed through collaboration with over 50 neurophysiology laboratories, NeuroConv addresses key challenges through three core components: format-specific DataInterfaces that abstract parsing complexity, multi-stream Converters that integrate heterogeneous data modalities, and optimized writing strategies for large-scale datasets including chunked operations and cloud-compatible storage. NeuroConv's design enables researchers to convert complex, multi-modal experimental sessions with minimal code while preserving critical metadata and temporal alignment across recording systems. By removing technical barriers NeuroConv thus advances the transformation of neurophysiology toward FAIR (Findable, Accessible, Interoperable, and Reusable) data practices, facilitating reproducible research and accelerating scientific discovery.

Keywords Neurodata without borders, Neurophysiology, Data standardization, Scientific software

Published Jul 10, 2025

Correspondence to
Heberto Mayorquin
h.mayorquin@gmail.com

Open Access

Copyright © 2025 Mayorquin *et al.*. This is an open-access article distributed under the terms of the Creative Commons Attribution 4.0 International license, which enables reusers to distribute, remix, adapt, and build upon the material in any medium or format, so long as attribution is given to the creator.

1. Introduction

Modern neurophysiology produces complex, multimodal, large-scale datasets. Sharing this data has potential to accelerate the rate of scientific progress, but doing so effectively requires disciplined organization, annotation, and distribution to make it Findable, Accessible, Interoperable, and Reusable (FAIR) [1], [2]. However, the neurophysiology community faces significant challenges with sharing native data formats: the field's diversity, spanning optical microscopy, electrophysiology, and behavioral tracking, involves multiple acquisition systems with formats often optimized for rapid data capture rather than shareability or portability. These formats vary widely in efficiency, metadata completeness and longevity of support, creating substantial barriers for data sharing [3], [4], [5].

The Neurodata Without Borders (NWB) format addresses these challenges and has become a leading solution, embedding rich contextual metadata directly alongside primary data to preserve interpretability over time [3], [5]. Crucially, NWB can accommodate the field's diversity by uniting disparate modalities such as simultaneous behavioral and electrophysiological recordings typically stored in separate files and formats, thereby enabling rigorous

re-analysis and bolstering reproducibility. Additionally, NWB architecture provides compression and chunking capabilities enabling efficient storage and data access even for large scale datasets.

Yet several factors still hinder broad NWB adoption. Converting neurophysiology data to any standardized format requires substantial technical expertise, from understanding diverse source format specifications to mastering the programming interfaces needed for data manipulation (PyNWB/MatNWB). Modern experimental setups amplify these challenges, as neurophysiology experiments now routinely combine multiple modalities (electrophysiology, imaging, optogenetics, behavior) each with unique format quirks, metadata particularities and synchronization needs. These technical barriers and resource requirements create a significant hurdle for NWB adoption.

Automated conversion pipelines address these challenges by eliminating the need for laboratories to repeatedly solve the same technical problems. Rather than requiring each research group to develop custom conversion scripts, automated tools provide standardized, tested pathways from diverse source formats to NWB. This approach not only prevents duplicated effort and inconsistent implementations across the field but also democratizes access to data standardization, ensuring that all laboratories can preserve and share their data regardless of available technical expertise. By lowering the barriers to NWB adoption, automated pipelines enable the neuroscience community to sustainably build a collective repository of standardized, reusable experimental data, ultimately accelerating scientific discovery through enhanced data sharing and reproducibility.

Table 1. *Key challenges in building automated conversion pipelines to the NWB format.*

Challenge	Description
Format diversity	Source data formats span both proprietary and open standards, with many formats existing in multiple versions and internal variations
Metadata complexity	Metadata requirements vary substantially across experimental paradigms, with critical information often stored inconsistently or incompletely
Scale challenges	Dataset sizes frequently reach hundreds of gigabytes to terabytes, requiring specialized handling for memory-efficient processing
Multi-modal integration	Experiments often combine multiple recording systems (electrophysiology, imaging, behavior) each storing data in different formats. This also introduces the problem of data alignment and synchronization across modalities.

2. NeuroConv: Architecture and Design

To address the challenges described in Introduction (see Table 1), we developed NeuroConv, an open source library that automates the ingestion and conversion of neurophysiology data from diverse formats into NWB. This section describes in detail how NeuroConv's architecture addresses each of these challenges through a modular, extensible design that maintains both flexibility and ease of use.

2.1. Handling Format Diversity

The challenge of format diversity in neurophysiology extends beyond their sheer number (47 supported at the moment). Many formats, such as Neuralynx, have multiple versions, while others, like TIFF, exhibit significant internal variability in how labs use them. NeuroConv addresses this complexity through a unified architecture centered on the DataInterface abstraction. DataInterface is an abstract class for reading data, and each supported format has a dedicated DataInterface that encapsulates the format-specific logic for data

reading, metadata extraction, and NWB conversion while presenting a consistent API to users. The DataInterface serves as the fundamental building block of NeuroConv, providing a standardized pathway from diverse source formats to NWB output. This abstraction enables users to work with any supported format using identical code patterns, regardless of the underlying format complexity or internal details. The minimal conversion pipeline is illustrated in Figure 1:

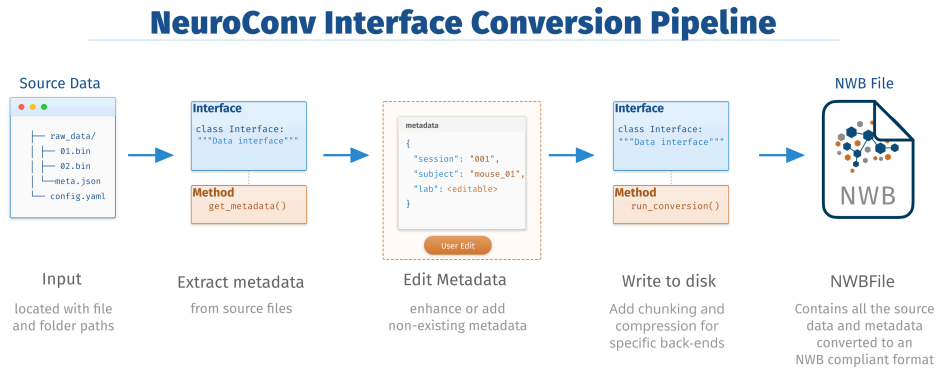


Figure 1. *The process begins with source data (e.g., binary recordings, metadata, and configuration files). A data-specific DataInterface object is instantiated and used to extract metadata. The resulting metadata can be optionally edited by the user to fill in missing or corrected fields. The finalized metadata and source data are then processed to write a complete NWB file compliant with the standard*

Programmatically, this process can be summarized in a few lines of code, as shown below. The DataInterface handles all format-specific complexities internally, from parsing proprietary binary structures to extracting embedded metadata, while ensuring the output adheres to NWB best practices:

```
from datetime import datetime
from zoneinfo import ZoneInfo
from neuroconv import DataInterface

# Initialize the DataInterface for a specific format
interface = DataInterface(file_path="path/to/data/file")

# Extract the metadata from the source format
metadata = interface.get_metadata()

# Modify metadata as needed, add missing or correct existing fields
metadata["NWBFile"]["experimenter"] = ["Baggins, Bilbo"]
metadata["NWBFile"]["experiment_description"] = "Example neurophysiology experiment"
metadata["NWBFile"]["institution"] = "University of Middle Earth"
# session_start_time is required for conversion
metadata["NWBFile"]["session_start_time"] = datetime(2020, 1, 1, 12, 30, 0, tzinfo=ZoneInfo("Middle-
earth/Shire"))

# Add the data to an NWB File and write it to disk
interface.run_conversion(nwbfile_path="path/to/nwbfile.nwb", metadata=metadata)
```

This core pattern ensures consistency and simplicity across all supported formats:

1. Instantiate the appropriate DataInterface
2. Extract and modify metadata as needed
3. Execute the conversion

The DataInterface abstracts away format-specific complexities: from parsing proprietary binary structures to extracting embedded metadata.

Currently supporting 47 distinct input formats (Table 2), each DataInterface is comprehensively documented and demonstrated in the Conversion Gallery, where users can find

complete examples requiring only ∼5 lines of code to perform full data conversion. Throughout the conversion process, NeuroConv enforces NWB Best Practices for metadata and data organization while optimizing data storage for both archival purposes and cloud computing requirements.

Table 2. *Comprehensive list of data formats supported by NeuroConv, organized by experimental modality and data type.*

Modality	Sub-modality	Format
Extracellular Electrophysiology	*Recording*	AlphaOmega
		Axona
		Biocam
		Blackrock
		European Data Format (EDF)
		Intan
		MaxOne
		MCSRaw
		MEArec
		Neuralynx
		NeuroScope
		OpenEphys
		Plexon
		Plexon2
		Spike2
		Spikegadgets
		SpikeGLX
		Tucker-Davis Technologies (TDT)
		White Matter
	Sorting	Blackrock
		Cell Explorer ([6])
		KiloSort ([7])
		Neuralynx
		NeuroScope
		Phy
		Plexon
Intracellular Electrophysiology	—	ABF

Optical Physiology	*Imaging*	Bruker
		HDF5
		Micro-Manager
		Miniscope ([8])
		Scanbox
		ScanImage
		Thor
		Tiff
	Segmentation	CaImAn ([9])
		CNMFE ([10])
		EXTRACT ([11])
		Minian ([12])
		Suite2P ([13])
	Fiber Photometry	TDT Fiber Photometry
Behavior	*Motion Tracking*	DeepLabCut ([14])
		FicTrac ([15])
		LightningPose ([16])
		Neuralynx NVT
		SLEAP ([17])
	Audio/Video	Videos
	Operant Conditioning	MedPC
General Data	*Image*	Image (png, jpeg, tiff, etc)
	Text/Tabular	CSV
		Excel

Each format example in our documentation includes basic code snippets that demonstrate how to use the DataInterface for that specific format, including metadata extraction, modification, and the conversion process. This modular approach allows users to easily adapt examples to their specific needs while providing a consistent interface across different data formats. For example, converting electrophysiology data acquired with Intan requires only these steps:

```python
from datetime import datetime
from zoneinfo import ZoneInfo
from pathlib import Path
from neuroconv.datainterfaces import IntanRecordingInterface

file_path = f"{ECEPHY_DATA_PATH}/intan/intan_rhd_test_1.rhd" # This can also be .rhs
interface = IntanRecordingInterface(file_path=file_path, verbose=False)

# Extract what metadata we can from the source files
metadata = interface.get_metadata()
# session_start_time is required for conversion. If it cannot be inferred
# automatically from the source files you must supply one.
session_start_time = datetime(2020, 1, 1, 12, 30, 0, tzinfo=ZoneInfo("US/Pacific"))
metadata["NWBFile"].update(session_start_time=session_start_time)

nwbfile_path = f"{path_to_save_nwbfile}"  # This should be something like: "./saved_file.nwb"
interface.run_conversion(nwbfile_path=nwbfile_path, metadata=metadata)
```

Supporting diverse data formats each with unique specifications, quirks, and version variations, represents a formidable technical challenge. To address this complexity, NeuroConv adopts a strategic approach: we leverage the collective expertise of the open-source neuroscience community by building upon established, well-tested libraries for format parsing. This philosophy recognizes that the neuroscience community has already invested considerable effort in developing robust parsers for specific data domains, and rather than duplicating this substantial work, we can build upon these proven foundations to create a more reliable and maintainable conversion ecosystem.

While we maintain specialized extractors for some formats (DeepLabCut, MedPC, FicTrac, etc), the majority of our DataInterfaces delegate the complex task of format parsing to domain-specific tools that have been extensively validated through widespread community use. DataInterfaces for electrophysiology utilize python-NEO [18] and SpikeInterface [19], which provide unified interfaces for raw and processed electrophysiology data, and ProbeInterface [20] for probe geometry metadata. For optical physiology, the NeuroConv team maintains ROIExtractors, which provides a unified API for optical physiology data across both raw imaging and processed segmentation outputs. ROIExtractors relies heavily on established libraries such as TiffFile [21]. Examples in behavioral data include the use of sleap-io for reading pose estimation data from SLEAP, SciPy [22] for audio, and OpenCV [23] for video.

Crucially, NeuroConv operates as an active contributor rather than a passive consumer of these dependencies. As our extensive usage of these libraries reveals bugs, performance bottlenecks, or functional limitations, we proactively engage with upstream developers by filing detailed, reproducible issue reports, submitting pull requests with tested fixes, contributing regression tests to prevent future breakages, and participating constructively in release planning discussions. This reciprocal development workflow ensures that improvements discovered during NWB conversion efforts propagate upstream, strengthening the broader neuroscience software ecosystem while continuously enhancing NeuroConv's own reliability and robustness.

As NeuroConv's format support has expanded to 47+ formats, dependency management has become increasingly complex. Each format often requires specialized libraries with potentially conflicting version requirements, creating dependency resolution challenges that can make installation difficult or impossible. For example, libraries for reading different formats may require different versions of Python or different versions of common dependencies such as numpy. Additionally, installing the dependencies of every DataInterface would create an unnecessary and inefficient environment with hundreds of packages, many of which users never need for their conversion.

To address these challenges, we rely on installation extras to manage installation complexity. Users can specify only the formats they need during installation:

```
pip install "neuroconv[spikeglx,phy,deeplabcut]"
```

This approach aggregates only the required dependencies for selected formats, avoiding dependency conflicts while ensuring that the installation remains lightweight and manageable for end users. Users working with a specific subset of formats can maintain clean, minimal environments without the overhead of unused dependencies.

2.2. Multi Stream Conversions

Neurophysiology experiments typically involve multiple simultaneous data streams from different modalities, such as raw electrophysiology recordings, spike-sorted data, and behavioral video. Each stream may be stored in a different format, leading to complex conversion requirements. NeuroConv's architecture supports multi-stream conversions through the aggregation of DataInterfaces within a Converter framework as illustrated Figure 2

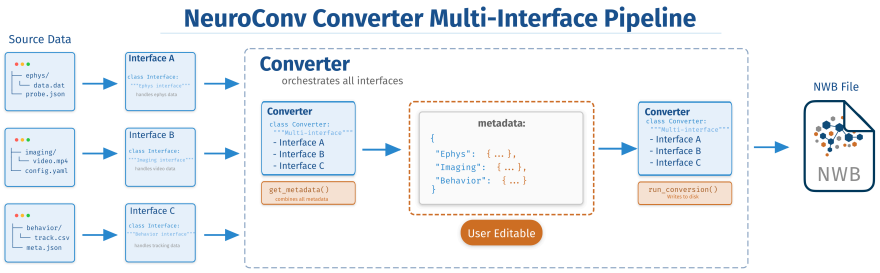


Figure 2. *The Converter orchestrates multiple specialized interfaces (A, B, C), each handling different data types (electrophysiology, imaging, and behavior). Individual interfaces extract metadata from their respective data sources, which the Converter combines into a single, user-editable metadata structure. The Converter then coordinates all interfaces to produce a unified NWB file containing all data modalities. This design pattern enables flexible, modular integration of heterogeneous neuroscience data into a single standardized format.*

The converter pattern enables combining multiple DataInterface instances into a single conversion workflow. This allows users to convert all relevant data streams from an experiment into a single NWB file, ensuring that all data is properly aligned and associated with the correct metadata. The Converter class handles the orchestration of multiple DataInterfaces, managing the order of operations and resolving any conflicts in metadata or data organization. Here's an example of conversion for a multi-modal experimental session:

```python
from datetime import datetime
from zoneinfo import ZoneInfo
from neuroconv import ConverterPipe
from neuroconv.datainterfaces import SpikeGLXRecordingInterface, PhySortingInterface,
DeepLabCutInterface

# Initialize the DataInterfaces for each data stream
recording_interface = SpikeGLXRecordingInterface(file_path="path/to/recording/file")
sorting_interface = PhySortingInterface(file_path="path/to/sorting/file")
behavior_interface = DeepLabCutInterface(file_path="path/to/behavior/file")

# Create the ConverterPipe with the DataInterfaces
data_interfaces = [recording_interface, sorting_interface, behavior_interface]
converter = ConverterPipe(data_interfaces=data_interfaces)

# Extract metadata from all interfaces
metadata = converter.get_metadata()

# Modify metadata as needed, add missing or correct existing fields
metadata["NWBFile"]["experimenter"] = ["Baggins, Bilbo"]
metadata["NWBFile"]["experiment_description"] = "Multi-modal neurophysiology experiment"
metadata["NWBFile"]["institution"] = "University of Middle Earth"
# session_start_time is required for conversion
metadata["NWBFile"]["session_start_time"] = datetime(2020, 1, 1, 12, 30, 0, tzinfo=ZoneInfo("Middle-
earth/Shire"))

# Run the conversion to create an NWB file
converter.run_conversion(nwbfile_path="path/to/nwbfile.nwb", metadata=metadata)
```

Note that the same pattern used for single interfaces extends seamlessly to multi-stream conversions. The user initializes multiple DataInterfaces for each data stream, aggregates them into a ConverterPipe, extracts metadata, modifies it as needed, and then runs the conversion process to create an NWB file. This modular approach allows NeuroConv to handle complex experimental setups with multiple data streams while maintaining a consistent interface for users.

2.3. Multi-modal time synchronization

Precise temporal alignment across diverse recording modalities is essential for accurate multi-modal data analysis and reproducibility. NeuroConv streamlines this critical process by providing intuitive, unified methods for time synchronization, leveraging common temporal references like hardware clocks or synchronization pulses. NeuroConv provides convenience functions for extracting times from pulse signals, and for aligning time using a single starting pulse or for regular pulses sent between systems. These alignment strategies can correct for differences in starting time and for temporal drift between systems, and work for systems with very different sampling rates. This automation enforces best practices for temporal metadata in NWB and enhances downstream analysis integrity.

2.4. Handling High-Volume Data

Modern acquisition systems, such as multi-probe Neuropixel recordings or whole-brain optical imaging, generate massive volumes of data that continue to grow year over year [24], [25], [26]. These volumes of data pose a variety of challenges both for conversion and for long-term storage. Moreover, as cloud computing emerges as a solution [27], [28] for managing and storing large datasets, ensuring efficient accessibility for the scientific community becomes a critical consideration.

A critical feature is the ability to process datasets larger than available RAM. NeuroConv inherits from the work performed by the NWB core group with iterative writing to stream data in manageable chunks, with configurable buffer sizes based on available resources. We have extended this approach to support chunked reading that allows buffered writing

of large datasets. This approach enables processing of arbitrarily large files and has been successfully tested on 100+ GB files using computers with only 8 GB of RAM.

For storage optimization, NeuroConv leverages chunking and compression in HDF5 and Zarr, the currently supported backends in NWB. Chunking allows large datasets to be read in manageable chunk sizes, and lossless compression algorithms allow us to reduce file size without altering the values of the data. There are many options for compression algorithms, and they present a trade-off between storage space and access speed [29]. NeuroConv exposes an easy-to-use API for configuring chunking and compression settings at the dataset level, allowing for quick experimentation while providing sensible defaults that work for most users.

Determining optimal chunk parameters involves complex tradeoffs [30]. Large chunks minimize the number of read operations but may require decompressing unnecessary data when accessing small subsets. Small chunks provide more precise access but increase overhead, particularly for cloud storage where each chunk requires a separate HTTP range request. Generally, appropriate chunking requires understanding the most common data access patterns. By understanding common analysis workflows and visualization patterns, it becomes feasible to implement evidence-based heuristics for chunk sizing across common data types, such as voltage recordings and imaging datasets.

2.5. Cloud Deployment

NeuroConv supports both local installation (Linux, Windows, or macOS) and cloud deployment through a maintained Docker image containing all dependencies. We've developed a YAML-based specification language for defining conversion pipelines, validated through JSON schema. This specification can fully describe multi-subject, multi-session conversions with custom metadata at each level, enabling automated conversion through containerized NeuroConv deployments.

2.6. Testing and Quality Assurance

Ensuring reliable conversion across diverse neurophysiology data formats requires a robust testing infrastructure. Our testing framework rests on two pillars: an automated continuous integration (CI) pipeline built using GitHub Actions and a comprehensive test data library.

Our continuous integration pipeline ensures code quality and maintains compatibility across operating systems through standard software engineering practices. The pipeline runs on every pull request and includes several key components: unit tests covering the core functionality of the library; integration tests ensuring the library works as expected with real data; cross-platform testing to verify compatibility across all supported operating systems; documentation builds to ensure up-to-date documentation; doctest functionality to verify that our conversion gallery works with the current version of the code, preventing documentation drift; code style checks ensuring consistency and adherence to best practices; and test coverage analysis to ensure the code is well-tested and maintainable. The code coverage of NeuroConv stands at 90%, which exceeds standard practices [31], [32]. Failed tests block pull request merging, maintaining code quality standards while facilitating rapid development.

Our test data infrastructure comprises carefully curated libraries spanning all supported formats, organized across three specialized domains: extracellular electrophysiology (developed in collaboration with NEO and SpikeInterface teams), optical physiology, and behavior. Each library contains representative files selected to encompass common usage patterns, format variations, edge cases, and diverse experimental configurations. This comprehensive coverage ensures that our testing captures real-world scenarios, from

legacy format versions to modern acquisition systems with missing data streams or unconventional metadata structures. For continuous integration, we leverage G-Node's git-annex technology for efficient large file management, maintaining lightweight repositories while providing full access to the test datasets. During continuous integration, GitHub Actions automatically downloads and caches these libraries, ensuring tests execute with current data while minimizing transfer overhead and maintaining rapid build times.

3. NWB Community and Software Ecosystem

3.1. NeuroConv and the NWB Conversion Landscape

NeuroConv occupies a strategic position within the broader NWB ecosystem, bridging the gap between low-level programming interfaces and high-level user tools. The conversion landscape offers different approaches suited to varying user needs and technical expertise levels.

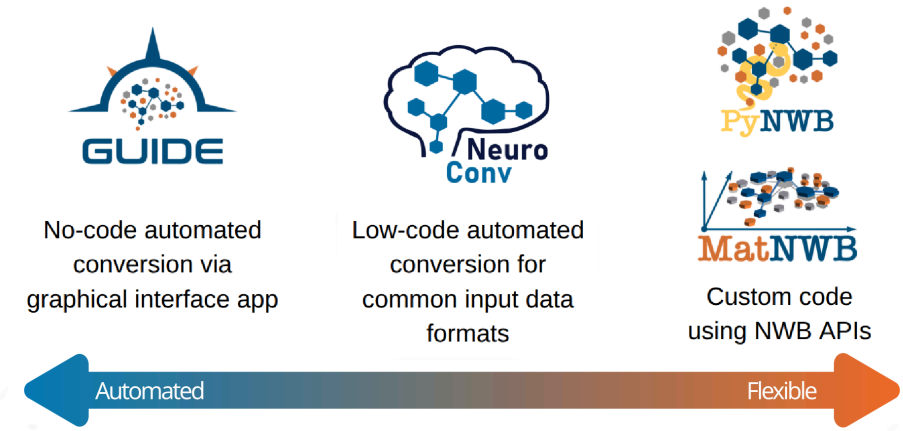


Figure 3. *This illustrates where NeuroConv stands in regard to other conversion tools. For low-level, high-precision control, users can employ the NWB APIs directly (PyNWB in Python, MatNWB in MATLAB). For a guided GUI-based experience, the NWB GUIDE offers the best option but may be too rigid for complex workflows. NeuroConv stands as a middle ground, automating the conversion of a large number of formats while still allowing for customization and flexibility.*

At the foundational level, PyNWB and MatNWB provide direct programmatic access to the NWB specification, offering maximum flexibility but requiring deep understanding of the standard. NWB GUIDE offers a graphical interface that democratizes access to NWB conversion through guided workflows, making it accessible to users without programming experience. NeuroConv complements both approaches by providing programmatic automation while maintaining the flexibility needed for complex, multi-modal experimental setups. This ecosystem approach ensures that researchers can choose the conversion method that best matches their technical expertise and experimental complexity, while all approaches converge on the same standardized output format.

3.2. Integration with Data Archives and Visualization

The Distributed Archives for Neurophysiology Data Integration (DANDI) Archive complements NWB by providing free hosting for terabyte-scale datasets, letting researchers satisfy NIH data-sharing mandates while archiving data in a versioned, API-driven repository connected to a rich ecosystem of visualization and analysis tools. NeuroConv integrates tightly with DANDI through automated upload functionality that validates required metadata, enforces DANDI naming conventions, and automatically uploads NWB files to the archive.

Together, NWB, NeuroConv, and DANDI showcase a comprehensive pipeline that spans the entire data lifecycle from acquisition to publication and reuse, supporting emerging software domains from electrophysiological spike sorting to calcium imaging segmentation and behavioral pose estimation.

4. Current Limitations

While NeuroConv has significantly improved data standardization processes, some major challenges remain:

Format Coverage: Despite supporting 47 formats, the neurophysiology ecosystem continues to evolve with new acquisition systems, format versions, and processing software. Each new system often introduces proprietary formats with unique metadata structures and data organization schemes. While NeuroConv's modular architecture allows users to develop custom DataInterfaces for unsupported formats, this process requires substantial technical expertise in both the source format's internal structure and NeuroConv's interface architecture. Additionally, maintaining custom interfaces as both the source format and NeuroConv evolve presents ongoing maintenance challenges for individual laboratories.

Custom Laboratory Formats: Many laboratories develop custom data storage solutions tailored to their specific experimental workflows, often utilizing MATLAB files, custom CSV schemas, or bespoke binary formats. These custom formats frequently exhibit significant variability not only between laboratories but even within the same laboratory over time as experimental protocols evolve. The heterogeneous nature of these formats, ranging from simple tabular data to complex nested structures with laboratory-specific metadata conventions, makes automated conversion particularly challenging. While NeuroConv can handle some standardized custom formats, the diversity of these approaches often requires manual intervention or custom code development. For these cases, we provide documentation for building custom DataInterfaces that build upon our established abstractions and integrate directly with other DataInterfaces in multi-modal Converter workflows. Since users understand their own data formats best, they are well-positioned to develop these interfaces. This approach is particularly valuable for behavioral data where standardization remains elusive [33].

Metadata Completeness: While NeuroConv automatically extracts available metadata from source formats, many acquisition systems store incomplete or inconsistent metadata. Critical information such as experimental conditions, subject details, or calibration parameters may be missing, incorrectly formatted, or stored in non-standard locations. This limitation requires manual metadata curation and validation, which can be time-consuming for large datasets. Additionally, some formats store metadata in proprietary or undocumented structures that may not be fully accessible through existing parsing libraries.

Complex Experimental Paradigms: Highly specialized experimental setups such as those involving custom-built hardware, novel stimulation protocols, or unique behavioral paradigms, may not map directly onto NWB's data organization schemes. While NWB provides extension mechanisms for custom data types, leveraging these extensions through NeuroConv requires additional development effort and deep understanding of the NWB specification. These edge cases often require manual intervention or custom conversion pipelines that may not be easily generalizable to other laboratories.

Programming Prerequisites: While NeuroConv substantially reduces the coding burden, it still requires basic programming knowledge, including object-oriented concepts. While we aim to make the library as user-friendly as possible, users must still be comfortable with Python programming to effectively utilize NeuroConv. This includes understanding how to

install the library, manage dependencies, and write scripts that leverage the DataInterface classes for conversion.

5. Future Directions

Enhanced User Experience: We are implementing comprehensive improvements to examples, tutorials, and documentation. We aim to slowly transition to the Diataxis [34] framework which will provide clearer learning pathways for users with different backgrounds and goals.

NWB Standard Evolution: We actively participate in NWB schema development and maintain close alignment with emerging standards through engagement with NWB Extensions Proposals. By monitoring and contributing to current developments, including enhanced schemas for experimental events (NWBEP001), extracellular electrophysiology (NWBEP002), and optical physiology (NWBEP003, NWBEP004), we ensure that NeuroConv incorporates the latest standard improvements as they become available for the benefit of our users.

AI-Assisted Conversion: We are exploring large language model integration to advance our core mission of automating neurophysiology data conversion. More concretely, we are developing an LLM-powered conversion agent that uses retrieval-augmented generation over neuroconv documentation and curated examples to synthesize NWB-compliant conversion code and validate the results with nwb-inspector. Grounded in verifiable sources, the agent streamlines converter creation while minimizing hallucinations and ensuring reproducibility.

Cloud-Optimized Storage: We are implementing improved chunking patterns and compression strategies for large datasets to enhance cloud access performance. This includes systematic experimentation through the NWB Benchmarks Project to determine optimal chunk sizes and compression algorithms [29]. Our goal is to implement evidence-based heuristics that ensure efficient and performant data storage, as well as having a friendly and clear API that allows customization and flexibility for users to adapt to their specific needs. In the future, we would also like to explore the inclusions of cost optimization considerations for cloud storage [35].

6. Conclusion

We believe that NeuroConv represents a critical step toward realizing the vision of FAIR neurophysiology data. By automating the conversion of diverse data formats into a common standard, we enable researchers to focus on scientific discovery rather than data wrangling. The success of this approach depends not only on technical implementation but on fostering a community that values standardization, reproducibility, and open science.

Our work demonstrates that effective scientific software development requires balancing automation with flexibility, standardization with customization, and ease of use with powerful capabilities. The challenges we have addressed such as format diversity, metadata complexity, and scale are not unique to neurophysiology but represent broader issues in scientific computing that require community-driven solutions.

The broader implications extend beyond neurophysiology to any scientific domain grappling with data heterogeneity and the need for standardization. We believe that our approach of abstracting format complexity through unified interfaces, while maintaining extensibility through modular architecture, provides a template for similar challenges in other fields.

As the neurophysiology community continues to generate increasingly complex and large datasets, tools like NeuroConv become essential infrastructure for scientific progress. By lowering barriers to data standardization and sharing, we contribute to a future where scientific data is truly FAIR, accelerating discovery and enhancing reproducibility across the field.

References

[1] M. D. Wilkinson *et al.*, "The FAIR Guiding Principles for scientific data management and stewardship," *Scientific data*, vol. 3, no. 1, pp. 1–9, 2016, doi: 10.1038/sdata.2016.18.

[2] M. E. Martone, "The past, present and future of neuroscience data sharing: a perspective on the state of practices and infrastructure for FAIR," *Frontiers in Neuroinformatics*, vol. 17, p. 1276407, 2024, doi: 10.3389/fninf.2023.1276407.

[3] J. L. Teeters *et al.*, "Neurodata without borders: creating a common data format for neurophysiology," *Neuron*, vol. 88, no. 4, pp. 629–634, 2015.

[4] M. Stead and J. J. Halford, "Proposal for a standard format for neurophysiology data recording and exchange," *Journal of Clinical Neurophysiology*, vol. 33, no. 5, pp. 403–413, 2016.

[5] O. Rübel *et al.*, "The Neurodata Without Borders ecosystem for neurophysiological data science," *Elife*, vol. 11, p. e78362, 2022.

[6] P. C. Petersen, J. H. Siegle, N. A. Steinmetz, S. Mahallati, and G. Buzsáki, "CellExplorer: A framework for visualizing and characterizing single neurons," *Neuron*, vol. 109, no. 22, pp. 3594–3608, 2021, doi: 10.1016/j.neuron.2021.09.002.

[7] M. Pachitariu, S. Sridhar, J. Pennington, and C. Stringer, "Spike sorting with Kilosort4," *Nature Methods*, vol. 21, no. 5, pp. 914–921, 2024, doi: 10.1038/s41592-024-02232-7.

[8] C. Guo *et al.*, "Miniscope-LFOV: a large-field-of-view, single-cell-resolution, miniature microscope for wired and wire-free imaging of neural dynamics in freely behaving animals," *Science advances*, vol. 9, no. 16, p. eadg3918, 2023, doi: 10.1126/sciadv.adg3918.

[9] A. Giovannucci *et al.*, "CaImAn an open source tool for scalable calcium imaging data analysis," *elife*, vol. 8, p. e38173, 2019.

[10] E. A. Pnevmatikakis *et al.*, "Simultaneous denoising, deconvolution, and demixing of calcium imaging data," *Neuron*, vol. 89, no. 2, pp. 285–299, 2016.

[11] H. Inan, M. A. Erdogdu, and M. Schnitzer, "Robust estimation of neural signals in calcium imaging," *Advances in neural information processing systems*, vol. 30, 2017.

[12] Z. Dong *et al.*, "Minian, an open-source miniscope analysis pipeline," *Elife*, vol. 11, p. e70661, 2022, doi: 10.7554/eLife.70661.

[13] M. Pachitariu *et al.*, "Suite2p: beyond 10,000 neurons with standard two-photon microscopy," *BioRxiv*, p. 61507, 2016.

[14] A. Mathis *et al.*, "DeepLabCut: markerless pose estimation of user-defined body parts with deep learning," *Nature neuroscience*, vol. 21, no. 9, pp. 1281–1289, 2018.

[15] R. J. Moore, G. J. Taylor, A. C. Paulk, T. Pearson, B. van Swinderen, and M. V. Srinivasan, "FicTrac: a visual method for tracking spherical motion and generating fictive animal paths," *Journal of neuroscience methods*, vol. 225, pp. 106–119, 2014, doi: 10.1016/j.jneumeth.2014.01.010.

[16] D. Biderman *et al.*, "Lightning Pose: improved animal pose estimation via semi-supervised learning, Bayesian ensembling and cloud-native open-source tools," *Nature Methods*, vol. 21, no. 7, pp. 1316–1328, 2024.

[17] T. D. Pereira *et al.*, "SLEAP: A deep learning system for multi-animal pose tracking," *Nature methods*, vol. 19, no. 4, pp. 486–495, 2022.

[18] S. Garcia *et al.*, "Neo: an object model for handling electrophysiology data in multiple formats," *Frontiers in Neuroinformatics*, vol. 8, p. 10, 2014, doi: https://doi.org/10.3389/fninf.2014.00010.

[19] A. P. Buccino *et al.*, "SpikeInterface, a unified framework for spike sorting," *eLife*, vol. 9, p. e61834, 2020, doi: https://doi.org/10.7554/eLife.61834.

[20] S. Garcia, J. Sprenger, T. Holtzman, and A. P. Buccino, "ProbeInterface: a unified framework for probe handling in extracellular electrophysiology," *Frontiers in Neuroinformatics*, vol. 16, p. 823056, 2022, doi: https://doi.org/10.3389/fninf.2022.823056.

[21] Christoph Gohlke, "cgohlke/tifffile: v2025.10.16." [Online]. Available: https://zenodo.org/doi/10.5281/zenodo.6795860

[22] P. Virtanen *et al.*, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020, doi: https://doi.org/10.1038/s41592-019-0686-2.

[23] G. Bradski, "The opencv library.," *Dr. Dobb's Journal: Software Tools for the Professional Programmer*, vol. 25, no. 11, pp. 120–123, 2000.

[24] N. A. Steinmetz, C. Koch, K. D. Harris, and M. Carandini, "Challenges and opportunities for large-scale electrophysiology with Neuropixels probes," *Current opinion in neurobiology*, vol. 50, pp. 92–100, 2018.

[25] H. Benisty, A. Song, G. Mishne, and A. S. Charles, "Review of data processing of functional optical microscopy for neuroscience," *Neurophotonics*, vol. 9, no. 4, p. 41402, 2022.

[26] C. Stringer and M. Pachitariu, "Analysis methods for large-scale neuronal recordings," *Science*, vol. 386, no. 6722, p. eadp7429, 2024, doi: 10.1126/science.adp7429.

[27] G. Juve *et al.*, "Scientific workflow applications on Amazon EC2," in *2009 5th IEEE international conference on e-science workshops*, 2009, pp. 59–66.

[28] J. Moore *et al.*, "OME-NGFF: a next-generation file format for expanding bioimaging data-access strategies," *Nature methods*, vol. 18, no. 12, pp. 1496–1498, 2021.

[29] A. P. Buccino, O. Winter, D. Bryant, D. Feng, K. Svoboda, and J. H. Siegle, "Compression strategies for large-scale electrophysiology data," *Journal of Neural Engineering*, vol. 20, no. 5, p. 56009, 2023.

[30] D. M. T. Nguyen, J. C. Cortes, M. M. Dunn, and A. N. Shiklomanov, "Impact of Chunk Size on Read Performance of Zarr Data in Cloud-based Object Stores," *Authorea Preprints*, 2023.

[31] M. Ivanković, G. Petrović, R. Just, and G. Fraser, "Code coverage at Google," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019, pp. 955–963.

[32] W. Felidré, L. Furtado, D. A. Da Costa, B. Cartaxo, and G. Pinto, "Continuous integration theater," in *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2019, pp. 1–10.

[33] R. Ly, M. Avaylon, M. Wulf, A. Kepecs, and O. Rübel, "Structured behavioral data format: An NWB extension standard for task-based behavioral neuroscience experiments," *bioRxiv*, pp. 2024–2021, 2024, doi: 10.1101/2024.01.08.574597.

[34] D. Procida, "The Diátaxis Documentation Framework." [Online]. Available: https://diataxis.fr/

[35] D. Yuan, L. Cui, X. Liu, E. Fu, and Y. Yang, "A cost-effective strategy for storing scientific datasets with multiple service providers in the cloud," *arXiv preprint arXiv:1601.07028*, 2016.

Open Access | https://doi.org/10.25080/uhje9464

SciPy 2025

July 7 - July 13, 2025

Proceedings of the 24th Python in Science Conference
ISSN: 2575-9752

Performing Object Detection on Drone Orthomosaics with Meta's Segment Anything Model (SAM)

Nicholas McCarty[1]

[1]Upskilled Consulting

Abstract

Accurate and efficient object detection and spatial localization in remote sensing imagery is a persistent challenge. In the context of precision agriculture, the extensive data annotation required by conventional deep learning models poses additional challenges. This paper presents a fully open source workflow leveraging Meta AI's Segment Anything Model (SAM) for zero-shot segmentation, enabling scalable object detection and spatial localization in high-resolution drone orthomosaics without the need for annotated image datasets. Model training and/or fine-tuning is rendered unnecessary in our precision agriculture-focused use case. The presented end-to-end workflow takes high-resolution images and quality control (QC) check points as inputs, automatically generates masks corresponding to the objects of interest (empty plant pots, in our given context), and outputs their spatial locations in real-world coordinates. Detection accuracy (required in the given context to be within 3 cm) is then quantitatively evaluated using the ground truth QC check points and benchmarked against object detection output generated using commercially available software. Results demonstrate that the open source workflow achieves superior spatial accuracy — producing output `20% more spatially accurate`, with `400% greater IoU` — while providing a scalable way to perform spatial localization on high-resolution aerial imagery (with ground sampling distance, or GSD, < 30 cm).

Keywords object detection, spatial localization, drone orthomosaic

Published Jul 10, 2025

Correspondence to
Nicholas McCarty
nick@upskilled.consulting

Open Access

Copyright © 2025 McCarty. This is an open-access article distributed under the terms of the Creative Commons Attribution 4.0 International license, which enables reusers to distribute, remix, adapt, and build upon the material in any medium or format, so long as attribution is given to the creator.

Acknowledgments

We gratefully acknowledge the contributions of the open source community — thank you to the giants on whose shoulders we stand.

This work was funded by FiOR Innovations and Woodburn Nursery & Azaleas. We deeply appreciate their support and partnership.

A special thanks to Paniz Herrera, MBA, MSIST, Ryan Marinelli, PhD Fellow at the University of Oslo, and Danny Clifford for their help proofreading.

2. Introduction

Image segmentation is a critical task in geospatial analysis, enabling the identification and extraction of relevant features from high resolution remote sensing imagery [1]. However, extracting actionable information (i.e., object detection and spatial localization) can be constrained by the need for large, labeled datasets to train deep learning models in order to then perform inference and (hopefully) produce the desired output. This bottleneck is particularly acute in agricultural domains, where variability in conditions and object types complicates manual annotation [2].

Recent advances in foundation models, such as Meta AI's Segment Anything Model (SAM), offer a promising path forward. SAM is designed for promptable "zero-shot" segmentation. "Prompt engineering", in this context, involves using points and bounding boxes to focus the model's efforts on more efficiently generating masks corresponding to objects of interest [3]. Providing these prompts allows accurate masks to be generated for novel objects (ones not included in SAM's training corpus), without domain-specific training. Masks can also be generated automatically with no such prompting. SAM's automatic mask generator will effectively "detect" everything using open source model checkpoints and generate masks for each object in a provided image [4].

While SAM's ability to generalize is impressive [2], [4], its performance on remote sensing imagery and fine-grained features requires careful workflow integration and evaluation [1]. This paper describes a comprehensive, open source workflow for object detection and spatial localization in high-resolution remote sensing imagery, built around SAM and widely used geospatial Python libraries [5], [6], [7], [8]. The complete process is delineated, from data loading and preprocessing to mask generation, post-processing, and quantitative accuracy assessment, culminating in a rigorous comparison against the results produced using the proprietary software (see code). Precision, accuracy, F1 score, mean deviation (in cm), and Intersection-over-Union (IoU) are calculated in order to quantify the relative quality of the output produced using each workflow[1].

3. Motivation

Precision agriculture relies on accurate object detection for tasks such as plant counting, health monitoring, and the automated control of heavy equipment. Traditional deep learning approaches can become hindered by the cost and effort of generating carefully annotated data, limiting scalability and accessibility. Proprietary solutions, while effective, can be expensive and opaque, impeding reproducibility and customization.

[1]Output evaluation details are discussed in the Appendix.

Figure 1. *The derived centroids of the objects detected in drone orthomosaics are used to automate this nursery trimmer.*

SAM's zero-shot segmentation capability directly addresses the data annotation bottleneck, enabling rapid deployment in novel contexts. By developing an open source workflow around SAM, an end-to-end pipeline is created which allows for the quantitative evaluation of spatial accuracy with respect to objects detected in high-resolution aerial imagery. This modular workflow can also be repurposed as an automated data annotation pipeline for downstream model training/fine-tuning, if required[2].

4. Approach

Our approach integrates SAM's segmentation strengths with traditional geospatial data processing techniques, which lends itself to our precision agriculture use case. The workflow, like any other, can be thought of as a sequence of steps (visualized above and described below), each with their own sets of substeps:

- **Data Ingestion**: Loading GeoTIFF orthomosaics and QC point CSVs, extracting spatial bounds and coordinate reference systems (CRS) using Rasterio or GDAL.
- **Preprocessing**: Filtering QC points to those within image bounds, standardizing coordinate columns, and saving filtered data for downstream analysis.
- **Mask Generation**: Tiling large images for efficient processing, running SAM's automatic mask generator (***ViT-H*** variant) on each tile, and filtering masks by confidence.
- **Post-Processing**: Converting masks to polygons, filtering by area and compactness, merging overlapping geometry, and extracting centroids.
- **Accuracy Evaluation**: Calculating point-to-centroid deviations (in centimeters) between detected objects and QC points, compiling results, and generating visual and tabular reports.
- **Benchmarking**: Quantitatively comparing SAM-based results against the evaluated output using identical evaluation metrics (precision, recall, IoU, etc.; see Appendix for details).

It should be noted that there are no model training or fine-tuning steps included in our workflow, as we are using a foundation model to generate masks. This is analogous to using ChatGPT to generate text, which does not require users to train or fine-tune the underlying foundation model in order to do so.

This approach is carried out entirely using open source Python libraries, ensuring transparency and extensibility.

5. Methodology

5.1. Data and Environment

- **Imagery**: High-resolution GeoTIFF orthomosaic.
  - **Image size**: 18,200 x 55,708; 1,013,885,600 pixels
  - **Total area**: 545,243 sq ft; 12.5 acres
  - **GSD**: 0.71 cm
- **Ground Truth**: QC points in CSV format, containing spatial coordinates and unique identifiers.

[2]See this Colab notebook for details.

- **Coordinate Reference System (CRS) Transformations**: All spatial operations are performed using the NAD83 CRS (EPSG:6859), with reprojection to the WGS84 CRS (EPSG:4326) for downstream reporting and nursery trimmer automation.
- **Dependencies**[3]:
  - GDAL [5]
  - GeoPandas [7]
  - Matplotlib [9]
  - NumPy [10]
  - OpenCV [11]
  - OpenPyXL [12]
  - Pandas [13], [14]
  - Pillow [15]
  - Rasterio [8]
  - Segment Anything[4] [4]
  - Shapely [6]
  - Torch [16]

5.2. Workflow

1. Data Ingestion
2. Mask Generation
3. Post Processing
4. Accuracy Evaluation
5. Benchmarking

5.2.1. *Data Ingestion and Preprocessing*

$$
\begin{aligned}
&\boldsymbol{a}. && \text{Load GeoTIFF file (s)} \\
&&& \text{Extract image bounds : } \left(\min_x, \min_y, \max_x, \max_y\right) \\
&&& \text{Extract coordinate reference system (CRS)} \\
&\boldsymbol{b}. && \text{Load QC point CSV} \\
&&& \text{Clean and standardize column names} \\
&&& \text{Assign unique IDs to each QC point} \\
&&& \text{Reproject QC points to match image CRS (if needed)} \\
&\boldsymbol{c}. && \text{For each QC point :} \\
&&& \boldsymbol{if} \text{ point within image bounds } \boldsymbol{then} \\
&&& \text{Keep point} \\
&&& \boldsymbol{else} \\
&&& \text{Discard point} \\
&\boldsymbol{d}. && \text{Save filtered QC points to output CSV}
\end{aligned}
\tag{1}
$$

[3] See requirements.txt for version details.

[4] Inference was accelerated using `CUDA 12` (`cuDF 25.2.1`) on a `T4` GPU within our Colab notebook environment.

5.2.2. *Mask Generation*

$$\begin{aligned} &\textbf{a}. && \text{Initialize SAM (ViT -H checkpoint)} \\ & && \text{Use GPU if available} \\ &\textbf{b}. && \text{Tile ("chip") input images} \\ & && \text{Iteratively generate masks using SAM} \\ &\textbf{c}. && \text{Filter masks with confidence } \geq 80\% \\ &\textbf{d}. && \text{Convert masks to polygons} \\ & && \text{Aggregate results and export as GeoJSON} \end{aligned} \tag{2}$$

5.2.3. *Post-Processing*

$$\begin{aligned} &\textbf{a}. && \text{Filter polygons (e. g., based on area threshold)} \\ &\textbf{b}. && \text{Merge overlapping polygons} \\ & && \text{Extract individual (non -overlapping) polygons} \\ &\textbf{c}. && \text{Extract centroids from polygons} \end{aligned} \tag{3}$$

5.2.4. *Accuracy Evaluation*

$$\begin{aligned} &\textbf{a}. && \text{Load merged geometry and filtered QC points} \\ &\textbf{b}. && \text{Iteratively compute geometric deviation from QC points} \\ & && \text{Output deviations in centimeters} \end{aligned} \tag{4}$$

See Appendix for accuracy evaluation methodology details.

5.2.5. *Benchmarking*

$$\begin{aligned} &\textbf{a}. && \text{Apply accuracy evaluation steps to proprietary software output} \\ &\textbf{b}. && \text{Compare workflows using IoU, precision, recall, and F1 scores} \end{aligned} \tag{5}$$

See code for benchmarking methodology details.

6. Results

6.1. Proprietary Workflow

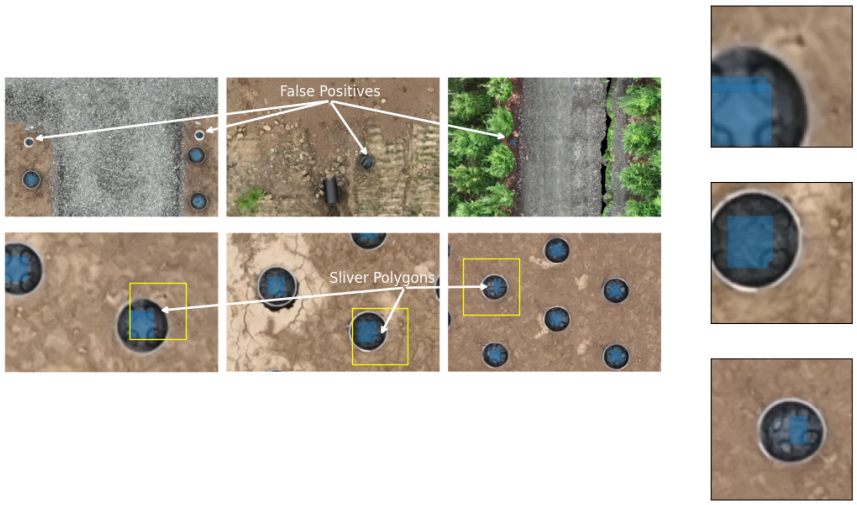


Figure 2. *19 false positives (FP) and hundreds of sliver polygons were observed in the output produced using the proprietary software.*

The bounding boxes that were output using this workflow (against which we are benchmarking ours) can be viewed as a layer overlain onto the GeoTIFF orthomosaic using GIS software[5]. What was of particular use to us is the fact that zero false negatives (FN) were observed in the output, though 19 false positives (FP) were. This empirical knowledge equips us with something not usually possessed in use cases such as this: the number of true positives (TP), which allows us to leverage such metrics as precision, recall, and the harmonic mean of the two, F1 score, to perform a rigorous comparison against our open source workflow (see code).

6.2. Open Source Workflow

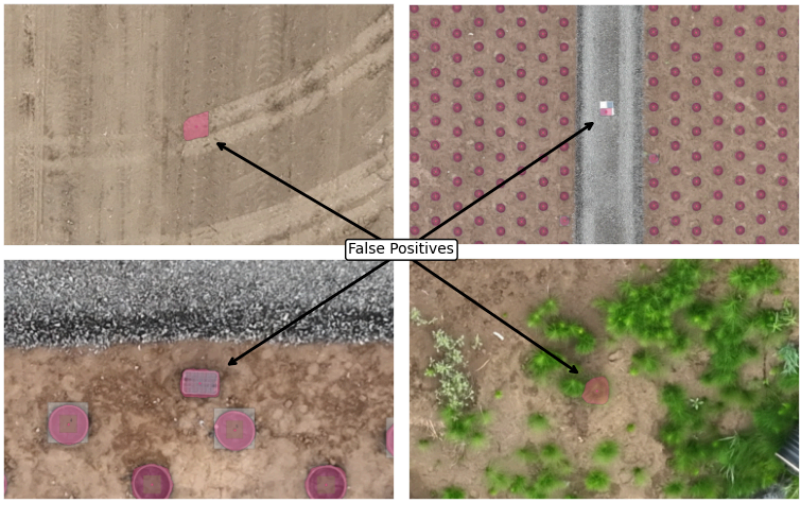


Figure 3. *18 FP were observed in the output produced using the open source workflow.*

[5]We used open source QGIS [17] as our selected data viewer.

Merging the overlapping geometry and filtering out the empirically observed FP allowed for us to ascertain exactly how many TP (18,736) there are in the benchmark output and derive how many FP (18) and FN (65) there are in our workflow output, which enabled us to conduct our performance comparison.

Table 1. *Performance Comparison*

Workflow	Detection Quality Metrics			Localization Accuracy Metrics	
	Precision	Recall	F1 Score	Mean Deviation (cm)	IoU
Proprietary	0.9990	1.0000	0.9995	1.39	0.18
Open Source	0.9990	0.9956	0.9973	1.20	0.74

It can be observed that empty plant pots tend to be ∼64 pixels (px) wide and tall; with QC points corresponding to actual pot centroids, we were able to create 64-by-64px boxes to facilitate our IoU calculations (see code). These calculations further allow us to assess the relative alignment between the detection output geometry and our "ground truth" geometry.

This work makes it easy to identify down to the individual QC point ID level which detection centroids deviate from said point by more than 3 cm, which is the tolerance specified by our client. In aggregate, we were able to gain a quantified sense of the mean deviation (in cm) of the output produced by each workflow. However, visual inspection revealed that some detection geometry was flagged as having out-of-tolerance centroids when the QC points were themselves off-center. This is to say that some detections from both workflows were flagged as being out-of-tolerance when they observably were not.

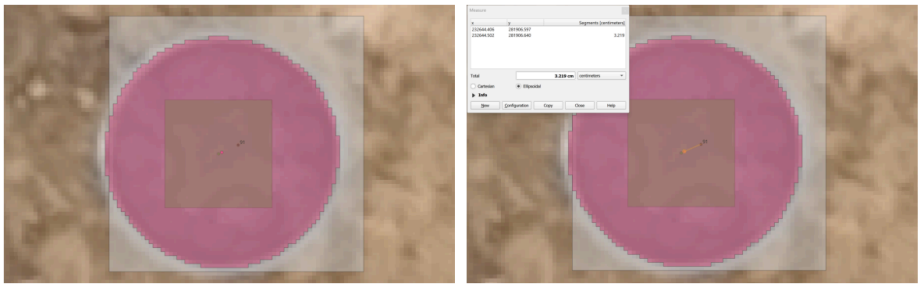


Figure 4. *Visual inspection of the detected centroids relative to QC point 91 reveal that the QC point is off-center.*

Visual inspection also reveals that our detections (the pink circle) and those produced using the commercial software (the beige square) have greater overall coverage with respect to the QC geometry (the grey square). This provides intuition as to why the IoU calculations revealed a 400% increase in coverage with respect to the geometry produced using SAM's automatic mask generator, zero-shot.

7. Discussion

7.1. Key Findings

The open-source workflow powered by Meta AI's Segment Anything Model (SAM) outperformed a commercial alternative in object detection and spatial localization on high-resolution drone imagery. It achieved `20% higher spatial accuracy` (1.20 cm vs. 1.39 cm deviation) and a `400% higher Intersection-over-Union (IoU)` (0.74 vs. 0.18), indicating stronger alignment between the detections and the actual object boundaries. Both methods had near-perfect precision, but the open-source approach showed slightly lower recall due to 65 false nega-

tives. It should be noted, however, that these FN were a direct result of the filtering substep in our workflow, which excludes detections outside of the provided geometry area and compactness thresholds; see code.

Nevertheless, its overall performance supports its suitability for precision agriculture and downstream automation.

7.2. Precision Agriculture Challenges

Our work began with an eye toward tackling a major challenge in agricultural remote sensing: the need for extensive manual annotation. SAM's zero-shot segmentation enables accurate object detection without domain-specific training, making it scalable and adaptable for new use cases with minimal setup.

7.3. Benefits of Open Source

Built entirely on open source geospatial tools, our open source workflow offers transparency, reproducibility, and flexibility. It can be tailored to suit various tasks, even automating annotation for model training, thereby supporting broader adoption with respect to high-resolution remote sensing imagery, in general.

7.4. Practical Impact

Meeting professional-grade tolerance requirements (e.g., < 3 cm) enables real-world applications, such as automating heavy equipment, based on precise object localization. This demonstrates how automated workflows can reduce manual labor and support more efficient agricultural practices.

7.5. Limitations and Future Work

Our approach to tiling ("chipping") high-resolution orthomosaics, processing 588 individual 1280-by-1280px tiles at an average pace of 11 seconds per tile, required a total processing time of ∼110 minutes running on a Colab single T4 GPU instance. It is important to note that an overlap of 25% (320px) between tiles during processing was required to ensure that geometry was not produced containing "holes" or malformations; merging overlapping polygons after filtering (based on area and compactness calculations, in this case) helped us ensure the overall quality of the geometric output.

Future work will be centered on building an open source CLI and Python package[6], which will allow users to pass orthomosaics as inputs and get geometry meeting desired spatial charactersitics as an output.

8. Conclusion

We present a robust, open source workflow for object detection and spatial localization in high-resolution drone orthomosaics, leveraging SAM's zero-shot segmentation capabilities. Our quantitative evaluation demonstrates improved accuracy over a commercially available software solution, underscoring the potential of foundation models and open source tools to advance scalable, cost-effective feature extraction. This work provides a template for further research and deployment in diverse contexts.

To our knowledge, this is the first comparative evaluation of an open source segmentation model against commercial software in a context requiring high (< 3 cm) spatial accuracy.

[6]We have since open-sourced the `orthomasker` Python package and CLI; work on a GUI is currently underway.

Our results demonstrate that the workflow not only matches but in some cases exceeds performance metrics with respect to the evaluated output.

9. Conflicts of Interest

The author declares no conflicts of interest.

10. AI Usage Disclosure

AI tools (ChatGPT, Perplexity, and NotebookLM) were used:

- in writing portions of the workflow integration code,
- to generate Matplotlib subplots, process flow diagrams, pseudocode, L A T E X, etc.
- for proofreading and light revision to reduce potential publication errors.

11. Code

Data and code required to replicate our approach can be found using the links below:

12. Appendix

12.1. Accuracy Evaluation Methodology

The evaluation focuses on two primary categories of metrics: ***localization accuracy*** and ***detection quality***; the employed methodology relies on the following data:

- **Ground Truth Quality Control (QC) Points** (P_{QC}), defined as a set of N_{QC} known spatial coordinates, $P_{QC} = \{p_j\}_{j=1}^{N_{QC}}$, where each $p_j = (x_j, y_j)$ represents the centroid (in real-world coordinates) of our object of interest (empty plant pots), serving as the ground truth for spatial localization.
- **Detected Object Centroids** (C_{Det}) are used, which are a set of N_{Det} centroids, $C_{Det} = \{c_k\}_{k=1}^{N_{Det}}$, where each $c_k = (x'_k, y'_k)$ is the centroid extracted from a polygon representing an object detected by the workflow.
- **Detected Object Polygons** (G_{Det}) are included, representing a set of N_{Det} polygons, $G_{Det} = \{g_k\}_{k=1}^{N_{Det}}$, where each g_k is a polygon generated from a SAM-produced mask after post-processing.
- **Ground Truth Polygons** (G_{GT}) the calculation of Intersection over Union (IoU) implies the existence of a corresponding set of N_{QC} ground truth polygons, $G_{GT} =$

$\{g'_j\}_{j=1}^{N_{QC}}$, where each g'_j delineates the extent of the object associated with ground truth point p_j. This allows for the quantification of the spatial alignment between the detection bounding boxes and those associated with the QC points, in aggregate. The provided code details how we created this geometry and performed the calculations.

References

[1] Q. Wu and L. P. Osco, "samgeo: A Python package for segmenting geospatial data with the Segment Anything Model (SAM)," *Journal of Open Source Software*, vol. 8, no. 89, p. 5663, 2023, doi: https://doi.org/10.21105/joss.05663.

[2] L. P. Osco *et al.*, "The Segment Anything Model (SAM) for Remote Sensing Applications: From Zero to One Shot." [Online]. Available: https://arxiv.org/abs/2306.16623

[3] A. Mayladan, H. Nasrallah, H. Moughnieh, M. Shukor, and A. J. Ghandour, "Zero-Shot Refinement of Buildings' Segmentation Models using SAM." [Online]. Available: https://arxiv.org/abs/2310.01845

[4] A. Kirillov *et al.*, "Segment Anything." [Online]. Available: https://arxiv.org/abs/2304.02643

[5] GDAL/OGR contributors, "GDAL/OGR Geospatial Data Abstraction software Library," 2025. doi: 10.5281/zenodo.5884351.

[6] S. Gillies *et al.*, "Shapely (Version 2.1.1)." 2025. doi: 10.5281/zenodo.5597138.

[7] K. Jordahl *et al.*, "geopandas/geopandas: v0.8.1." [Online]. Available: https://doi.org/10.5281/zenodo.3946761

[8] S. Gillies and others, "Rasterio: geospatial raster I/O for Python programmers." [Online]. Available: https://github.com/rasterio/rasterio

[9] J. D. Hunter, "Matplotlib: A 2D graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007, doi: https://doi.org/10.1109/MCSE.2007.55.

[10] C. R. Harris *et al.*, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, 2020, doi: https://doi.org/10.1038/s41586-020-2649-2.

[11] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[12] E. Gazoni and C. Clark, "OpenPyXL: A Python library to read/write Excel 2010 xlsx/xlsm/xltx/xltm files." [Online]. Available: https://openpyxl.readthedocs.io/

[13] The Pandas Development Team, "pandas-dev/pandas: Pandas." [Online]. Available: https://doi.org/10.5281/zenodo.3509134

[14] W. McKinney, "Data Structures for Statistical Computing in Python," in *Proceedings of the 9th Python in Science Conference*, Stéfan van der Walt and Jarrod Millman, Eds., 2010, pp. 56–61. doi: https://doi.org/10.25080/Majora-92bf1922-00a.

[15] A. Clark, "Pillow (PIL Fork) Documentation." [Online]. Available: https://buildmedia.readthedocs.org/media/pdf/pillow/latest/pillow.pdf

[16] A. Paszke *et al.*, "PyTorch: An Imperative Style, HighPerformance Deep Learning Library," *CoRR*, 2019, [Online]. Available: https://arxiv.org/abs/1912.01703

[17] QGIS Development Team, "QGIS Geographic Information System," 2021. [Online]. Available: https://www.qgis.org/

Open Access | https://doi.org/10.25080/ngdf5738

SciPy 2025

July 7 - July 13, 2025

Proceedings of the 24th Python in Science Conference
ISSN: 2575-9752

Extension of the OpenMC depletion module for transport-independent depletion

Oleksandr Yardas[1], **Paul Romano**[2], **Madicken Muk**[3], and **Kathryn Huff**[1]

[1]Dept. of Nuclear, Plasma, and Radiological Engineering, University of Illinois Urbana-Champaign, [2]Argonne National Laboratory, [3]Dept. of Nuclear Science and Engineering, Oregon State University Yet

Abstract

We have added functionality for running depletion simulations independently of neutron transport in OpenMC, an open source Monte Carlo particle transport code with an internal depletion module. Transport-independent depletion uses pre-computed static multigroup cross sections and fluxes to calculate reaction rates for OpenMC's depletion matrix solver. This accelerates the depletion calculation, but removes the spatial coupling between depletion and neutron transport.

We used a simple PWR pincell to validate the method against the existing transport-coupled depletion method. Nuclide concentration errors roughly scale with depletion time step size and are inversely proportional to the amount of the nuclide present in a depletable material. The magnitude of concentration error depends on the nuclide of interest. Concentration errors for low-abundance nuclides at longer (30-day) time steps exhibit large negative initial concentration the becomes more positive with time due to overestimation of nuclide production stemming from the lack of spatial coupling to neutron transport. For ten 3-day time steps, fission product concentration errors are all under 3%. Actinide concentration errors range from 10-15% for Am and Cm, 5-7% for Pu and Np, and 2% and less for U. Surprisingly, the numbers are similar for 30-day time steps. These results demonstrate the potential of this new method with moderate accuracy and extraordinary time savings for low and medium fidelity simulations. Concentration error characterization on larger models remains an open area of interest.

Keywords depletion, coupling, OpenMC, Cross sections

Published Jul 10, 2025

Correspondence to
Oleksandr Yardas
oyardas2@illinois.edu

Open Access

Copyright © 2025 Yardas *et al.*. This is an open-access article distributed under the terms of the Creative Commons Attribution 4.0 International license, which enables reusers to distribute, remix, adapt, and build upon the material in any medium or format, so long as attribution is given to the creator.

1. Introduction

Accurate simulations of advanced reactors on exascale computing hardware requires state-of-the-art software tools in Monte Carlo transport and heat-coupled computational fluid dynamics [1], [2]. These methods are computationally expensive, but deliver high confidence in their results. If any component in this tool chain could be simplified or sped up, it would reduce the computational cost on and economic cost from these exascale machines. For Monte Carlo neutron transport, depletion simulations are one of the more expensive simulations. *Depletion* is the process by which the nuclide composition changes due to nuclear transmutation and decay reactions occurring in a material subject to irradiation. Depletion capabilities were recently added [3] to OpenMC [4], an open source, community developed Monte Carlo particle transport code. The depletion solver utilizes OpenMC's Python API to iteratively run a transport simulation, calculate transmutation reaction rates, solve the depletion matrix (Eq. 2), and update the material composition. This makes transport-coupled depletion calculations computationally expensive, especially for large reactor models.

In the present work, we describe a new method for running depletion calculations without the need to iteratively run transport calculations by using pre-calculated microscopic cross sections and fluxes for each depletion step. At most, only one transport calculation needs to be run to generate these cross sections and fluxes. Neutron transport and depletion are tightly coupled because the change in material composition can result in changes to the spatial distribution of reaction rates. If few materials are used in a transport-independent depletion calculation, this effectively removes spatial effects of the changing material composition due to the static fluxes and cross sections.

This limits the practical application of transport-independent depletion to domains where capturing spatial effects are a secondary concern. One such application is in global fuel-cycle analysis where we are more concerned with the overall change in material composition. Fuel cycle simulations typically model many different reactors simultaneously over long time periods. The capabilities described in this paper have been utilized in Cyclus [5] – an open source fuel cycle simulator – for coupling depletion to the fuel cycle simulaton [6]. Another area of application is in facilities where the coupling between neutron transport and depletion is effectively one-way. This is often the case for experimental facilities with a neutron source where the source rate is high enough to cause activation of materials but not high enough to produce meaningful burnup of materials. Many proposed fusion energy facilities would fall into this category. For such applications, it is unnecessary to run a fully coupled simulation between neutron transport and depletion; instead, a neutron transport simulation corresponding to the beginning of irradiation can determine reaction rates relative to the neutron source rate, and then the depletion calculation can be performed using the reaction rates scaled by any changes in the neutron source rate over time. For fusion systems, these calculations are often carried out with FISPACT-II [7]; See [8] for a recent example. The capabilities described in this paper have also been utilized for a recent study of shutdown dose rate in fusion systems [9].

This paper is organized as follows: In Methods, we describe the methods and algorithms used to calculate reaction rates using the new depletion capabilities. In Results, we describe our simulation and analyze the results. In Conclusion, we summarize our results, and discuss some gaps in the current implementation of the new feature.

2. Methods

Depletion is typically done in sub-regions that are small enough to have little spatial variation in the flux (and hence in the reaction rates). For example, in a model of a full reactor core, depletion regions may correspond to individual fuel pins, or even smaller sub-pin regions of radial rings and azimuthal segments. The general form of the equation governing depletion is

$$\begin{aligned}\frac{dN_i}{dt} = \sum \texttt{limits}_{j=1}^{n} \left[\int_0^\infty f_{j\to i}(E)\sigma_j(E)\phi(E,t)dE + \lambda_{j\to i} \right] N_j(t) \\ - \left[\int_0^\infty \sigma_i(E)\phi(E,t)dE + \sum \texttt{limits}_{j=1}^{n} \lambda_{i\to j} \right] N_i(t),\end{aligned} \tag{1}$$

where

$$
\begin{aligned}
N_i(t) &\equiv \text{density of nuclide i at time t} \\
\sigma_i(E) &\equiv \text{transmutation cross section for nuclide i at energy E} \\
\phi(E,t) &\equiv \text{neutron flux at energy E and time t} \\
f_{j\to i}(E) &\equiv \text{fraction of transmutation reactions in nuclide j that produce nuclide i} \\
\lambda_{j\to i} &\equiv \text{decay constant for decay modes in nuclide j that produce nuclide i} \\
n &\equiv \text{total number of nuclides.}
\end{aligned} \tag{2}
$$

The summations in Equation (1) are effectively over all nuclides except i, as $f_{i\to i}(E)$ and $\lambda_{i\to i}$ must be zero by definition. Equation (1) can be condensed into a matrix-vector equation:

$$\frac{d\boldsymbol{n}}{dt} = \boldsymbol{A}(\boldsymbol{n},t)\boldsymbol{n} \tag{3}$$

where

$$\boldsymbol{n} = \begin{pmatrix} N_1 \\ N_2 \\ \vdots \\ N_n \end{pmatrix}, \tag{4}$$

and $\boldsymbol{A}(\boldsymbol{n},t) \in \mathbb{R}^{n\times n}$ is the depletion matrix containing the decay and transmutation coefficients. The coefficients of $\boldsymbol{A}$ change with time and depend on the nuclide composition vector $\boldsymbol{n}$, making this a nonlinear system of equations. OpenMC utilized the CRAM method to solve this system of equations [3].

We had two design goals for transport-independent depletion functionality:

1. Enable use of OpenMC depletion capabilities using only the Python API
2. Enable depletion of materials directly given a multi-group flux for each material.

We created the `IndependentOperator` and `MicroXS` classes to achieve these design goals.

2.1. MicroXS

The `MicroXS` class contains cross section data indexed by nuclide, reaction name, and energy group. It also contains methods to import and export `MicroXS` objects from `.csv` files. In principle, a user could create multi-group microscopic cross sections with a transport solver code, and then use the `MicroXS` class to read in the cross section data for use in depletion. However, this is unnecessary as the `get_microxs_and_flux()` function runs an OpenMC k -eigenvalue calculation to create a `MicroXS` object and multi-group flux profile for each user-specified domain.

In the initial release of this feature in OpenMC v0.13.1, `MicroXS` sub-classed the `pandas.DataFrame`. class to store data and assumed a one-group structure. The v0.14.0 release removed the `pandas` dependency and refactored `MicroXS` class to store multi-group cross section data.

2.2. IndependentOperator

The `Operator` class contains data and methods for running transport-coupled depletion calculations and runtime processing of OpenMC statepoint files to extract the data needed to solve Equation (2). In this work, we refactored the `Operator` class into a `CoupledOperator` class maintaining the existing depletion capability. We implemented the transport-independent depletion machinery in a new class, `IndependentOperator` which uses an instance of the `MicroXS` class and corresponding fluxes for each depletion domain to calculate reaction rates for that domain using multi-group microscopic cross sections.

Calculating reactions rates using multi-group cross sections is mathematically equivalent to tallying the reaction rates directly. In transport-coupled depletion calculation, the transmutation reaction rates $r_{ij}(t)$ are tallied directly as:

$$r_{ij}(t) = \left[\int_0^\infty \sigma_{ij}(E)\phi(E,t)dE\right] N_i(t) \tag{5}$$

where i and j indicate the nuclide and reaction, respectively.

To perform depletion using multi-group cross sections, the multi-group cross sections and fluxes are multiplied to get a per-source neutron reaction rate:

$$r_{ij}(t) = \left[\sum_g \sigma_{ij,g}\phi_g(t)\right] N_i(t) \tag{6}$$

In practice, this introduces a discretization error, but in the limit of infinitely small energy groups, Equation (5) is equivalent to Equation (4). To get a time-dependent microscopic cross section or flux, we must perform transport calculations (and in the case of time-varying microscopic cross sections, incorporate thermal feedbacks). Herein lies the primary assumption of transport-independent depletion: *microscopic cross sections and reactor fluxes are static*.

Both the tallied and multi-group-calculated reaction rates have units of $\text{reactions}/\text{ src particle}$. A normalization factor f in src/s is applied to obtain the more conventional unit of $\text{reactions}/\text{s}$:

$$R_{ij} = r_{ij} f \tag{7}$$

This normalization factor is typically related to the power of the reactor and energy released from fission (`fission-q` normalization) or the external source strength (`source-rate` normalization). `IndependentOperator` may use either of these methods to normalize the reaction rates. While `CoupledOperator` may use several methods to calculate fission yields, including using the Monte Carlo simulation to directly tally fission yields, `IndependentOperator` relies on constant yield data included in the depletion chain file.

3. Results

To validate transport-independent depletion, we ran depletion simulations on a PWR with a single depletion zone for three cases:

1. Transport-coupled depletion,
2. Transport-independent depletion,
3. Transport-independent depletion with microscopic cross sections updated after each depletion step.

The first case serves as a reference solution we use to estimate the error in the solution of the new method. We used 25 inactive and 125 active batches, with 10^6 particles per batch. For the second case, we used three different group structures to test the effect on solution accuracy: one group, CASMO-8, and CASMO-40. The third case is much slower than the first case as the cross sections need to be reloaded after each depletion step. The third case is a sanity check that the cross sections are being computed correctly, as it should have a smaller error than the second case. Due to the cross section discretization that occurs before performing depletion, we do not expect the third case results to converge to the first case results in limit of infinite particles, but we would expect it to converge in the limit of infinite particles *and* energy groups. The third case was only run for one-group transport-independent depletion.

Table 1 and Table 2 contain the material parameters and compositions of our model, and Table 3 contains the geometric parameters. We used the ENDF/B-VII.1 nuclear data library available at openmc.org/data. We used the ENDF/B-VII.1 depletion chain in the PWR spectrum also available at openmc.org/data.

Table 1. *Material Parameters*

Item	Fuel	Cladding	Water
Density [g cm^{-3}]	10.4	6	1.0
Volume [cm^3]	0.1764π	–	–
S(α,β)	–	–	'c_H_in_H2O'

Table 2. *Material Compositions*

Fuel composition		Cladding composition		Water composition	
Nuclide	Composition [atom %]	Nuclide	Composition [atom %]	Nuclide	Composition [atom %]
^{15}O	0.000758	^{90}Zr	0.5145	^{1}H	1.999689
^{16}O	1.999242	^{91}Zr	0.1122	^{2}H	0.000311
^{234}U	0.000385	^{92}Zr	0.1715	^{15}O	0.999621
^{235}U	0.043020	^{94}Zr	0.1738	^{16}O	0.000379
^{236}U	0.000197	^{96}Zr	0.028		
^{238}U	0.956398				

Table 3. *Geometric Parameters*

Fuel Radius [cm]	Clad Radius [cm]	Water Bounding Box dimensions [cm $\times$ cm]
0.42	0.45	1.24×1.24

Depletion is a slow process whose effects on neutronics only start to apply on longer timescales (days and months). For validation purposes we ran each of the three cases with four different magnitudes of timestep size: 360 seconds, 4 hours, 3 days, and 30 days. All simulations ran for 10 depletion steps. We used the `PredictorIntegrator` time stepper, which is analogous to the Euler method. Predictor-corrector methods are not useful for transport independent depletion. The correction step will utilize the same reaction rates as the initial prediction due to the static fluxes and cross sections. This means any predictor-corrector method will produce the same numerical results as a predictor method but with higher floating-point errors due to the increased number of operations[1]. We used constant reaction rate Q values multiplied by the reaction rates to obtain the normalization factor f (`fission-q` normalization) for all cases with a linear power density of 174 W/cm.

The general trend in our results is that concentration errors are smaller for shorter depletion time steps, and larger for longer depletion time steps, however some nuclides exhibit more complicated behavior.

[1]Preliminary testing with predictor-corrector methods confirmed the higher floating point errors, however their magnitudes are marginal.

(a) (b)

Figure 1. *Relative actinide concentration error using 3-day time steps at 3, 12, 21, and 30 days of depletion for (a) constant cross sections; (b) updating cross sections.*

Figure 1a and Figure 1b show the relative error in predicted actinide concentration for constant cross sections and updating cross sections, respectively, using 3-day time steps. Figure 3a and Figure 3b show the same respective quantities for 30-day time steps. As expected, updating the cross sections at each depletion step results in low predicted nuclide concentration errors, on the order of a fraction of a percent. The error trend for constant cross sections depends both on the cross section and time step size. For example, constant cross sections using a 3-day time step underpredicts the ^{241}Pu concentration, but at longer time steps overpredicts the concentration. Figure 2 shows the overprediction is due to overcalculating the rate of (n, γ) reactions that occur on ^{240}Pu.

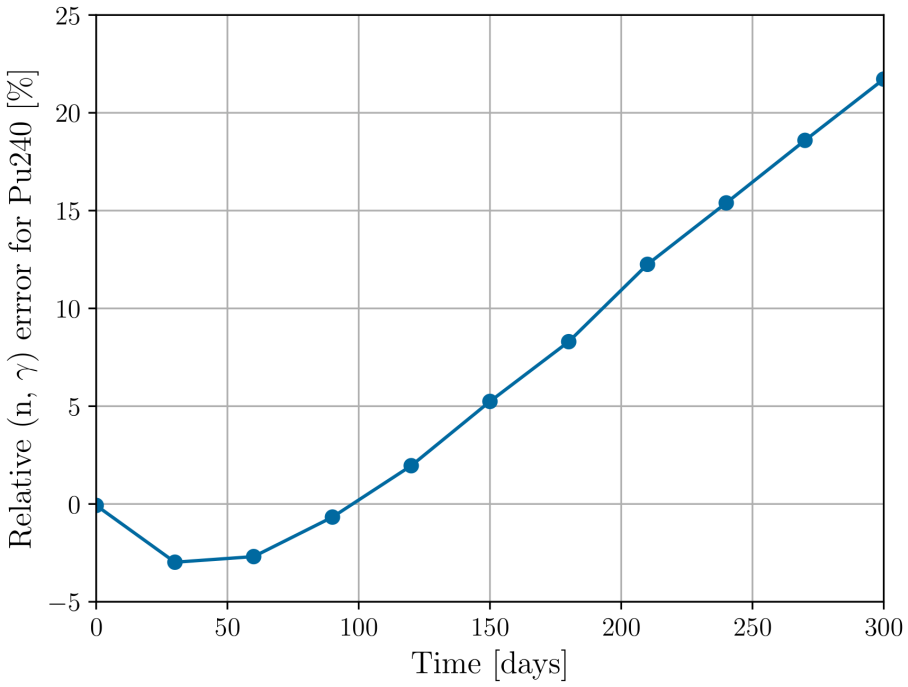


Figure 2. *Relative ^{240}Pu (n, γ) reaction rate error using constant cross sections and 30-day time steps.*

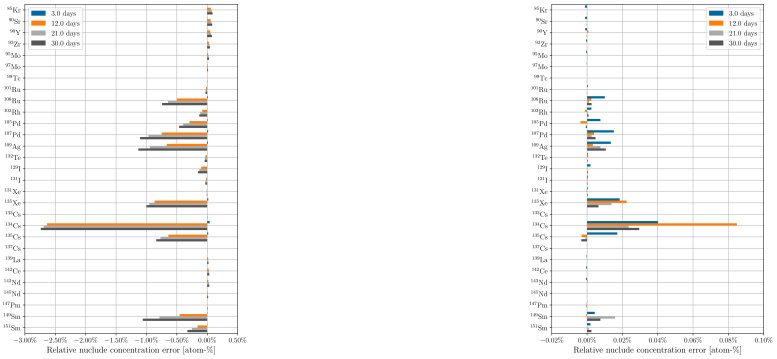
The error in ^{241}Pu concentration propagates to daughter nuclides that are related to the amount of ^{241}Pu, such as isotopes of Cm and Am. We observe a general trend that the less abundant nuclides have higher concentration error relative to transport-coupled depletion. The less abundant actinides have increased sensitivity to variations in production. The more abundant nuclides (U, ^{239}Np, ^{239}Pu, ^{240}Pu) tend to have low concentration errors, (5% or less), whereas the less abundant actinides (isotopes of Am, isotopes of Cm, ^{241}Pu, ^{242}Pu) tend to have high (more than 10%) concentration errors depending on the nuclide of interest.

(a) (b)

Figure 3. *Relative actinide concentration error using 30-day time steps at 30, 120, 210, and 300 days of depletion for (a) constant cross sections; (b) updating cross sections.*

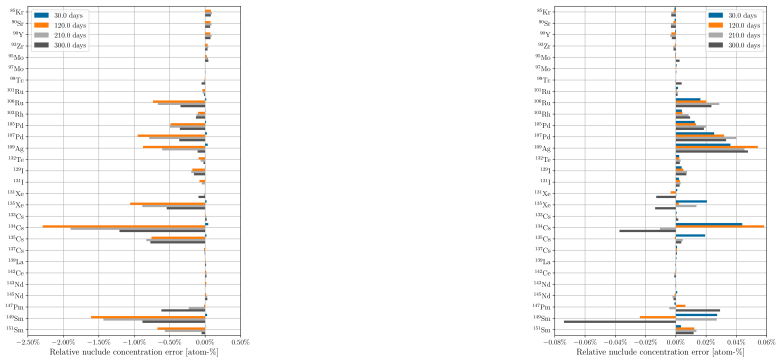
Figure 4a and Figure 4b show the relative error in predicted fission product concentration using constant cross sections and updating cross sections, respectively, using 3-day time steps. Figure 5a and Figure 5b show the same respective quantities for 30-day time steps. Similar to the actinides, updating the cross sections at each depletion step results in low predicted nuclide concentration errors. There is a lower concentration error across the board for many of these fission products compared to the actinides. This is due to the low concentration error of ^{235}U, which propagates to the fission products. Other actinides with higher concentration errors are orders of magnitude less abundant in the fuel than ^{235}U, so their effect on fission product concentration error is proportionally small.

Concentration errors for low-abundance nuclides using 30-day time steps follow an interesting trend where there is an initial large negative concentration error that becomes more positive over time. This behavior is due to the static cross sections and fluxes producing the same amount of these nuclides every timestep, whereas in the transport-coupled case, net production of the low-abundance nuclides decreases over time.



(a) (b)

Figure 4. *Relative fission product concentration error using 3-day time steps at 3, 12, 21, and 30 days of depletion for (a) constant cross sections; (b) updating cross sections.*



(a) (b)

Figure 5. *Relative fission product concentration error using 30-day time steps at 30, 120, 210, and 300 days of depletion for (a) constant cross sections; (b) updating cross sections.*

Repeating this analysis for both the CASMO-8 and CASMO-40 multi-group structures did not yield noticeable decreases in nuclide concentration errors for transport-independent depletion over the one-group case. Figure 6a and Figure 6b show the relative error in predicted actinide concentration using the CASMO-8 and CASMO-40 group structures, respectively, using 3-day time steps. Figure 7a and Figure 7b show the same respective quantities for 30-day time steps. It is possible in more complex models, like full reactor depletion, that the multi-group structure could become more important.

(a) (b)

Figure 6. *Relative actinide concentration error using constant cross sections and 3-day time steps at 3, 12, 21, and 30 days of depletion for the (a) CASMO-8 group structure; (b) CASMO-40 group structure.*

(a) (b)

Figure 7. *Relative actinide concentration error using constant cross sections and 3-day time steps at 30, 120, 210, and 300 days of depletion for the (a) CASMO-8 group structure; (b) CASMO-40 group structure.*

The transport-coupled simulations each took several hours to complete, whereas the transport-independent simulations using `IndependentOperator` took seconds to minutes to complete.

4. Conclusion

In this paper, we introduced transport-independent depletion in OpenMC. This method utilized pre-computed multi-group cross sections and fluxes to calculate reaction rates for a depletion calculation. The new method is much faster than running a transport-coupled depletion simulation, albeit with a penalty to accuracy. Better accuracy will be obtained for models where the neutron flux spectrum will be constant, i.e. in fusion systems and low

power fission reactors, so this new method should be used judiciously on problems where they are applicable.

This study focused exclusively on nuclide compositions. Future work may include building machinery to monitor criticality as the fuel depletes. A good starting point may be in implementing a method similar to the method to estimate k_∞ used in [10], wherein

$$k_\infty = \frac{\sum_i \sum_j N_i \nu_j \sigma_i^j}{\sum_i \sum_j N_i \sigma_i^j} \tag{8}$$

where j indicates the type of reaction, and ν_j is the neutron multiplication of reaction j, defined as

$$\nu_j = \tag{9}$$

In OpenMC, (n, xn) reactions are not used to calculate k_{eff} (and by extension k_∞). In this case, $\nu_x = 0$ in the numerator for all x, and we would use an 'effective' absorption cross section $\overline{\sigma}_a = \sigma_a - \sigma_{(n,2n)} - 2\sigma_{(n,3n)} - 3\sigma_{(n,4n)}$.

Future work could focus on characterizing the effect of sub-regions on the concentration errors. The geometry used in this study is a simple PWR pincell. It is possible that the error due to static fluxes could be reduced by dividing the pincell into sub-regions to better capture radial and axial variations as is done in high-fidelity depletion of large reactor models. Another area for future work could be on studying the importance of energy group discretization for more complex reactor models. We did not find a difference in concentration errors between one-group, CASMO-8 group, and CASMO-40 group transport-independent depletion for the pincell model. It is possible that energy group discretization becomes more important for complex models with distinct depletion zones.

References

[1] P. K. Romano *et al.*, "A Code-Agnostic Driver Application for Coupled Neutronics and Thermal-Hydraulic Simulations," *Nucl. Sci. Eng.*, vol. 195, pp. 391–411, 2021, doi: 10.1080/00295639.2020.1830620.

[2] E. Merzari *et al.*, "Exascale Multiphysics Nuclear Reactor Simulations for Advanced Designs," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, Denver, CO, USA, 2023. doi: 10.1145/3581784.3627038.

[3] P. K. Romano, "Depletion capabilities in the OpenMC Monte Carlo particle transport code," *Annals of Nuclear Energy*, p. 15, 2021, doi: https://doi.org/10.1016/j.anucene.2020.107989.

[4] P. K. Romano, N. E. Horelik, B. R. Herman, A. G. Nelson, B. Forget, and K. Smith, "OpenMC: A state-of-the-art Monte Carlo code for research and development," *Annals of Nuclear Energy*, vol. 82, pp. 90–97, 2015, doi: 10.1016/j.anucene.2014.07.048.

[5] K. D. Huff *et al.*, "Fundamental concepts in the Cyclus nuclear fuel cycle simulation framework," *Advances in Engineering Software*, vol. 94, pp. 46–59, 2016, doi: 10.1016/j.advengsoft.2016.01.014.

[6] A. M. Bachmann, O. Yardas, and M. Munk, "An Open-Source Coupling for Depletion During Fuel Cycle Modeling," *Nuclear Science and Engineering*, vol. 0, no. 0, pp. 1–14, 2024, doi: 10.1080/00295639.2024.2393940.

[7] J.-C. Sublet, J. W. Eastwood, J. G. Morgan, M. R. Gilbert, M. Fleming, and W. Arter, "FISPACT-II: An Advanced Simulation System for Activation, Transmutation and Material Modelling," *Nucl. Data Sheets*, vol. 139, pp. 77–137, 2017, doi: 10.1016/j.nds.2017.01.002.

[8] T. Eade, B. Colling, J. Naish, L. W. Packer, and A. Valentine, "Shutdown dose rate benchmarking using modern particle transport codes," *Nucl. Fusion*, vol. 60, no. 5, p. 56024, 2020, doi: 10.1088/1741-4326/ab8181.

[9] E. E. Peterson, P. K. Romano, P. C. Shriwise, and P. A. Myers, "Development and validation of fully open-source R2S shutdown dose rate capabilities in OpenMC," *Nucl. Fusion*, vol. 64, p. 56011, 2024, doi: 10.1088/1741-4326/ad32dd.

[10] M. Lovecký, L. Piterka, J. Prehradný, and R. Škoda, "UWB1 – Fast nuclear fuel depletion code," *Annals of Nuclear Energy*, vol. 71, pp. 333–339, 2014, doi: https://doi.org/10.1016/j.anucene.2014.04.021.

Open Access | https://doi.org/10.25080/kvfm5762

SciPy 2025

July 7 - July 13, 2025

Proceedings of the 24th Python in Science Conference
ISSN: 2575-9752

Pipeline-level differentiable programming for the real world

Andrin Rehmann[1†], **Dion Häfner**[1†], **Alessandro Angioi**[1], **Yongquan Qu**[1,2,3], and **Andrei Paleyes**[1]

[1]Pasteur Labs, Brooklyn, NY, USA, [2]Columbia University, Manhattan, NY, USA, [3]NSF Center for Learning the Earth with Artificial Intelligence and Physics, Manhattan, NY, USA; [†]Contributed Equally

Abstract

Automatic differentiation is increasingly being adopted beyond machine learning to tackle optimization problems in various scientific and engineering contexts. These methods have catalyzed the development of differentiable simulators, solvers, 3D renderers, and other powerful components, under the umbrella of differentiable programming. However, building pipelines that propagate gradients effortlessly across components introduces unique challenges. Real-world pipelines often span diverse technologies, frameworks, computing environments, and human skill sets. We argue for the need to support automatic differentiation on a system level to enable further real-world utility of differentiable programming, and ultimately, tangible progress within inverse modeling and design. To that end, we present Tesseract, a software ecosystem that provides pipeline-level differentiable programming at scale, and demonstrate its utility on a non-trivial shape optimization case study.

Keywords automatic differentiation, differentiable programming, differentiable physics, remote procedure calls, distributed computing, computer-aided engineering, containerization, scientific computing, scientific machine learning, simulation, interoperability

1. Introduction

Countless problems in science and engineering can be framed as tuning tasks, where parameters of a complex process — such as a physical simulation or a lab experiment — are iteratively tuned to maximize an objective. For such tasks, differentiable programming (DP) has emerged as a powerful tool, due to its ability to automate and accelerate gradient-based operations, enabling "differentiable physics" by codifying the use of gradient information to optimize, correct, or control physical systems. At the core of DP is the technique known as automatic differentiation (autodiff, AD) to compute partial derivatives of computer programs without the need to spell out explicit forms of said derivatives. Autodiff techniques enjoy great success in the fields of artificial intelligence and machine learning (ML) thanks to proliferation of deep learning software frameworks such as Autograd, TensorFlow, JAX, and PyTorch [1]. However, applications of AD in differentiable physics are largely untested at industrial scale, and ML frameworks are rarely designed for, nor tested on, practical science and engineering *systems* (as opposed to single components).

We propose "Differentiable Physics Programming" (DPP), a system engineering approach to AD-driven, simulation-heavy pipelines, including but not limited to workflows within computational fluid dynamics (CFD) and computer-aided engineering (CAE). DPP composes heterogeneous, end-to-end differentiable pipelines from components such as physics simulations, differentiable meshers, solvers, and learned models. As such, DPP is a sub-field of Simulation Intelligence (SI) [2], where motifs from numerical simulation, scientific com-

Published Jul 10, 2025

Correspondence to
Andrin Rehmann
andrin.rehmann@simulation.science

Open Access

Copyright © 2025 Rehmann *et al.*. This is an open-access article distributed under the terms of the Creative Commons Attribution 4.0 International license, which enables reusers to distribute, remix, adapt, and build upon the material in any medium or format, so long as attribution is given to the creator.

puting, and machine learning are combined to unlock fundamentally new possibilities for design and discovery in science and engineering.

To demonstrate DPP in action we leverage Tesseract, a software ecosystem that enables pipeline-level AD at real-world scale [3]. Tesseract resolves the above challenges via universal, autodiff-native software interfaces, containerization, and dataflow-based orchestration, built to be highly modular and interoperable with physics simulation tools and engineering data types. This allows scientists and engineers of diverse backgrounds to build complex workflows centered around simulation and data-driven surrogate models and propagate gradients throughout entire workflows, thus unleashing the potential of AD on end-to-end applications.

We give an overview of Tesseract's software design and functionality, and demonstrate its utility on a non-trivial problem, showcasing the minimization of compliance of a parametric structure made of a linear elastic material through a physical solver. Our aim in developing Tesseract is to present the community with the tools necessary to realize the potential of DPP broadly, and to build the community that drives the development of said workflows.

2. The state of pipeline-level differentiable programming

Real-world pipelines often span diverse technologies, frameworks, computing environments (local vs. distributed clusters; CPU vs. GPU), and teams with varying expertise. Additionally, legacy systems and non-differentiable components often need to coexist with modern AD-enabled frameworks. This provides a unique set of challenges for building and maintaining differentiable pipelines fit for real-world applications.

To the best of our knowledge, modern scientific computing tools do not provide native support for service-level AD, limiting all autodiff processing to a single compute context or hand-implemented remote procedure calls, e.g. via an AD-native framework (such as JAX, TensorFlow, PyTorch, Julia) and a service layer that exposes gradient-specific endpoints. Crucially, there are no standardized gradient APIs on the level of REST/gRPC protocols, so all of these solutions for service-level AD require large efforts to define AD-relevant endpoints, input and output schemas, and glue code. Additionally, it is necessary to factor in orchestration of distributed gradient calculation jobs and workflows.

This severely constrains the ability for researchers to realize the potential of DPP, which is demonstrated in applications like NeuralGCM [4] and solver-in-the-loop approaches [5]. These early applications, while powerful, are typically confined to 2 independent components, and coupled in an ad hoc way[1]. Better tooling and standardized interfaces are needed to enable the community to build complex, multi-step pipelines that can be easily shared and reused.

We are not the first to recognise this gap, and there are several examples in academic literature for solutions to distributed AD. B. T. Baker, A. Khanal, V. D. Calhoun, B. Pearlmutter, and S. M. Plis [6] proposes a technique for distributed training of deep neural networks that leverages the outer-product structure of the gradient of a network layer. K. Rush, Z. Charles, and Z. Garrett [7] adopted AD to the context of federated learning of ML models. Y. Tang, Z. Ding, D. Jankov, B. Yuan, D. Bourgeois, and C. Jermaine [8] considered the problem of differentiating computations expressed in relational databases. Nevertheless, even experimental support for pipeline-level AD is nascent, and it is remarkable that recent surveys on the state of AD do not consider it being an important research direction (B. Van Merriënboer, O.

[1]The solver-in-the-loop code is pure python, see an example pipeline here; components cannot be reused without knowledge over the specific details of their interface and installing all dependencies locally. NeuralGCM instead ships two separate components: the dynamical core of their physical simulation Dinosaur and the learned parts PressureLevelModel, both written in JAX.

Breuleux, A. Bergeron, and P. Lamblin [9], A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind [1]).

3. Tesseracts enable differentiable physics programming at scale

3.1. What is a Tesseract

Tesseracts are components that allow scientists to expose experimental, research-grade software to the world. They are self-contained, self-documenting, and self-executing, via command line, HTTP, and Python interfaces. They are designed to be easy to create, easy to use, and easy to share, including in a production environment. Crucially, Tesseracts provide built-in support for propagating gradient information at the level of individual components, making it easy to build complex, diverse software pipelines that can be optimized end-to-end.

This functionality is implemented in Tesseract Core, the foundational library for defining and serving autodiff-capable components as containerized services. In its simplest form, every Tesseract is a functional compute unit that has a single entrypoint `apply`, which wraps a software functionality of the user's choice. Other API functions of a Tesseract relate to the operation implemented in `apply`, for example `input_schema` returns expected input structure and types, `jacobian` implements a derivative, and so forth. A Tesseract is usually – but not necessarily – built and distributed as a Docker image, which allows it to be run in any environment that supports Docker, including local machines, cloud platforms, and HPC clusters without explicit dependency management. Tesseracts can be run as standalone services, or composed into larger pipelines, which can be orchestrated using a variety of tools, such as Kubernetes, Airflow, or Prefect.

An overview of Tesseract creation process is depicted in Figure 1.

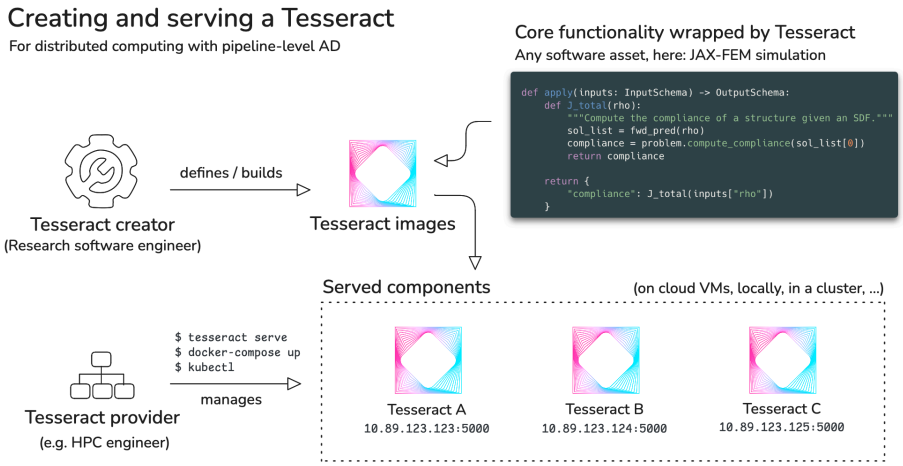


Figure 1. *The process of defining, creating, and serving a Tesseract.*

3.2. Scientific pipelines with Tesseract

Since all Tesseracts can be seen as standalone and stateless components that expose a handful of endpoints with fixed schemas, it is trivial to build pipelines connecting multiple Tesseracts, thus creating complex workflows. Multi-step computational workflows are very common across various branches of science. For example, a CAE pipeline might include steps for generating geometry, meshing, simulation, and postprocessing. Data processing

pipelines replace error-prone manual workflows with a structured, automated solution, improving reproducibility, quality, scalability, and collaboration.

To support native integration into JAX-based pipelines, we also provide Tesseract-JAX, which wraps Tesseract Core components as JAX-compatible primitives. This allows users to compose, differentiate, and jit-compile[2] pipelines involving Tesseracts using standard JAX transformations like `grad`, `jit` and `vmap`. Crucially, this allows users to auto-differentiate entire pipelines that include Tesseracts with a single call to `jax.grad` (Figure 2).

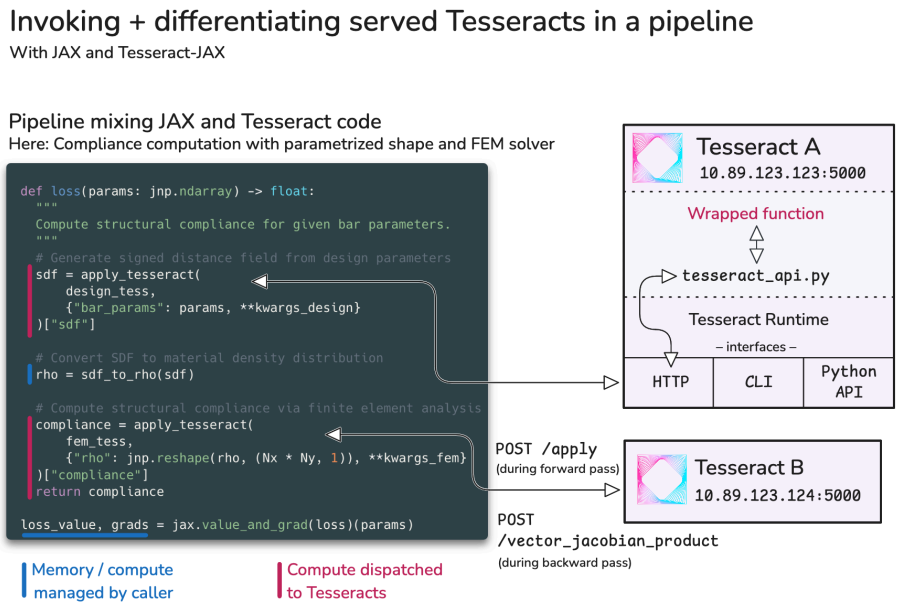


Figure 2. *The process of defining multi-tesseract pipelines with Tesseract-JAX.*

Several demonstrations of such pipelines already exist in Tesseract ecosystem:

- **4D-Variational data assimilation for a chaotic dynamical system.** Using Tesseract Core, we hand-implement a data assimilation pipeline for a chaotic dynamical system. It exploits auto-differentiation capabilities of JAX to backpropagate gradients through the data generating process for efficient solution of the 4D-Variational problem.
- **Gradient-based optimization of a differentiable CFD simulation.** Uses Tesseract-JAX to automatically register Tesseracts as JAX-compatible functions, and performs optimization over them through a computational fluid dynamics (CFD) simulation.

While illustrative, these examples are relatively simple and only contain a single Tesseract that is being differentiated in an optimization context. In practice, scientific pipelines are often more complex, involving multiple Tesseracts that need to be composed together and differentiated end-to-end. This is where Tesseract shines, as it allows users to build complex workflows that can be easily shared and reused.

3.3. Case study: Parametric shape optimization with differentiable FEM simulation

👉 See supplementary notebook for full case study 👈

[2]In this context, with "jittable" we imply that JAX does not throw errors when Tesseracts are wrapped with `jax.jit`, although no actual compilation of internal Tesseract code is happening. This is still useful, as it allows users to mix and match Tesseract computations with JAX code in the same functional block. Inner components of Tesseracts may be JIT compiled separately if using JAX internally.

We present a concrete demonstration for how the Tesseract ecosystem enables differentiable physics programing, in the form of a novel case study showcasing parametric end-to-end shape optimization of a geometric model with respect to its physical properties.

The pipeline consists of several steps:

1. **Assemble**: construct a parametric geometry using a standard 3D geometry library.
2. **Discretize**: compute an SDF (signed distance field) on a fixed grid.
3. **Transform**: apply a sigmoid function to transform the SDF into a density field.
4. **Solve**: compute the compliance of the design via a finite element solver.

In this pipeline, steps (1)+(2) and (4) are implemented as Tesseract and containerized, while step (3) is implemented as a standard JAX function (Figure 3).

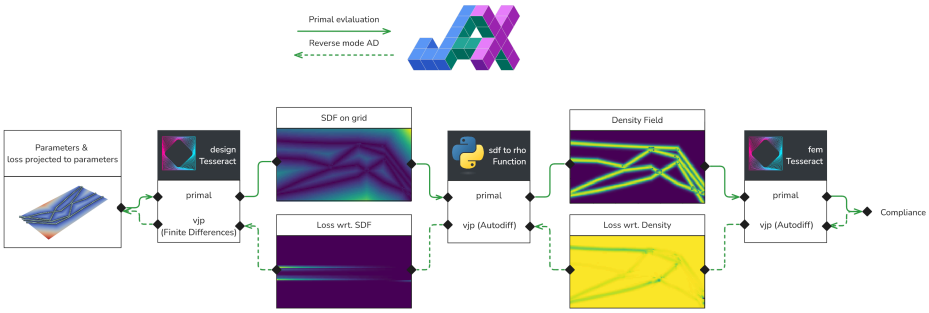
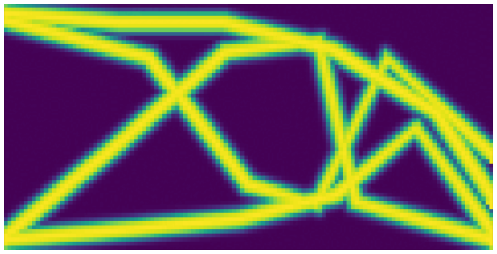


Figure 3. *Data flow through a Tesseract-based pipeline for parametric shape optimization. Involves two separate Tesseracts: one for computing a signed distance field (SDF) from a parametric geometry, and another for computing the compliance of a structure given a density field via finite element analysis. In the picture, the plots display the intermediate results and gradients which are passed between components, and the boxes with a Python or Tesseract logo represent the actual components that are implemented in Python or as a Tesseract in this demo. The primal evaluation, with solid green lines, is executed first, left-to-right, after which the backward pass, in dotted green lines, calculates the gradient of the the loss function with respect to the pipeline's input parameters.*

Since the mapping from design parameters to SDF is done through an industry-standard geometric processing library (PyVista), it is not natively auto-differentiable. Therefore, we implement a custom Tesseract endpoint computing a vector-Jacobian product using finite differences. We then leverage Tesseract-JAX to assemble the full processing pipeline, and use the standard gradient computation function `jax.grad` to compute the total derivative of the objective (compliance) with respect to the design parameters. This allows us to apply gradient-based optimization methods to the design parameters, such as gradient descent or Adam.

We compare our solution with the free form topology optimization solution that is implemented in the jax-fem library [10]. We observe that our solution yields a structure that is strikingly similar to the free form topology optimization solution, even though in our case the design space is parametrized by a small number of parameters, and not solving an unconstrained topology optimization problem.

Parametric Optimization (Ours)	**Free Form Topology Optimization (jax-fem example)**

Parametric Optimization (Ours)	**Free Form Topology Optimization (jax-fem example)**
	

Despite being a synthetic example, this has line-of-sight to many important real-world applications, such as shape optimization of mechanical structures parametrized by a CAD model, or optimization of fluid flow through a complex geometry. Tesseract pipelines allow us to swap out toy components with real-world implementations, while keeping the overall structure of the pipeline intact.

3.4. Unique value-add

In a world without Tesseract, the above would be significantly more difficult for several reasons.

Heterogeneity of gradient computation. In this pipeline some components rely on automatic differentiation, while others rely on finite differences. With Tesseracts, each component is free to implement its own gradient computation strategy, and the pipeline can be composed without knowledge about the details of each component.

Modularity. The components of the pipeline are implemented as Tesseract components, which allows us to easily swap out components and reuse them in other pipelines. For example, we could replace the design space Tesseract relying on PyVista with one relying on OpenSCAD, or other commercial CAD software. This is evidenced in the compact representation of the pipeline in Figure 2, which directly maps to the logical steps of the implemented process (*assemble-discretize-transform-solve* above).

Dependency management. Tesseract components are containerized, which allows us to easily manage dependencies and ensure that the pipeline runs in a consistent environment, which greatly simplifies working with heavyweight scientific software like numerical solvers or graphics processing libraries.

Computing resources. Tesseract components can be run on different computing resources, which allows us to easily scale the pipeline and run it on different machines. For example, we are able to run the finite element solver on a GPU machine and the design space Tesseract on a separate CPU node to make the most of available resources.

4. Related work

Tesseracts offer a combination of containerised runtime for scientific computing and native AD capabilities. Considered separately, both these areas are rich with existing tools and frameworks.

Containerised runtime. There are several projects that provide containerisedruntime infrastructure for scientific computations. Some, including MLServer [11], BentoML [12], or Triton Inference Server [13], target primarily machine learning workloads. Other projects, such as UM-Bridge [14] or Singularity [15], cover a broader scope of scientific domains. A key

feature that separates Tesseract ecosystem from these project is its strong focus on gradient computation, and batteries-included approach to the entire SI component lifecycle.

Automatic differentiation. Deep learning frameworks like PyTorch [16] and TensorFlow [17] implement AD and leverage it extensively for training of ML models. AD is also one of the main features of JAX, a Python library for high performance numerical computing [18], and JAX primitives strongly influenced the design of Tesseract's API. However, these frameworks support AD on a program level, which assumes that the entire program is locally available and traceable. While composite solutions are possible (e.g. combining an AD-capable backend with an HTTP service), there is no native support for distributed AD.

To the best of our knowledge, Tesseract is the first software project that combines these two areas as first-class citizens, and provides a comprehensive solution for building and deploying differentiable scientific computing pipelines.

5. Future work

The Tesseract ecosystem already supports a wide range of scientific and engineering workflows. However, not all envisioned use cases are sufficiently supported with tooling, validation, demonstration, and documentation. Ongoing work emphasizes several key directions to improve Tesseract's applicability and usability in real-world scenarios.

Distributed and cloud-native automatic differentiation. That is, gradient-based workflows that span multiple machines and heterogeneous environments, including HPC clusters and cloud platforms. This involves enabling remote gradient execution, efficient recomputation strategies, and distributed pipeline orchestration.

Cross-framework integration. Many scientific pipelines today mix components from different programming ecosystems. Tesseracts enable end-to-end gradient propagation across JAX, PyTorch, Julia, and other tools—treating them as composable, AD-aware Tesseract components.

Wrapping simulations behind unified, differentiable interfaces. Similar to frameworks like OpenAI Gymnasium, Tesseracts can serve as wrappers for expensive simulations—e.g., CFD solvers, structural mechanics codes—that expose unified APIs along with gradient endpoints. This makes it easy to use classical simulation software within differentiable optimization or reinforcement learning pipelines.

Interoperability in CAE and beyond. Many domain-specific tools in CAE are poorly integrated with AD tooling. Tesseracts are a natural fit to provide robust support for meshing, geometry processing, and legacy formats in differentiable workflows, allowing users to plug in existing CAE pipelines without compromising engineering efficiency.

Pipeline-level DPP for real-world applications. Ultimately, Tesseract aims to demonstrate the real-world viability of DPP across full scientific workflows. This includes hybrid pipelines that blend simulation and machine learning (e.g., surrogate modeling, control, inverse design), and stress-testing DPP tooling under realistic constraints.

6. Conclusions

The Tesseract ecosystem is born out of realization that modern autodiff tooling is limited in its scaling capabilities. Tesseracts demonstrate how this gap can be addressed by elevating the concept of derivative computation to the systems level.

In particular, the focus on composing modular components into end-to-end differentiable pipelines addresses a critical barrier in real-world scientific computing, where workflows

often span multiple communities, frameworks, and computational environments. By containerizing components and standardizing interfaces, Tesseracts bridge the gap between specialized tools, legacy systems, research-grade software, and modern AD engines.

7. Further reading

- Tesseract documentation.
- Tesseract Core repository on GitHub.
- Tesseract Community Forum; in particular, the showcase section.

References

[1] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind, "Automatic differentiation in machine learning: a survey," *Journal of machine learning research*, vol. 18, no. 153, pp. 1–43, 2018.

[2] A. Lavin *et al.*, "Simulation Intelligence: Towards a New Generation of Scientific Methods," *arXiv preprint arXiv:2112.03235*, 2021, doi: 10.48550/arXiv.2112.03235.

[3] D. Häfner and A. Lavin, "Tesseract Core: Universal, autodiff-native software components for Simulation Intelligence," *Journal of Open Source Software*, vol. 10, no. 111, p. 8385, 2025, doi: 10.21105/joss.08385.

[4] D. Kochkov *et al.*, "Neural general circulation models for weather and climate," *Nature*, vol. 632, no. 8027, pp. 1060–1066, 2024, doi: 10.1038/s41586-024-07744-y.

[5] K. Um, R. Brand, Y. R. Fei, P. Holl, and N. Thuerey, "Solver-in-the-loop: Learning from differentiable physics to interact with iterative PDE-solvers," *Advances in neural information processing systems*, vol. 33, pp. 6111–6122, 2020, doi: 10.48550/arXiv.2007.00016.

[6] B. T. Baker, A. Khanal, V. D. Calhoun, B. Pearlmutter, and S. M. Plis, "Peering beyond the gradient veil with distributed auto differentiation," *arXiv preprint arXiv:2102.09631*, 2021.

[7] K. Rush, Z. Charles, and Z. Garrett, "Federated automatic differentiation," *Journal of Machine Learning Research*, vol. 25, no. 357, pp. 1–39, 2024.

[8] Y. Tang, Z. Ding, D. Jankov, B. Yuan, D. Bourgeois, and C. Jermaine, "Auto-differentiation of relational computations for very large scale machine learning," in *International Conference on Machine Learning*, 2023, pp. 33581–33598.

[9] B. Van Merriënboer, O. Breuleux, A. Bergeron, and P. Lamblin, "Automatic differentiation in ML: Where we are and where we should be going," *Advances in neural information processing systems*, vol. 31, 2018.

[10] T. Xue *et al.*, "JAX-FEM: A differentiable GPU-accelerated 3D finite element solver for automatic inverse design and mechanistic data science," *Computer Physics Communications*, p. 108802, 2023, doi: 10.1016/j.cpc.2023.108802.

[11] Seldon, "MLServer: An open source inference server for your machine learning models." [Online]. Available: https://github.com/SeldonIO/MLServer/

[12] C. Yang *et al.*, "BentoML: The framework for building reliable, scalable and cost-efficient AI application." [Online]. Available: https://github.com/bentoml/bentoml

[13] NVIDIA Corporation, "Triton Inference Server: An Optimized Cloud and Edge Inferencing Solution." [Online]. Available: https://github.com/triton-inference-server/server

[14] L. Seelinger, V. Cheng-Seelinger, A. Davis, M. Parno, and A. Reinarz, "UM-Bridge: Uncertainty quantification and modeling bridge," *Journal of Open Source Software*, vol. 8, no. 83, p. 4748, 2023, doi: 10.21105/joss.04748.

[15] G. M. Kurtzer, V. Sochat, and M. W. Bauer, "Singularity: Scientific containers for mobility of compute," *PloS one*, vol. 12, no. 5, p. e177459, 2017, doi: 10.1371/journal.pone.0177459.

[16] A. Paszke *et al.*, "Automatic differentiation in PyTorch," *Autodiff Workshop, NIPS*, 2017.

[17] M. Abadi *et al.*, "TensorFlow: a system for Large-Scale machine learning," in *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, 2016, pp. 265–283.

[18] J. Bradbury *et al.*, "JAX: composable transformations of Python+NumPy programs." [Online]. Available: http://github.com/jax-ml/jax

Open Access | https://doi.org/10.25080/vwty6796

SciPy 2025

July 7 - July 13, 2025

Proceedings of the 24th Python in Science Conference
ISSN: 2575-9752

Phlower: A Deep Learning Framework Supporting PyTorch Tensors with Physical Dimensions

Riku Sakamoto[1]

[1]RICOS Co. Ltd.

Abstract

We present Phlower, an open-source deep learning library that extends PyTorch tensors to incorporate physical dimensions — such as time (T), mass (M), and length (L) — and enforces dimensional consistency throughout computations.

When applying deep learning to physical systems, tensors often represent quantities with physical dimensions. Ensuring dimensional correctness in such operations is crucial to maintaining physical validity. To address this challenge, Phlower provides `PhlowerTensor`, a wrapper class of PyTorch tensor that tracks physical dimensions and ensures that tensor operations comply with dimensional consistency rules.

This paper introduces the core features of Phlower and demonstrates its capability to maintain dimensional correctness through representative examples.

Keywords deep learning, physical dimension, numerical simulation

1. Introduction

The simulation of physical phenomena plays a critical role in various aspects of daily life, including vehicle design and weather forecasting. As a new paradigm, *physics-informed machine learning*, which integrates machine learning with physical simulations, has recently gained attention, particularly with the emergence of Physics-Informed Neural Networks (PINNs) [1] and Graph Neural Networks (GNNs). Compared to traditional simulation methods, it offers the potential to accelerate predictions of physical behavior and to uncover new physical laws that might otherwise remain hidden.

As shown in Table 1, Physical dimensions are the fundamental quantities used to describe physical systems. These include time (T), mass (M), length (L), electric current (I), thermodynamic temperature (Θ), amount of substance (N), and luminous intensity (J). In numerical simulations of physical phenomena, physical dimensions ensure the correctness of computations. Extending this concept to deep learning provides several advantages, we can enhance the reliability of model architectures by preventing invalid operations between physically incompatible quantities. This capability helps avoid subtle implementation errors and improves code readability and maintainability, especially in scientific and engineering applications where preserving physical meaning is critical.

PyTorch [2] is a deep learning framework that provides tensor operations and automatic differentiation capabilities. However, it does not natively support physical dimensions. To fill this gap, we develop Phlower, an open-source deep learning framework, and introduce `PhlowerTensor`, a wrapper class of PyTorch tensors and enables physical dimension tracking. This ensures that tensor operations adhere to dimensional consistency rules.

Published Jul 10, 2025

Correspondence to
Riku Sakamoto
sakamoto@ricos.co.jp

Open Access

Copyright © 2025 Sakamoto. This is an open-access article distributed under the terms of the Creative Commons Attribution 4.0 International license, which enables reusers to distribute, remix, adapt, and build upon the material in any medium or format, so long as attribution is given to the creator.

From an implementation perspective, `PhlowerTensor` holds an underlying PyTorch tensor, making conversion between PyTorch tensors and `PhlowerTensor` straightforward. Moreover, `PhlowerTensor` maintains compatibility with PyTorch's autograd system and core tensor operations such as `sum()`, enabling seamless integration into existing PyTorch-based workflows with minimal code modifications. Program 1 illustrates its usage.

Table 1. *Physical Dimensions [3]*

quantity	Symbol
time	T
mass	M
length	L
electric current	I
thermodynamic temperature	Θ
amount of substance	N
luminous intensity	J

```python
from phlower import phlower_tensor
import torch

# Example: Calculating kinetic energy (E = 0.5 * m * v^2)

# A mass (m) has dimensions M^1
mass = phlower_tensor(torch.rand(10, 1), dimension={"M": 1})

# A velocity (v) has dimensions L^1 T^-1
velocity = phlower_tensor(torch.rand(10, 3, 1), dimension={"L": 1, "T": -1})

kinetic_energy = 0.5 * mass * torch.sum(velocity ** 2, axis=-1)

print(kinetic_energy.dimension)
# Output: PhlowerDimensionTensor(T: -2.0, L: 2.0, M: 1.0, I: 0.0, Theta: 0.0, N: 0.0, J: 0.0)
```

Program 1. *PhlowerTensor carrying physical dimensions*

In this paper, we will introduce Phlower's key features and demonstrate how physical dimensions are handled in deep learning workflows. The remainder of this paper is organized as follows:

- Core Concepts introduces the core concept of Phlower and its key features. Defining `PhlowerTensor` and enforcing dimensional consistency are discussed, along with examples of tensor operations that maintain physical dimensions.
- Use Cases presents possible use cases of `PhlowerTensor` in deep learning applications, particularly in scientific machine learning and physics-informed models. It highlights how `PhlowerTensor` can prevent dimensional inconsistencies and enforce dimensional validity in feature engineering.
- Additional Features discusses additional features of Phlower, including YAML-based model definition and shape semantics. These features enhance the usability and flexibility of Phlower in deep learning workflows.
- Related Work compares Phlower with related work, highlighting its unique features and advantages over existing libraries for handling physical dimensions in scientific computing and machine learning.

2. Core Concepts

2.1. Design Philosophy

We design Phlower to extend PyTorch tensors with support for physical dimensions while maintaining full compatibility with existing PyTorch workflows.
Its design is guided by the following key principles:

- **Dimensional Consistency**: Phlower enforces dimensional consistency during tensor operations, ensuring that operations involving tensors with incompatible physical dimensions raise informative errors. This prevents subtle bugs and improves the reliability of deep learning models.
- **Seamless Integration**: Phlower is designed to integrate seamlessly with PyTorch, allowing users to leverage existing PyTorch functionalities while adding physical dimension tracking. This includes compatibility with PyTorch's autograd system and core tensor operations.
- **Lightweight Implementation**: `PhlowerTensor` is implemented as a lightweight wrapper around PyTorch tensors, minimizing the overhead of using physical dimensions. This allows users to easily convert between PyTorch tensors and `PhlowerTensor` without significant performance impact.

2.2. Defining PhlowerTensor

Program 2 shows how to create a `PhlowerTensor` from a PyTorch tensor. The `phlower_tensor` function takes a PyTorch tensor and a dictionary specifying the physical dimensions.

```python
from phlower import phlower_tensor
import torch

# Create a PhlowerTensor with physical dimensions at each point
# Example: A tensor representing velocity with dimensions L^1 T^-1
velocity = phlower_tensor(torch.rand(10, 3), dimension={"L": 1, "T": -1})
print(velocity)

# Example: A tensor representing pressure with dimensions M^1 T^-2 L^-1
pressure = phlower_tensor(torch.rand(10, 1), dimension={"M": 1, "T": -2, "L": -1})
print(pressure)
```

Program 2. *Create PhlowerTensor from PyTorch Tensor*

2.3. Dimensional Consistency

`PhlowerTensor` enforces dimensional consistency during tensor operations. Program 3 demonstrates how `PhlowerTensor` ensures dimensional correctness when computing kinetic energy, defined as $E = 0.5 \cdot m \cdot \boldsymbol{v}^2$, where m is mass and $\boldsymbol{v}$ is velocity.

As illustrated in Figure 1, this example assumes that each point is associated with a three-dimensional velocity vector and a scalar mass. The resulting kinetic energy has dimensions $M^1L^2T^{-2}$, corresponding to the physical dimension of energy.

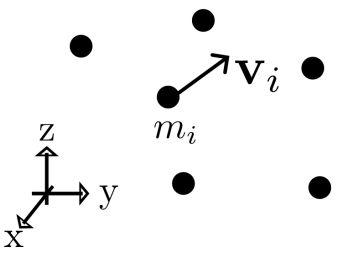


Figure 1. *Example of point clouds. Each point has mass and velocity.*

```python
from phlower import phlower_tensor
import torch

# Example: Calculating kinetic energy (E = 0.5 * m * v^2)

# Mass has dimensions M^1
mass = phlower_tensor(torch.rand(10, 1), dimension={"M": 1})

# Velocity has dimensions L^1 T^-1
velocity = phlower_tensor(torch.rand(10, 3, 1), dimension={"L": 1, "T": -1})

kinetic_energy = 0.5 * mass * torch.sum(velocity ** 2, axis=-1)

print(kinetic_energy.dimension)
# Output: PhlowerDimensionTensor(T: -2.0, L: 2.0, M: 1.0, I: 0.0, Theta: 0.0, N: 0.0, J: 0.0)
```

Program 3. *Calculating kinetic energy with PhlowerTensor*

On the other hand, Phlower disallows operations involving incompatible physical dimensions. Program 4 shows `PhlowerTensor` raise an error when attempting to add a pressure tensor to a velocity tensor.

```python
# Example: A tensor representing velocity with dimensions L^1 T^-1
velocity = phlower_tensor(torch.rand(10, 3),  dimension={"L": 1, "T": -1})

# Example: A tensor representing pressure with dimensions M^1 T^-2 L^-1
pressure = phlower_tensor(torch.rand(10, 1),  dimension={"M": 1, "T": -2, "L": -1})

# Attempting to add velocity and pressure tensors with incompatible dimensions
try:
    result = velocity + pressure
except ValueError as e:
    print(f"Error: {e}")
    # Output: Error: Add operation for different physical dimensions is not allowed.
```

Program 4. *Attempting to add tensors with incompatible dimensions*

PhlowerTensor supports in-place operations (e.g., +=, *=) while enforcing dimensional consistency. Program 5 illustrates an example of in-place operations for `PhlowerTensor`.

```python
# Example: In-place operation for non-dimensional tensor
# Objects other than PhlowerTensor (e.g., float) are treated as non-dimensional
non_dimensional_value = phlower_tensor(torch.rand(10, 3),  dimension={})
non_dimensional_value += 5.0

# Example: A tensor representing pressure with dimensions M^1 T^-2 L^-1
pressure = phlower_tensor(torch.rand(10, 1),  dimension={"M": 1, "T": -2, "L": -1})

# In-place subtraction with compatible dimensions
pressure -= phlower_tensor(torch.tensor(1.0),  dimension={"M": 1, "T": -2, "L": -1})

# Attempting in-place addition with incompatible dimensions
try:
    pressure += 5.0
except ValueError as e:
    print(f"Error: {e}")
    # Output: Error: Add operation for different physical dimensions is not allowed.
```

Program 5. *In-place operation with dimensions*

2.4. Array Operations

This section demonstrates how `PhlowerTensor` supports array operations while maintaining dimensional consistency. For example, Program 6 shows that multiple velocity tensors can

be stacked along a new dimension. The resulting tensor remains `PhlowerTensor` with appropriate physical dimension, even though the standard `torch.stack` is called.

```python
from phlower import phlower_tensor
import torch

# Example: Stacking velocity tensors with compatible dimensions
# Assuming we have multiple velocity tensors with dimensions L^1 T^-1
velocity1 = phlower_tensor(torch.rand(10, 3), dimension={"L": 1, "T": -1})
velocity2 = phlower_tensor(torch.rand(10, 3), dimension={"L": 1, "T": -1})

# Stacking along a new dimension (e.g., time)
stacked_velocity = torch.stack([velocity1, velocity2])
print(stacked_velocity.shape)
# Output: torch.Size([2, 10, 3])
print(stacked_velocity.dimension)
# Output: PhlowerTensor with dimensions L^1 T^-1 and shape (2, 10, 3)
```

Program 6. *Stacking tensors with compatible dimensions*

3. Use Cases

This section presents potential use cases of PhlowerTensor in scientific machine learning and physics-informed models. PhlowerTensor's ability to track physical dimensions and enforce dimensional consistency makes it a valuable tool for ensuring the correctness of computations in these domains.

3.1. Preventing Dimensional Inconsistencies in Physics-Informed Models

In physics-informed neural networks (PINNs), the loss function often includes terms derived from differential equations. For example, when modeling a system governed by Navier-Stokes equations, the loss function is composed of terms like the continuity equation and momentum equations. These equations involve physical quantities such as velocity, pressure, and density, each with specific physical dimensions. PhlowerTensor can be used to ensure that these terms are dimensionally consistent, preventing errors that could arise from mixing quantities with incompatible dimensions.

3.2. Enforcing Dimensional Validity in Feature Engineering

Phlower can reject invalid operations in feature engineering (e.g., subtracting pressure from velocity) due to its enforcement of dimensional consistency. This capability is particularly useful in domains such as fluid dynamics (e.g., aerodynamic analysis of vehicles), where data includes a variety of physical quantities with different dimensions.

4. Additional Features

4.1. YAML-Based Definition

Phlower provides a YAML-based model definition system that allows users to define, configure, and reuse machine learning models efficiently. Program 7 shows an excerpt of an example YAML file. This feature simplifies experimentation by enabling users to modify model architectures and hyperparameters without changing the underlying code.

```yaml
- nn_type: MLP
  name: ENCODER1
  input_keys:
    - feature1
  output_key: mlp1
  destinations:
    - Concat
  nn_parameters:
    nodes: [-1, 20, 200]
    activations: ["relu", "relu"]
```

Program 7. *Example of YAML-based model definition in Phlower*

4.2. Shape Semantics

Each index in the shape of a `PhlowerTensor` carries semantic meaning, a concept referred to as *shape semantics* in Phlower. Program 8 illustrates how `PhlowerTensor` can represent time-series pressure using shape semantics. This positional information is crucial for ensuring that tensor operations remain semantically meaningful and consistent with the underlying physical model.

```python
from phlower import phlower_tensor
import torch


# Example: PhlowerTensor representing pressure.
# Here, we assume that the number of time steps is 10, the number of spatial points is 100.
time_series_pressure = phlower_tensor(
    torch.rand(10, 100, 1),
    dimension={"M": 1, "T": -2, "L": -1},
    is_time_series=True,
)

print(time_series_pressure.is_time_series)
# Output: True

print(time_series_pressure.n_vertices())
# Output: 100

# Accessing the last time step of the time series
pressure_at_last_time_step = time_series_pressure[-1]

# Phlower automatically detects that accessed tensor is not a time series
print(pressure_at_last_time_step.is_time_series)
# Output: False
```

Program 8. *Shape Semantics in PhlowerTensor*

4.3. Output of model structure

Phlower provides a feature to export the model structure in a Mermaid diagram format, making it easier to integrate them into documentation. Figure 2 shows an example of model architecture diagram. This feature is useful for visualizing model structure and understanding the relationships between different components.

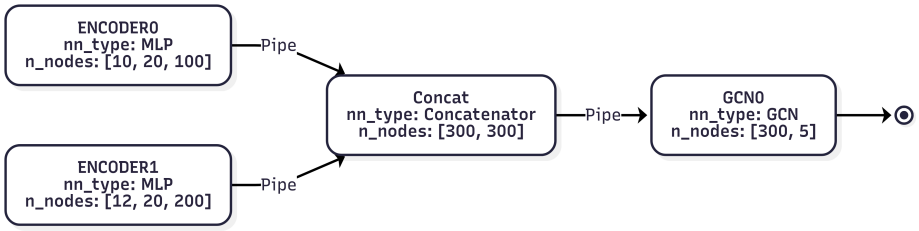


Figure 2. *Example of a model structure diagram generated by Phlower.*

5. Related Work

Several libraries and frameworks have been developed to support physical dimensions in scientific computing.

- **Pint** [4]: A Python library for handling physical quantities with units. It allows users to define quantities such as `5 meters` or `25 degrees Celsius`, and supports conversions between different unit systems, e.g., from meters to feet or from Celsius to Fahrenheit. Pint provides `Quantity` objects that encapsulate both the numeric value and its unit, and supports NumPy-like operations while preserving unit consistency.
- **Python Quantities** [5]: A library that extends NumPy to support physical quantities with units. It enables dimensional analysis and unit conversions in scientific computations. `Python Quantities` is designed to be compatible with NumPy arrays, allowing seamless integration with NumPy operations.

In contrast to Pint or Python Quantities, Phlower has two key differences. First, it does not provide unit conversion functionalities; instead, it focuses on enforcing **dimensional consistency** in tensor operations. Second, `PhlowerTensor` represents physical dimensions as an exponent vector of physical quantities (such as mass, length, and time) and stores this metadata alongside a PyTorch tensor. This design enables seamless integration with PyTorch, supporting GPU acceleration and automatic differentiation while tracking and validating the physical dimensions of tensor computations.

Thus, while Pint and Python Quantities aim to provide general-purpose unit handling and conversions in scientific computing built on NumPy, Phlower is specifically designed to integrate dimension checking into deep learning workflows built on PyTorch.

6. Limitations

- **Runtime Overhead**: Dimensional consistency is enforced at runtime, which introduces additional computational overhead. This may impact performance in scenarios where a large numbers of tensor operations are performed, particularly in real-time applications.
- **Partial Compatibility with PyTorch**: Although `PhlowerTensor` is designed to integrate with PyTorch, certain advanced features such as in-place operations or custom autograd functions may require manual adaptation or may not be fully supported.

7. Conclusion and Future Work

In traditional numerical simulations of physical phenomena, physical dimensions play a critical role in ensuring the correctness of computations. To bring this advantage into deep learning, Phlower integrates physical dimensional information into PyTorch tensor objects. By enforcing dimensional consistency, Phlower enables users to avoid errors and improve the reliability of their deep learning models. We believe that Phlower will serve as a valuable tool for researchers and engineers working at the intersection of deep learning and physical simulation.

Future work includes extending the library to support more complex physical models and enhancing the YAML-based model definition system.

References

[1] M. Raissi, P. Perdikaris, and G. Karniadakis, “Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations,” *Journal of Computational Physics*, vol. 378, pp. 686–707, 2019, doi: https://doi.org/10.1016/j.jcp.2018.10.045.

[2] A. Paszke *et al.*, “PyTorch: An Imperative Style, High-Performance Deep Learning Library,” in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett, Eds., 2019. doi: https://doi.org/10.48550/arXiv.1912.01703.

[3] D. Newell and E. Tiesinga, “The International System of Units (SI), 2019 Edition,” Gaithersburg, MD, 2019. doi: https://doi.org/10.6028/NIST.SP.330-2019.

[4] H. E. Grecco, “Pint: Physical quantities package.” [Online]. Available: https://github.com/hgrecco/pint

[5] D. Dale, “Python Quantities: Support for physical quantities with units, based on NumPy.” [Online]. Available: https://github.com/python-quantities/python-quantities

Open Access | https://doi.org/10.25080/frwc3537

SciPy 2025

July 7 - July 13, 2025

Proceedings of the 24th Python in Science Conference
ISSN: 2575-9752

SciPy Proceedings: An Exemplar for Publishing Computational Open Science

Rowan Cockett[1,2,3], **Franklin Koch**[1,2], and **Steve Purves**[1,2,3]

[1]Project Jupyter, [2]Curvenote Inc., [3]Continuous Science Foundation

Abstract

The SciPy Proceedings have served as a cornerstone of scholarly communication within the scientific Python ecosystem since their introduction in 2008. In 2024, the publication process underwent a significant transformation, adopting a new open-source infrastructure built on MyST Markdown and Curvenote. This transition enabled a web-first, interactive publishing workflow that improves reproducibility, readability, and metadata quality. Authors submit articles through GitHub, receive new structured feedback via continuous integration (CI), and benefit from new live previews, automated PDF generation with Typst, and archival Journal Article Tag Suite (JATS) XML export. This paper details the infrastructure, authoring workflow, and technical tooling that powers the new SciPy Proceedings. We demonstrate how these advances support executable content, interactive figures, and modern metadata standards—offering a model for computational publishing in other scientific communities.

Keywords open science, jupyter, scientific communication, publishing, authoring, reproducibility

1. Introduction

Scientific publishing faces a fundamental disconnect between published research and the computational methods that produced it. While researchers increasingly rely on code, data, and interactive computational workflows, traditional academic publishing remains rooted in static, paper-based formats that cannot capture the dynamic nature of modern scientific inquiry. This is particularly acute in computational science, where reproducibility requires access to executable code, interactive visualizations, and the ability to verify results through direct experimentation.

The reproducibility crisis in science has been well-documented, with studies showing that a significant portion of published research cannot be replicated due to missing code, data, or insufficient methodological detail [1], [2]. In computational research, this problem is amplified by the inherent complexity of software-based methods, where subtle differences in environments, dependencies, or implementation details can lead to dramatically different results [3]. Traditional PDF-based publishing, while excellent for narrative content, fundamentally cannot execute code, provide interactive exploration of data, or adapt to different computational environments.

Since its inception in 2008, the SciPy Proceedings have offered a high-quality venue for publishing scientific and technical contributions built on the Python ecosystem. The SciPy Proceedings are an opportunity for accepted speakers at the SciPy Conference to submit a peer-reviewed publication, and there have been over 350 peer-reviewed articles published. The articles often represent a chance to document or understand the motivations behind new scientific software or tools built for the scientific Python community. This type of work is critical for advancement across many research disciplines, but traditional academic

Published Jul 10, 2025

Correspondence to
Rowan Cockett
rowan@curvenote.com

Open Access

Copyright © 2025 Cockett *et al.*. This is an open-access article distributed under the terms of the Creative Commons Attribution 4.0 International license, which enables reusers to distribute, remix, adapt, and build upon the material in any medium or format, so long as attribution is given to the creator.

journals can be reluctant to publish it. Without the SciPy Proceedings, many of these articles may not exist.

In 2024, the SciPy Proceedings committee reimagined the entire publishing workflow by adopting modern, open-source community-built infrastructure built on MyST Markdown and Curvenote. This transition enabled the SciPy Proceedings to be part of a larger movement and ecosystem of open source publishing tools and workflows. The result was a new model of computational publishing that supports executable content, high-quality structured metadata, and a web-first reading experience with interactive features. This shift not only enhances the publishing experience for authors and reviewers but also significantly improves the way readers interact with scientific content, fundamentally advancing the principles of open science and reproducible research.

Figure 1. *Side-by-side comparison of a 2023 Proceedings listing page, which only included titles, authors, and links to PDFs, and the 2024 Proceedings listing in the new format, using modern web components to render thumbnails and additional metadata.*

2. A New Model for Scientific Publishing

The most significant improvement in the 2024 Proceedings is the transition from static PDFs to web-native articles rendered directly from MyST Markdown (https://mystmd.org). Each article is hosted by Curvenote on the SciPy Proceedings website as an interactive web page, preserving semantic structure and supporting accessibility, deep linking, and citation preview. This transition addresses the fundamental limitations of traditional publishing by enabling the kind of interactive, executable content that computational research requires.

2.1. Executable Content and Reproducibility

The new format enables true reproducibility by allowing readers to execute code directly within the article. Articles can include executed Jupyter Notebooks (e.g. A. Lujan [4]) with full output from code cells already present, interactive code blocks that can be re-run, live data connections that pull from versioned repositories, and environment specifications that document the exact computational environment used. This executable content transforms the reading experience from passive consumption to see the computational work directly integrated into the reading experience. This direct engagement builds trust in the research and enables the kind of iterative exploration that drives scientific discovery.

2.2. Enhanced Discoverability and Navigation

The web-native format also provides discoverability through structured content and semantic markup. Articles feature linkable sections, figures, and code blocks that enable precise citation and reference (e.g. to a specific section or figure), semantic structure that search engines can understand and index, rich metadata that connects articles to related research, datasets, and software, and cross-references that link to related work, dependencies, and extensions.

2.3. Interactive Exploration and Visualization

The new format supports the kind of interactive exploration that computational research demands: 3D interactive visualizations using tools like `k3d`, `pythreejs`, or `vtk.js` that respond to user input, dynamic plots that update based on parameter changes, embedded computational widgets that allow real-time experimentation, and responsive design that adapts to different devices and screen sizes. These interactive elements transform static figures into dynamic tools for understanding. Readers can explore data from multiple angles, test hypotheses through direct manipulation, and gain deeper insights through hands-on exploration.

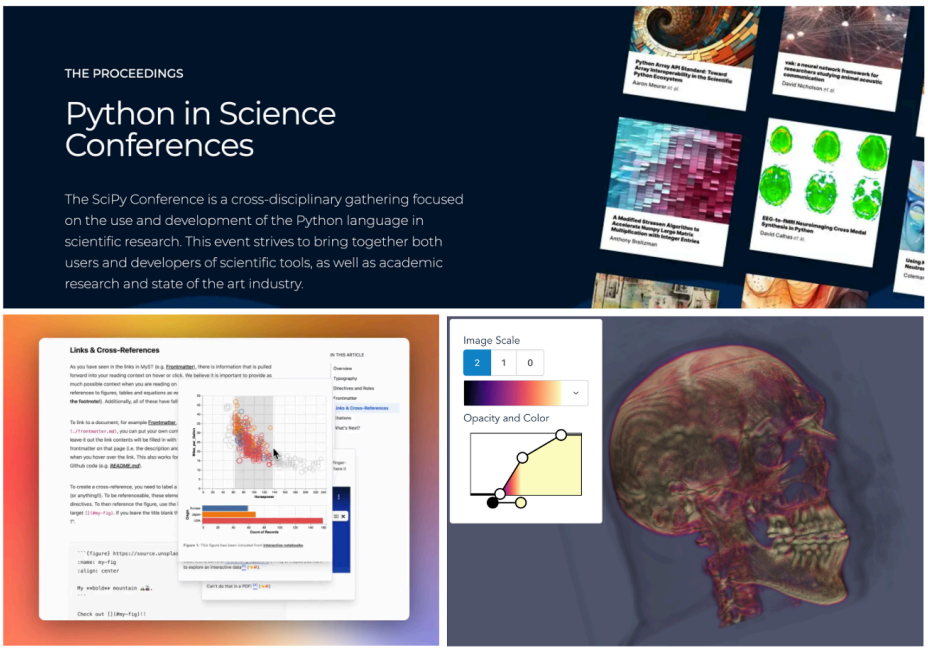


Figure 2. *We have focused on web-native rendering, hover previews and including interactive figures. Showing content from M. McCormick and P. Elliott [5] and R. Cockett, S. Purves, F. Koch, and M. Morrison [6].*

2.4. The Authoring Workflow

The new authoring and submission workflow is aimed at transforming the traditional publishing process into a collaborative, iterative experience that leverages modern software development practices. Authors write their articles using a standardized MyST template available on GitHub from SciPy. Each submission consists of a `myst.yml` file capturing comprehensive article metadata, including authorship and author Open Researcher and Contributor IDs (ORCIDs), affiliations and Research Organization Registry IDs (RORs), keywords, and licensing; Markdown or Jupyter Notebook source files using MyST syntax, including the main article and any supplementary material; and static assets such as figures and datasets, with clear organization and documentation.

After previewing the article locally with `myst start`, authors open a Pull Request on GitHub. This submission triggers a GitHub Actions Continuous Integration (CI) pipeline that performs comprehensive quality checks: structural validation of the document to ensure proper formatting and organization, metadata schema validation (e.g., required ORCID, DOIs for references) to ensure completeness, content validation including word count limits, figure references, and citation completeness, rendering to HTML for preview to catch

formatting issues early, PDF generation via Typst to ensure print compatibility, and Journal Article Tag Suite (JATS) XML export for archival and indexing.

This automated quality assurance process serves multiple purposes. It catches common errors early in the submission process (See Figure 3), reducing the burden on reviewers and authors. It ensures consistency across all articles in the proceedings, improving the overall quality of the publication. It provides immediate feedback to authors, enabling rapid iteration and improvement—with authors getting structured and actionable feedback in less than a minute.

The GitHub-based workflow, popularized by the Journal of Open Source Software (JOSS) [7], [8], enables a collaborative review that is more transparent and effective than traditional peer review processes. Reviewers often leave line-level comments directly on the source content, enabling precise feedback, suggest changes through GitHub's suggestion feature, making it easy for authors to accept improvements, track the evolution of articles through the commit history, understanding how feedback was incorporated, and collaborate with other reviewers through threaded discussions, building consensus on improvements.

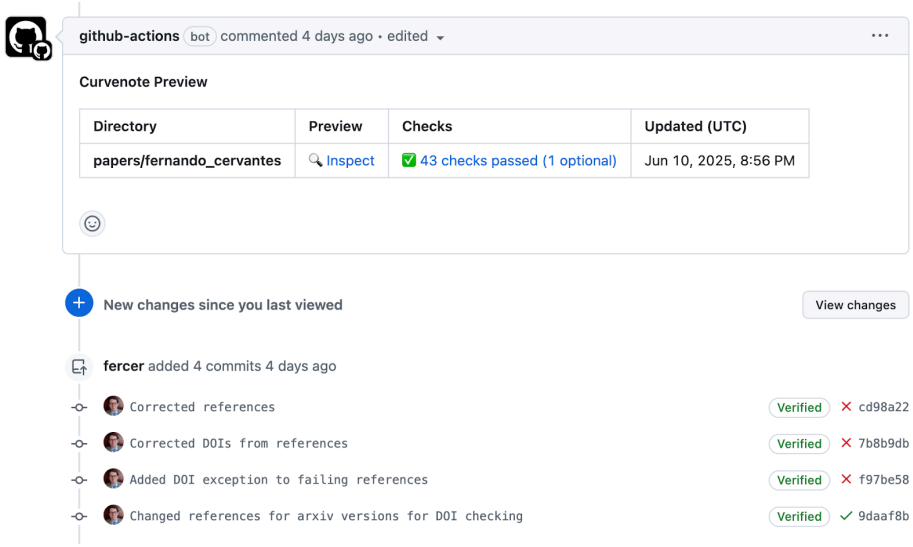


Figure 3. *Screenshot of a GitHub Pull Request showing CI failure due to missing ORCID metadata, with the corrected version in a follow-up commit (Cervantes-Sanchez, 2025).*

3. Technical Foundation

The transition to the new SciPy Proceedings infrastructure required consideration of tool selection to balance functionality, openness, accessibility, and community needs. Each component was chosen to advance the goals of reproducible, interactive scientific publishing while maintaining the open-source ethos of the SciPy community.

The markup language that the proceedings used for 16 years was reStructuredText and a restricted version of LaTeX was added in 2022. In 2024, MyST Markdown was selected as the core authoring format for several reasons. As an extension of standard Markdown, it provides a familiar, accessible authoring experience while adding the semantic structure needed for scientific publishing. MyST supports executable content through Jupyter integration, rich metadata through YAML frontmatter, cross-references and citations, mathematical notation through LaTeX syntax, and multiple output formats including HTML, PDF, and JATS XML. The tooling also supports LaTeX directly, so the options introduced in 2022 were still supported. MyST is developed as an open-source project with strong community

involvement, ensuring that the format evolves with the needs of scientific publishing while remaining accessible to all contributors; the project is governed under Project Jupyter.

The Curvenote CLI is based on top of MyST and is also available under MIT license, it adds specific metadata checks that can be run locally as well as through the open source GitHub actions. The PDF template is SciPy-specific and can be used for any MyST content or forked for other proceedings. The Curvenote CLI can be used to send a draft of a submission to web hosting infrastructure, managed by Curvenote, to display the built manuscripts and publish them; the Curvenote platform is commercial and enables publishing management and the hosting of in-progress drafts. For institutions or individuals who prefer to self-host this style of proceedings, the open-source components can combined independently using MyST Markdown directly (see MyST documentation for deployment instructions), providing full control over the end-to-end infrastructure. Examples of this include A. Karakuzu [9], a journal that hosts interactive preprints and has a similar vision for interactive scientific narratives [9]; Neurolibre use very similar open-source infrastructure, including MyST Markdown and Open Journals, which powers JOSS.

The transition from LaTeX to Typst represents a significant modernization of the PDF generation process. LaTeX, while powerful, is complex, fragile and has large dependency chains. Build times can be unpredictable, and subtle changes in document structure can cause complex compilation failures. Typst offers significantly faster compilation times and more predictable behavior, reducing the friction in the authoring and review process. Typst provides a more modern, programmable approach to typesetting that enables the creation of highly customized templates. This allowed us to build a SciPy-specific template that maintains consistency across all articles while supporting the advanced formatting requirements of scientific publishing (See Figure 4). Typst is developed as an open-source project, ensuring that the typesetting engine remains accessible and modifiable by the community.

The transition to Typst has delivered measurable improvements: PDF generation is now 5-10x faster than with LaTeX, compilation failures are rare and easily diagnosed, the SciPy template can be easily forked, modified or extended, and it supports advanced typography and layout features (See Figure 4).

The full archive of scientific content over the 18 years has been transitioned to an open GitHub archive that includes the built assets (JATS, PDF, and Manuscript Exchange Common Approach (MECA) files) for every paper; these were previously available through each branch of the SciPy Proceedings repository.

3.1. Build System and Output Formats

The SciPy Proceedings infrastructure is built on Curvenote's GitHub Actions and the Curvenote CLI, both available through the MIT License. Upon opening a submission pull request and on each subsequent commit, content is automatically validated. This validation ensures all content is formatted correctly so the article can fully build. It also checks that all required metadata is in place and all submission guidelines, such as word count, number of keywords, etc. are met. After this validation the submission is rendered across several formats using a reproducible build pipeline:

- **Web (HTML)**: Rendered using MyST Markdown and published to the Proceedings site.
- **PDF**: Generated using Typst, replacing LaTeX for greater speed, reliability, and customizability.
- **JATS XML**: Introduced in 2024, this machine-readable format supports long-term preservation and interoperability with scholarly indexing systems.

- **Previews**: The previews from the Pull Request are hosted by Curvenote, allowing reviewers and authors to inspect the rendered output, exactly as it will appear in the final proceedings publication[1].

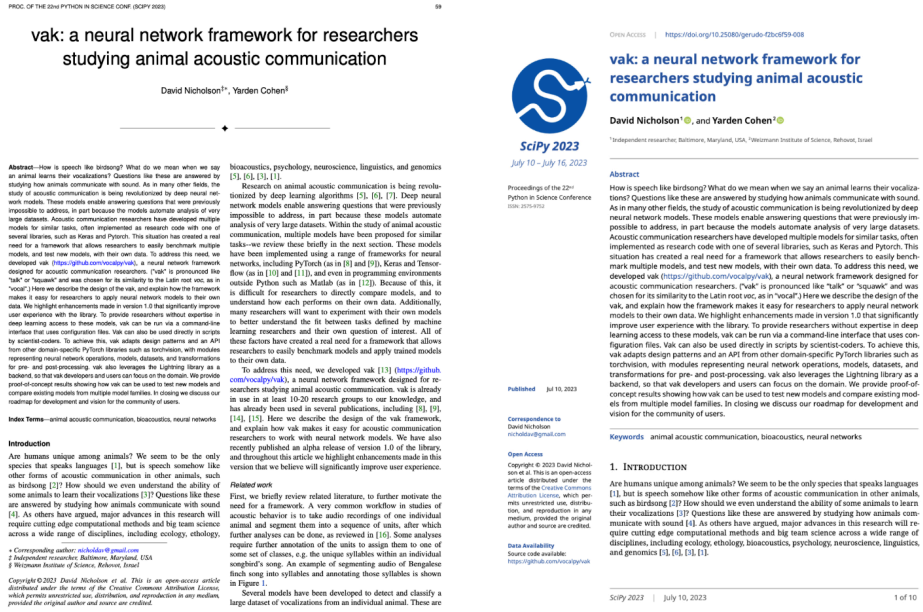


Figure 4. *Comparison between 2023 and 2024 PDF templates, 2024 improved the branding and template to be more modern; the template now uses Typst over LaTeX, which also improves the build speed. The template is available under the MIT license.*

The transition to Typst allowed us to build a fully customized SciPy Proceedings PDF template, ensuring consistency with the new web style while supporting traditional academic formatting and citation standards.

3.2. Metadata and Interoperability

The move to structured MyST Markdown and schema-validated metadata unlocks new possibilities for reuse, attribution, and discoverability. Metadata is required at the submission stage and validated throughout the review process, including author names and ORCIDs, institutional affiliations (RORs), keywords, license, and abstract, and DOI for citations and associated datasets or software.

Each component of an article (e.g., figures, equations, or code blocks) is preserved in a structured way, improving reusability. For example, figures can be referenced externally, cited independently, or embedded into institutional repositories without modification. JATS XML export supports interoperability with discovery engines and indexing services, while also laying the foundation for cross-journal content federation and machine-readable citation networks.

As more conferences, journals, preprint servers, and MyST-powered documentation sites adopt the structured information and metadata standards, these resources create an interconnected network of scientific knowledge that is browsable, inter-linked, and discoverable. Each article becomes a node in this network, with rich metadata, cross-references, and executable content that can be linked, cited, and built upon from any project. This can allow for reuse from other projects, for example:

[1]For fully open-source workflows, these previews can be hosted by GitHub pages or similar static rendering sites (see GitHub Pages deployment).

1. Referencing figures from other publications, [4];
2. Pulling in context via hover previews from [6];
3. Showing rich interactive visualizations from [5]; or
4. Showing videos from [10].

These cross-references provide direct access to a modular component of a research article or resource and can be shown in context (try hovering over one of the links above in the web-version, or watch the video in Figure 5). These are standard features in any MyST Markdown powered document [6], and provide exciting possibilities for new in-context reading experiences.

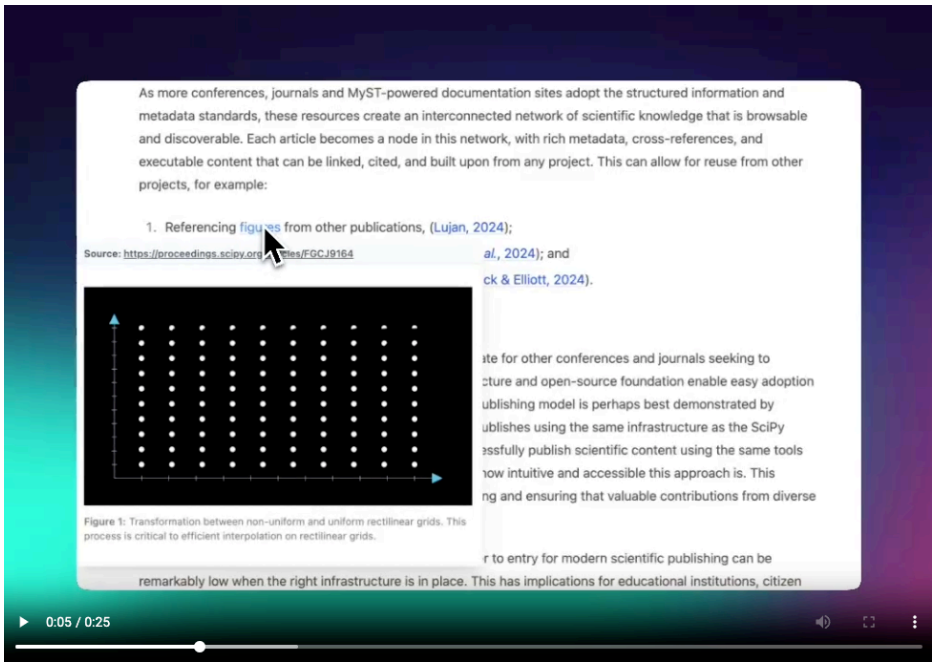


Figure 5. *Hover cross-references between articles, using the MyST metadata and content standards.*

3.3. Demonstrating Accessibility

The SciPy Proceedings model provides a replicable template for other conferences and journals seeking to modernize their publishing practices. The modular architecture and open-source foundation enable easy adoption by other scientific communities. The accessibility of this publishing model is perhaps best demonstrated by Morganton Scientific, a high school science journal that publishes using the same infrastructure as the SciPy Proceedings [11]. The fact that high school students can successfully publish scientific content using the same tools and workflows as professional researchers demonstrates how intuitive and accessible this approach is. This accessibility is crucial for democratizing scientific publishing and ensuring that valuable contributions from diverse communities can be shared and recognized.

The success of Morganton Scientific shows that the barrier to entry for modern scientific publishing can be remarkably low when the right infrastructure is in place. This has implications for educational institutions, citizen science projects, and emerging research communities that might otherwise struggle to establish traditional publishing channels.

4. Conclusion

As computational research continues to grow in importance across all scientific disciplines, the need for publishing infrastructure that can effectively communicate this work will only increase. The SciPy Proceedings demonstrate that this challenge can be met through community-driven development, open-source tools, and a commitment to the fundamental principles of scientific communication. The 2024 transition addresses the core challenges of reproducibility, interactivity, and discoverability in scientific publishing. Through modern web technologies and open-source infrastructure, the Proceedings have demonstrated that scientific communication can evolve to meet the needs of modern research while maintaining the rigor and quality that the scientific community expects.

The new SciPy Proceedings infrastructure provides a working example of how scientific publishing can embrace the interactive, executable nature of computational research. By combining MyST Markdown, Curvenote, GitHub-native review workflows, and Typst, the Proceedings offer authors and readers a more dynamic, structured, and metadata-rich experience for computational sciences.

The infrastructure is designed to be replicable by other scientific communities. The modular architecture, open-source components, and documented processes provide a template that can be adapted for different domains and use cases. The success of the SciPy Proceedings demonstrates that this approach can work at scale and provides a path for other communities seeking to modernize their publishing practices.

Acknowledgements

We gratefully acknowledge the contributions of Chris Calloway, Meghann Agarwal, Stefan van der Walt, and Jim Weis for their support, coordination, and feedback throughout the transition to the new SciPy Proceedings infrastructure. Their efforts were instrumental in shaping the submission process, reviewing tooling, and ensuring a smooth experience for authors and reviewers alike.

We thank the current SciPy 2025 Proceedings Committee — Amey Ambade, Ana Comesana, Sean Freeman, Sanhita Joshi, Charles Lindsey, and Hongsup Shin — for their dedication and hard work in supporting authors, managing reviews, and ensuring the quality and integrity of this year's volume.

We also thank the many members of past SciPy Proceedings committees and infrastructure teams whose work laid the foundation for this transformation. The continuity, care, and community leadership built over the last 17 years have made this evolution possible.

References

[1] "Estimating the reproducibility of psychological science," *Science*, vol. 349, no. 6251, 2015, doi: 10.1126/science.aac4716.

[2] M. Baker, "1,500 scientists lift the lid on reproducibility," *Nature*, vol. 533, no. 7604, pp. 452–454, 2016, doi: 10.1038/533452a.

[3] V. Stodden *et al.*, "Enhancing reproducibility for computational methods," *Science*, vol. 354, no. 6317, pp. 1240–1241, 2016, doi: 10.1126/science.aah6168.

[4] A. Lujan, "multinterp: A Unified Interface for Multivariate Interpolation in the Scientific Python Ecosystem," in *Proceedings of the 23rd Python in Science Conference*, in SciPy. 2024, pp. 1–19. doi: 10.25080/fgcj9164.

[5] M. McCormick and P. Elliott, "ITK-Wasm: Universal spatial analysis and visualization," in *Proceedings of the 23rd Python in Science Conference*, in SciPy. 2024, pp. 256–279. doi: 10.25080/tcfj5130.

[6] R. Cockett, S. Purves, F. Koch, and M. Morrison, "Continuous Tools for Scientific Publishing: Using MyST Markdown and Curvenote to encourage continuous science practices," in *Proceedings of the 23rd Python in Science Conference*, in SciPy. 2024, pp. 121–136. doi: 10.25080/nkvc9349.

[7] A. M. Smith *et al.*, "Journal of Open Source Software (JOSS): design and first-year review," *PeerJ Computer Science*, vol. 4, p. e147, 2018, doi: 10.7717/peerj-cs.147.

[8] P. Diehl, C. Soneson, R. C. Kurchin, R. Mounce, and D. S. Katz, "The Journal of Open Source Software (JOSS): Bringing Open-Source Software Practices to the Scholarly Publishing Communityfor Authors, Reviewers, Editors, and Publishers," *Journal of Librarianship and Scholarly Communication*, vol. 12, no. 2, 2025, doi: 10.31274/jlsc.18285.

[9] A. Karakuzu, "Toward a woven literature: Open-source infrastructure for reproducible publishing," 2025, doi: 10.55458/neurolibre.00041.

[10] F. S. Bao, M. Qi, R. Tu, and E. Wan, "Funix - The laziest way to build GUI apps in Python: Make it (the app) before they fake it (in Figma or on a napkin)," in *Proceedings of the 23rd Python in Science Conference*, in SciPy. 2024, pp. 174–195. doi: 10.25080/jfyn3740.

[11] *Morganton Scientific*, vol. 2, 2024, doi: 10.62329/dwgm3685.

Open Access | https://doi.org/10.25080/wcfp5784

SciPy 2025

July 7 - July 13, 2025

Proceedings of the 24th Python in Science Conference
ISSN: 2575-9752

Empowering Learners - Teaching Reproducible Research with Open-Source Tools

Deborah Khider[1], **Julien Emile-Geay**[2], **David Edge**[3], and **Nicholas McKay**[3]

[1]University of Southern California Information Sciences Institute, [2]University of Southern California, [3]Northern Arizona University

Abstract

Reproducibility and open science are increasingly recognized as essential components of rigorous geoscientific research. However, gaps in training often hinder early-career scientists from fully adopting best practices in computational workflows and FAIR (Findable, Accessible, Interoperable, Reusable) publishing. The Facilitating Reproducible Open Geoscience (FROGS) initiative addresses this need by offering an ongoing series of integrated courses and self-paced online modules designed to build capacity in Python and R programming, time series analysis, and open science publishing within the geosciences. To date, three sequential courses have been completed, and this paper reports on these initial offerings. During the courses, participants engaged in hands-on learning activities combining synchronous instruction with asynchronous exercises, supported by the LeapFROGS platform. Survey results demonstrate high participant satisfaction, increased confidence in applying reproducible research methods, and active contributions to open source geoscience projects. This paper details the course design, curriculum content, participant outcomes, and the broader impact of FROGS in fostering a sustainable community of practice dedicated to open and reproducible geoscience. Our findings underscore the critical role of integrated training programs in advancing open science and highlight strategies for scaling such initiatives across scientific domains.

Keywords FAIR principles, reproducible science, open science

Published Jul 10, 2025

Correspondence to
Deborah Khider
khider@usc.edu

Open Access

Copyright © 2025 Khider *et al.*. This is an open-access article distributed under the terms of the Creative Commons Attribution 4.0 International license, which enables reusers to distribute, remix, adapt, and build upon the material in any medium or format, so long as attribution is given to the creator.

1. Introduction and Motivation

Sharing research data, software, and workflows enhances reproducibility, collaboration, and the directions of future research [1] and is fundamental to building a FAIR open science ecosystem. The desire to share and reuse scientific data and software has grown in recent years as funders [2], [3], [4] and publishers [5], [6], [7] have introduced open science policies emphasizing reproducibility, and as scientists increasingly recognize the benefits of open science [8], [9]. Consequently, the last decade has seen a proliferation of frameworks to promote open source resources, collaboration among scientists, and sharing of scientific research products. One critical element to the success and long-term sustainability of these resources is the training of scientists, especially in their early-career, in their use. This training needs to take place across scientific practice and publishing.

Facilitating Reproducible Open Geoscience (FROGS) is a U.S. National Science Foundation (NSF) funded initiative designed to speed up the integration of open source tools into the research and publication workflows of geoscientists. FROGS has offered three seven-week long courses on geoscience analysis in Python and R and FAIR publishing principles. The

courses emphasized science publishing as an integral part of a research product. Each course included a synchronous workshop that introduced overarching concepts, followed by asynchronous, detailed online materials combined with biweekly office hours designed to help participants integrate the resources into their practice. The asynchronous component was facilitated by the LeapFROGS learning platform [10], which provided summaries of the key concepts, curated additional resources from the open science community, and offered challenges for learners to assess their comprehension.

This paper provides an overview of the training activities in Overview of training activities, detailing the goals and design of the LeapFROGS platform, before discussing educational outcomes and contributions to open science in Contributions to open science.

2. Overview of training activities

2.1. LeapFROGS

The LeapFROGS platform combines lecture content with self-graded exercises to deliver self-paced modules covering diverse aspects of (geo)scientific research. Built using Gatsby and supported by a myBinder backend [11], it offers a Python sandbox environment for hands-on learning, bypassing environment setup difficulties that are often a limiting step for first-timers. The platform features seven modules that explore different facets of scientific practice and publishing. These modules leverage open source tools from the scientific Python community and are specifically tailored to geoscience applications. The platform's primary aim is to serve as a gateway into the extensive Python educational ecosystem.

The first module, *Introduction to Python*, covers fundamental concepts including numbers, variables, logic, strings, lists, tuples, sets, dictionaries, conditionals, loops, functions, and classes. The lessons reference the Trinket Python course [12], while additional coding exercises on LeapFROGS offer geoscience-relevant challenges to reinforce learning. The coding exercises are designed as fill-in-the-blank tasks (Figure 1). A key advantage of the platform is its ability to provide hints and reveal solutions to learners who encounter difficulties.

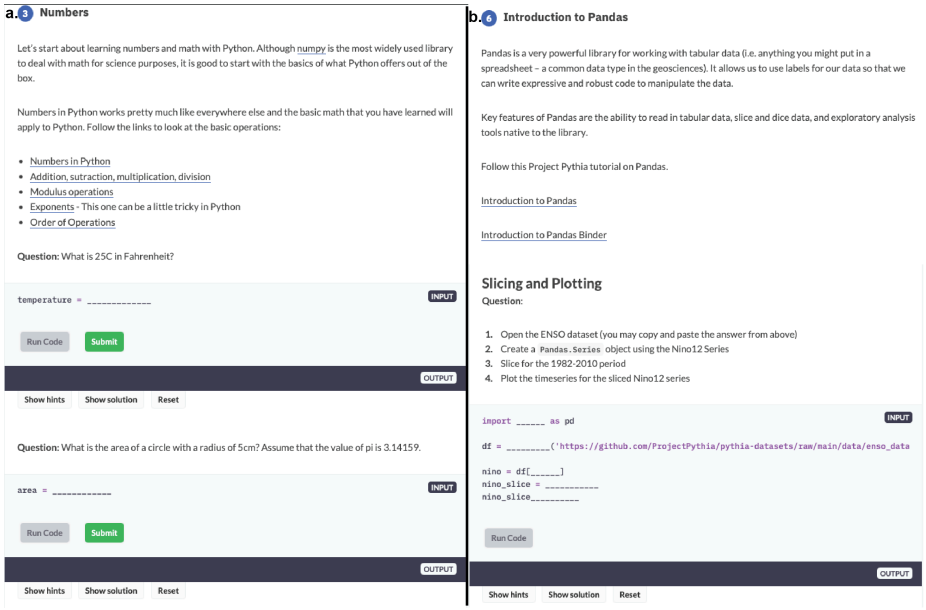


Figure 1. *Example of training model on the LeapFROGS platform. (a) An exercise on numbers in Python with links to the Trinket Python course, and (b) an tutorial on Pandas using materials developed by Project Pythia. LeapFROGS tests understanding of these concepts by asking to fill in the blank in a code cell than*

can be executed on myBinder so learners can run their code to check against the solution. Learners can also ask for helpful hints or see the solution.

The second module, *The Scientific Python Stack*, introduces libraries common to scientific Python such as Jupyter [13], NumPy [14], pandas [15], [16], Matplotlib [17], Cartopy [18], Seaborn [19], statsmodels [20], scikit-learn [21], [22], PyTorch [23], and Xarray [24]. This module adopts the same approach as the *Introduction to Python* module, featuring lessons from Project Pythia Foundations [25], alongside tutorials from library developers and coding exercises tailored to reinforce learning within a geoscience context (Figure 1).

The third module, *Timeseries Analysis*, introduces key concepts and practicums about time-series analysis, a common task in geoscience studies. The lecture materials were adapted from Emile-Geay's class at the University of Southern California (USC) on data analysis in the Earth and Environmental Sciences and its accompanying e-book [26]. The module covers common data processing techniques, measures of association between timeseries, the use of surrogates for significance testing, and spectral and wavelet analyses. The module is supported by a Jupyter Book [27] that makes use of the Pyleoclim software package [28].

The fourth module, *The Scientific Paper of The Future*, introduces concepts in FAIR scientific publishing. As its name indicates, this module presents an updated version [29] of the Geoscience Paper of the Future (GPF) [1]. The GPF represented one of the earliest community efforts to fully document, share, and cite all research products, including data, software, and computational provenance (i.e., describing the workflow used in a study, which links together data and software with their associated parameter values for reproducibility). Although the GPF was published before the FAIR principles [30], [31], [32], the guidelines outlined in this paper allow for compliance with FAIR. The updated materials reflect this compliance and provide additional guidance on the technologies such as Binder [11] that support FAIR, open and reproducible science. Learners on the platform are able to test their understanding of these concepts using a multiple choice quiz (Figure 2).

The fifth module, *Using GitHub for Your Research*, focuses on GitHub as a platform to collaborate, share scientific workflow and software, and support project management. The module covers the basics of repositories, forks, pull requests, understanding and setting up actions for automation, and linking a GitHub repository to Zenodo to obtain a persistent identifier. This module provides a high-level overview of GitHub Actions, including their purpose and the basic structure of a workflow file, while more detailed applications—such as containerization and continuous integration—are addressed in the final two modules. Lessons from this chapter were summarized from various sources including the GitHub documentation, blogs from Medium and other sources, and Project Pythia Foundations [25].

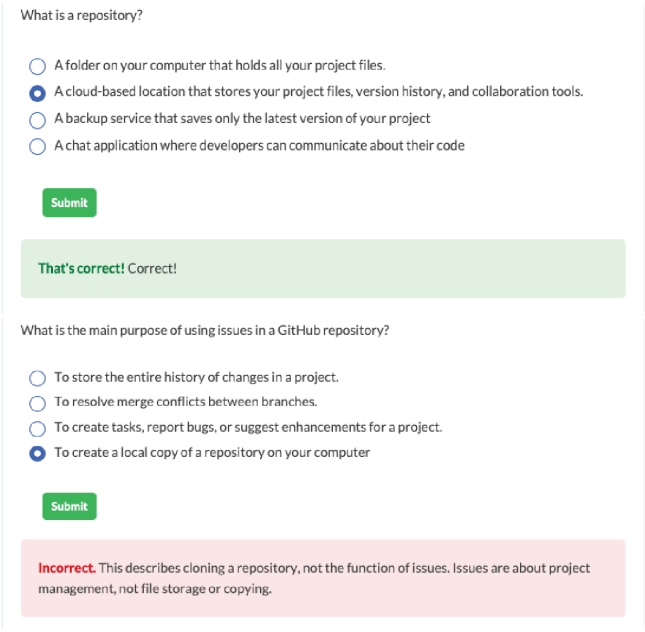


Figure 2. *Example of training model on the LeapFROGS platform. Learners can test their understanding using a self-paced quiz that provides feedback in case of an incorrect answer.*

The sixth module, *Sharing Reproducible Workflows*, provides information about using Docker and myBinder to share reproducible science results.

Finally, the last module, *Packaging Your Software for Sharing* walks through instructions and tutorials on creating Python packages, publishing it on software registries (e.g., The Python Package Index, PyPI), on creating documentation using Sphinx [33] and publishing it on readthedocs.org, on unit testing, on setting up GitHub actions for continuous integration, and on publishing on PyPI. Most of these tutorials were based on the materials developed by the PyOpenSci community.

2.2. Training activities

We facilitated three complementary training workshops designed to build geoscientists' skills in computational research, open science publishing, and software development. Participants in each workshop were selected to maximize diversity across geoscientific disciplines and backgrounds. A breakdown of participants by career stage and geoscientific discipline is given in Figure 3.

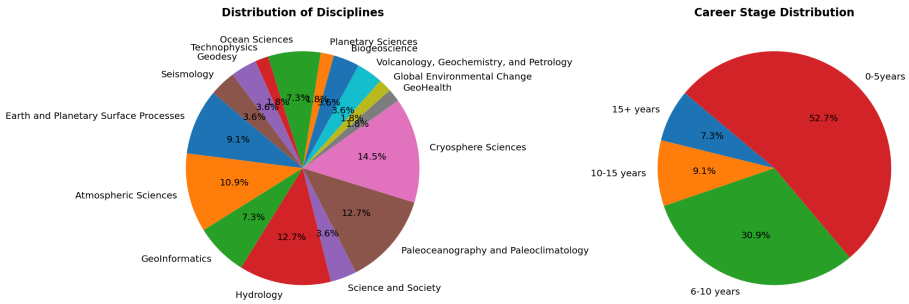


Figure 3. *Distribution of participants by geoscience disciplines and career stage across all three training workshops.*

2.2.1. *PyRATES: Python and R Analysis of Time Series*

Held in-person in June 2024 at the USC Information Sciences Institute (ISI), PyRATES offered foundational training in Python and R tailored to geoscientists, with an emphasis on time series analysis — a core expertise of the principal investigators that spans multiple geoscience domains. This workshop targeted early-career researchers with little prior programming experience and aimed to introduce the basics of scientific Python and R, statistical concepts in time series analysis, and FAIR science publishing principles. The workshop gathered 18 participants and covered modules 1 through 4 on the LeapFROGS platform. At the end of the asynchronous period, participants were asked to submit a reproducible notebook applying FAIR principles.

2.2.2. *FAIRLeap: FAIR Publishing in the Geosciences*

FAIRLeap was designed for researchers actively engaged in geoscience projects, including those preparing manuscripts or reproducibility studies. The workshop introduced FAIR publishing concepts, GitHub for project and software management, and tools such as Docker, Binder, and myBinder for sharing reproducible workflows (modules 4-6 on the LeapFROGS platform). Held virtually in February 2025, FAIRLeap gathered 21 participants, attending mostly asynchronously.

2.2.3. *Open Geoscience Hackathon*

The Open Geoscience Hackathon targeted researchers interested in open-source software development and contributions. The workshop focused on practical skills including opening pull requests, software packaging, unit testing, and continuous integration (module 7 on LeapFROGS). Held in person at USC ISI in May 2025, the hackathon gathered 17 participants. Morning sessions were reserved for high-level lectures and guided homework on the PyCatSim [34] toy package. This Python package, which simulates cat ownership, was specifically designed for this hackathon. The functions were straightforward to implement and served to illustrate object-oriented programming principles, methods for integrating data into packages, and error handling, all while demonstrating the process of contributing to an open-source project. In the afternoons, participants worked on their own package. By the end of the workshop, participants were able to share the foundation of their package on GitHub.

3. Contributions to open science

FROGS has contributed to open science in two main ways. First, it has trained the next generation of scientists in Python programming, data science libraries, time series analysis, geoscience data science, and open science sharing and publishing practices. In turn, these scientists have produced exemplars of reproducible studies within their respective communities.

3.1. Training the next generation of scientists

Exit surveys conducted following the three workshops demonstrated a highly favorable reception among participants. Specifically, 95% of respondents indicated they would recommend the workshops to colleagues, 90% reported that the activities enhanced their understanding of key concepts in geoscience practice and publishing, and 94% expressed increased confidence in applying these principles within their own research endeavors. Participants consistently highlighted the effectiveness of the workshops' hands-on, project-oriented approach, emphasizing the value of integrating lectures with practical coding exercises, such as contributing to exemplar packages like PyCatSim. Instructional modules

on automation, software testing, documentation, and version control via GitHub were particularly well received.

Nevertheless, participant feedback also identified areas for refinement. A notable challenge was accommodating the wide range of prior experience among attendees; some less-experienced participants encountered difficulties with advanced topics, including object-oriented programming, underscoring the need for preparatory materials or more structured introductory sessions. Suggestions were also made to extend the duration of workshops or incorporate additional applied exercises and case studies to foster deeper engagement. Furthermore, enhanced guidance on software environment setup and package publication workflows was recommended to mitigate technical barriers.

Taken together, these findings indicate that the workshops effectively impart practical skills and foster confidence in open science methodologies, while providing actionable insights to improve accessibility and pedagogical design for diverse participant cohorts.

3.2. Contributed workflows and software

In addition to providing comprehensive training, a significant outcome of the workshops was the active contribution of participants to open science projects. These contributions not only reinforced the practical skills acquired during the sessions but also fostered engagement within the open source geoscience community. The repositories developed or enhanced by participants span a diverse range of geoscience domains and are publicly available on GitHub. Table 1 summarizes these contributions, listing the associated scientific domains alongside the corresponding GitHub repositories.

Table 1. *Open science projects contributed by participants, with repository names shown as user/repository_name on GitHub.*

Research Area	Repository
Paleoceanography and Paleoclimatology	frozenarchives/pyRATES
Volcanology, Geochemistry, and Petrology	ruixiabai/XYplot
Ocean Sciences	ksc005/pyrates/
Earth and Planetary Surface Processes	Dewan-cpu/Decoding-Landslide-Hazard-Assessment
Atmospheric Sciences	sputhiyamadam/PYRATES_workshop
Geodesy	chongjh11/pyrates2024
Geoinformatics	IGCCP/mindat-locality
Seismology	vikkybass/PyRates-reproduc
Paleoceanography and Paleoclimatology	kurtlindberg/leafwaxtools
Cryosphere Sciences	Duyi-Li/multipol
GeoHealth	jeanmico/structuralnoisebarriers
Hydrology	preetika11
Paleoceanography and Paleoclimatology	tanaya-g/sedproxy_python
Hydrology	surabhiupadhyay/pyclimproj
Hydrology	jsacerot/Rpftools
Science and Society	nqulizada835/geocleaner
Earth and Planetary Surface Processes	lefitzpatrick/nbspredictor
Atmospheric Sciences	bosup/toy_package
Hydrology	zuhlmann/dataFrameGis

GeoInformatics	PratyushTripathy/pyrsgisHydrology

4. Conclusions

In total, the workshops engaged 56 participants spanning over a dozen geoscience subfields and multiple career stages. Participants produced 8 reproducible notebooks and contributed to 9 open-source repositories for software development, underscoring tangible skill acquisition and active engagement with open science practices. These contributions not only exemplify the immediate outputs of the training but also serve as building blocks for a sustainable community of practice that extends beyond the duration of the workshops. By fostering collaborative development and dissemination of geoscience software and workflows, FROGS has catalyzed lasting shifts in research norms toward greater openness and reproducibility.

FROGS has demonstrated that targeted training in computational methods and FAIR publishing principles can effectively embed open science practices into routine geoscience research workflows. Beyond individual capacity building, the initiative has nurtured a vibrant community of practice that continues to support collaborative development and knowledge sharing. Future efforts will focus on scaling these training models, integrating them with institutional curricula, and expanding partnerships with related open science initiatives to amplify impact. Continued evaluation of long-term participant engagement and contributions will guide ongoing refinement and ensure that FROGS remains responsive to the evolving needs of the geoscience community.

References

[1] Y. Gil *et al.*, "Toward the Geoscience Paper of the Future: Best practices for documenting and sharing research from data to software to provenance," *Earth and Space Science*, vol. 3, no. 10, pp. 388–415, 2016, doi: 10.1002/2015ea000136.

[2] R. P. Nugroho, A. Zuiderwijk, M. Janssen, and M. de Jong, "A comparison of national open data policies: lessons learned," *Transforming Government: People, Process and Policy*, vol. 9, no. 3, pp. 286–308, 2015, doi: 10.1108/tg-03-2014-0008.

[3] V. V. D. Eynden *et al.*, "Survey of Wellcome researchers and their attitudes to open research," 2016, doi: 10.6084/M9.FIGSHARE.4055448.V1.

[4] A. Zuiderwijk and M. Janssen, "Open data policies, their implementation and impact: A framework for comparison," *Government Information Quarterly*, vol. 31, no. 1, pp. 17–29, 2014, doi: 10.1016/j.giq.2013.04.003.

[5] P. Fox *et al.*, "Data and Software Sharing Guidance for Authors Submitting to AGU Journals." [Online]. Available: https://zenodo.org/record/5124741

[6] C. Erdmann *et al.*, "Guidance for AGU Authors - Jupyter Notebooks." [Online]. Available: https://zenodo.org/record/4774440

[7] *Nature Methods*, vol. 16, no. 3, p. 207, 2019, doi: 10.1038/s41592-019-0350-x.

[8] J. S. S. Lowndes *et al.*, "Our path to better science in less time using open data science tools," *Nature Ecology & Evolution*, vol. 1, no. 6, 2017, doi: 10.1038/s41559-017-0160.

[9] E. C. McKiernan *et al.*, "How open science helps researchers succeed," *eLife*, vol. 5, 2016, doi: 10.7554/elife.16800.

[10] D. Khider, J. Emile-Geay, and D. Edge, "LeapFROGS: A learning platform for Python, time series analysis and open science publishing." [Online]. Available: https://github.com/LinkedEarth/LeapFROGS

[11] Project Jupyter *et al.*, "Binder 2.0 - Reproducible, interactive, sharable environments for science at scale," in *Proceedings of the 17th Python in Science Conference*, Fatih Akici, David Lippa, Dillon Niederhut, and M. , Eds., 2018, pp. 113–120. doi: 10.25080/Majora-4af1f417-011.

[12] Trinket, "Getting started with Python." [Online]. Available: https://docs.trinket.io/getting-started-with-python#/welcome/where-we-ll-go

[13] T. Kluyver *et al.*, "Jupyter Notebooks - a publishing format for reproducible computational workflows," in *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, F. Loizides and B. Scmidt, Eds., Netherlands, 2016, pp. 87–90. [Online]. Available: https://eprints.soton.ac.uk/403913/

[14] C. R. Harris *et al.*, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, 2020, doi: https://doi.org/10.1038/s41586-020-2649-2.

[15] The Pandas Development Team, "pandas-dev/pandas: Pandas." [Online]. Available: https://doi.org/10.5281/zenodo.3509134

[16] W. McKinney, "Data Structures for Statistical Computing in Python," in *Proceedings of the 9th Python in Science Conference*, Stéfan van der Walt and Jarrod Millman, Eds., 2010, pp. 56–61. doi: https://doi.org/10.25080/Majora-92bf1922-00a.

[17] J. D. Hunter, "Matplotlib: A 2D graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007, doi: https://doi.org/10.1109/MCSE.2007.55.

[18] Met Office, "Cartopy: a cartographic python library with a Matplotlib interface," Exeter, Devon. [Online]. Available: https://scitools.org.uk/cartopy

[19] M. L. Waskom, "seaborn: statistical data visualization," *Journal of Open Source Software*, vol. 6, no. 60, p. 3021, 2021, doi: 10.21105/joss.03021.

[20] S. Seabold and J. Perktold, "statsmodels: Econometric and statistical modeling with python," in *9th Python in Science Conference*, 2010.

[21] F. Pedregosa *et al.*, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[22] L. Buitinck *et al.*, "API design for machine learning software: experiences from the scikit-learn project," in *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, 2013, pp. 108–122.

[23] A. Paszke *et al.*, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Advances in neural information processing systems*, 2019, pp. 8024–8035.

[24] S. Hoyer and J. Hamman, "xarray: N-D labeled arrays and datasets in Python," *Journal of Open Research Software*, vol. 5, no. 1, 2017, doi: 10.5334/jors.148.

[25] B. E. J. Rose *et al.*, "Pythia Foundations." 2023. doi: 10.5281/zenodo.7884571.

[26] J. Emile-Geay, *Data analysis in the earth & environmental sciences*, Third. FigShare, 2017, p. 265. doi: 10.6084/m9.figshare.1014336.

[27] J. Emile-Geay, "Time Series Analysis in Python - Practicum." [Online]. Available: https://github.com/LinkedEarth/PyRATES_practicums_py

[28] D. Khider *et al.*, "Pyleoclim: Paleoclimate Timeseries Analysis and Visualization With Python," *Paleoceanography and Paleoclimatology*, vol. 37, no. 10, 2022, doi: 10.1029/2022pa004509.

[29] D. Khider and Y. Gil, "The Scientific Paper of the Future v2024 Training Materials," 2024, doi: 10.6084/m9.figshare.25754244.v1.

[30] M. D. Wilkinson *et al.*, "The FAIR Guiding Principles for scientific data management and stewardship," *Scientific Data*, vol. 3, no. 1, 2016, doi: 10.1038/sdata.2016.18.

[31] A.-L. Lamprecht *et al.*, "Towards FAIR principles for research software," *Data Science*, vol. 3, no. 1, pp. 37–59, 2019, doi: 10.3233/ds-190026.

[32] C. Goble *et al.*, "FAIR Computational Workflows," *Data Intelligence*, vol. 2, no. 1–2, pp. 108–121, 2020, doi: 10.1162/dint_a_00033.

[33] G. Brandl, "Sphinx documentation." [Online]. Available: https://www.sphinx-doc.org/en/master/

[34] D. Khider, "PyCatSim: A fun, modular, object-oriented Python package for simulating cat behavior." [Online]. Available: https://github.com/khider/pyCatSim

Open Access | https://doi.org/10.25080/fncj2446

SciPy 2025

July 7 - July 13, 2025

Proceedings of the 24th Python in Science Conference
ISSN: 2575-9752

Numba v2: Towards a SuperOptimizing Python Compiler

Siu Kwan Lam[1], **Stuart Archibald**[1], and **Stan Seibert**[1]

[1]Anaconda, Inc.

Abstract

This paper presents early work on Numba v2, a general-purpose compiler for Python that integrates equality saturation (EqSat) as a foundation for program analysis and transformation. Whilst inspired in part by the needs of AI/ML workloads, and also supporting tensor optimizations, Numba v2 is not a tensor-oriented compiler. Instead, it provides a flexible framework where user-defined mathematical and domain-specific rewrites participate in the compilation process as a complement to the compilation supported by Numba today. This paper outlines the design of Numba v2's EqSat-based compiler and its potential to serve as an extensible, superoptimizing compiler for a broad range of numerical and scientific Python applications.

Keywords python, compiler, equality saturation, e-graph, MLIR

1. Introduction

The Numba team is developing a new version of the Numba compiler (Numba v2)[1] to meet the evolving computational demands of the modern workloads, including, but not limited to, those in AI/ML. Whilst these domains have popularized high-level optimization techniques based on tensor-oriented programming, Numba v2 is not a specialized tensor compiler; instead, it is a general-purpose compiler that supports tensors. It integrates equality saturation (EqSat) as a core mechanism for program analysis and transformation. The use of EqSat allows the compiler to blend user-defined mathematical rewriting and other domain specific rewriting into the transformation pipeline for compiling numerically oriented Python code with loop and branch constructs. Although Numba v2 is still under active development, this paper presents its new EqSat-based compilation strategy as a foundation for an extensible superoptimizing[2] compiler.

1.1. The importance of mathematical rewrites

AI/ML has popularized tensor-oriented programming. In contrast to array-oriented programming, where an array is a data-structure without mathematical meaning, tensor-orientated programming has an associated mathematical nature. This move to such higher level abstractions allows the compiler to manage low-level details such as buffer reuse, DMA transfers[3] to accelerators, and even asynchronous transfers in distributed systems. It also enables the compiler to do analysis and perform optimizations in terms of the mathematical properties of tensors.

Published Jul 10, 2025

Correspondence to
Siu Kwan Lam
siu@anaconda.com

Open Access

Copyright © 2025 Lam *et al.*. This is an open-access article distributed under the terms of the Creative Commons Attribution 4.0 International license, which enables reusers to distribute, remix, adapt, and build upon the material in any medium or format, so long as attribution is given to the creator.

[1]Early development work: Numba v2 Compiler Design.
[2]A superoptimizer exhaustively searches for the optimal program [1].
[3]A direct-memory-access (DMA) transfer refers to hardware peripherals handling read from and write to memory without involving the processor. These transfers occur asynchronously with respect to the processor.

One of the challenges with the currently hugely popular large-language models (LLMs) is that Transformers are computationally expensive. [2] manually optimized the *Attention* layer in *Transformer*s to improve performance, building on FlashAttention's [3] fusing of the softmax operation into the matrix-multiply operation. A reviewer of the paper asked why compilers couldn't perform the necessary fusion, to which the author answered:

> Compilers can generally perform fusion. However, optimizations that require mathematical rewriting of the same expression (while maintaining numerical stability) are generally harder for compilers.
>
> —- T. Dao [2] answering why compiler cannot create FlashAttention-2

The gap between what practitioners need and what compilers can provide is fulfilled by mathematical rewrites, expressed strategically using domain knowledge to allow for more aggressive optimization. In this paper, the use of EqSat is proposed to address this issue. EqSat allows mathematical rewrites and other code transformations to be expressed in declarative logical rules. The compiler can then explore all program variants derived from these rules, hence *saturation*. Finally, the optimal program variant is obtained via a cost-based extraction. This approach leads to a superoptimizing compiler.

1.2. Phase ordering problems in Numba v1

Numba v1 suffers from the phase-ordering problem—a well-known challenge in compiler design where the order of compiler passes can significantly impact the result. EqSat offers a principled solution by exploring all rewrite possibilities simultaneously without a specific order.

Numba v1 works by translating the Python bytecode for a function into an intermediate representation (IR) called Numba IR. Type inference is run on this to statically type the variables, it is then lowered to LLVM IR, which is then JIT compiled using the LLVM compiler infrastructure. For details see S. K. Lam, A. Pitrou, and S. Seibert [4].

The Python bytecode is an IR executed by the Python virtual machine, it is necessarily low level, so much so that the majority of the semantics expressed in source code, for example array expressions, are entirely lost once encoded into bytecode and must be rediscovered later in the compiler pipeline. Numba's IR is a near direct translation of Python bytecode and therefore also lacks high-level semantic information from the original source. As a result, recovering the programmer's intent requires significant analysis of the Numba IR, including control-flow reconstruction and type inference. This problem manifests itself acutely when, for example, trying to identify regions of code which are suitable for parallel execution or when trying to prove amenability for array expression fusion.

Following bytecode to Numba IR translation, the next step in compilation is type inference. This is the process of determining the type of all the variables in use. It is quite hard to do this using fixed-point iterative algorithms, particularly when protocol based code is being compiled (code using "dunder" methods). This is because types have to be propagated through the system, and then branches pruned and inlining performed based on the arrival of concrete type information, which then in turn may expose further areas of code that subsequently need further rounds of type propagation, branch pruning and inlining. The interdependence creates a cascading effect that is difficult to manage in a naive pass-based fixed-point iteration scheme. Essentially this is a compiler pass phase ordering problem with no guarantee that the type propagation will stabilize. A further side effect of this sort of problem, and incremental type inference in general, is that back-tracking invalid paths is computationally wasteful and is hard to debug. It also makes it hard to e.g. pass an untyped

list into a function and to permit compilation to continue with the view that the type may be resolved eventually.

1.3. A motivating case

To demonstrate some of these issues in relation to compiler flexibility and optimisation, a motivating example of the associativity of matrix multiplication can be used. In this example, six matrices are being multiplied together:

$$A1 \times A2 \times A3 \times A4 \times A5 \times A6 \tag{1}$$

Using NumPy syntax, it looks like this:

```
def six_matrix_multiply(A1, A2, A3, A4, A5, A6):
return A1 @ A2 @ A3 @ A4 @ A5 @ A6
```

From Numba v1's perspective, it will analyse the bytecode and create some Numba IR which will naively store the result from each operation in the chain into a temporary variable for use in the next operation. Type inference will then run and work out that each `@` is actually a `numpy.dot` operation and will resolve that as a function that will take two array inputs and return an array output. Numba will then lower this typed Numba IR to a chain of (wrapped) BLAS `GEMM` calls, with Numba's built-in runtime providing the memory buffers for the array temporaries.

Whilst Numba v1 did resolve the types correctly within its type system, because its type system is hard to extend, it cannot be made to go further and do the necessary shape inference needed to be able to optimise the sequence of matrix multiplications. Arguably Numba v1 could have a custom optimisation pass added to its pipeline that operates on the low level Numba IR and rewrites it to try and reuse memory buffers associated with temporaries and perhaps looks at the matrix memory ordering to try and improve cache utilization. However, this would be a very custom compiler pass, tuned to this one situation, and almost certainly quite hard to write.

Now imagine applying a Numba v2 that doesn't have the limitations of v1 to the same problem. First a higher-level IR is emitted than that comes from the bytecode. Subsequently, both type and shape inference are performed to identify the exact shapes of all the arrays in the computation. This now leads to an interesting problem, in that the shapes and operations are all known, but optimising this is incredibly hard in part because the problem space is so large. In fact, the number of possible grouping under the associativity rule is the Catalan number $C_5 = \frac{(2\times5)!}{(5+1)!5!} = 42$.

Consider that matrix multiplication is a well defined problem and that the approximate floating point cost of for multiplying matrices $P_{m\times k} \times Q_{k\times n}$ is $2mnk$. Furthermore, the associativity rule for multiplication is well known and easily expressed. There is now a problem space forming where the operations are known, the types and shapes of the operands are known, rules of associativity could be applied, and a cost is available for all operations. An optimal solution exists, but as noted before, the space is large. To solve this problem a new technique is employed, that of superoptimization via equality saturation. This technique uses a set of rewrite rules and effectively builds all possible programs simultaneously. It then assigns costs to each expression and then uses a cost model to extract the optimal solution.

2. Developing an Equality Saturation based solution

The ideas behind equality saturation lie in the e-graph data structure, which in turn originates from automatic theorem provers [5] and SAT solvers [6], where it was used to

efficiently represent congruence closure. More recently, this theory has been adapted into the powerful equality saturation (EqSat) technique, which has enabled term-rewriting-based compiler optimizations (M. Willsey, Chandrakana Nandi, Yisu Remy Wang, O. Flatt, P. Panchekha, and Z. Tatlock [7], Y. Zhang *et al.* [8]).

Traditional term rewriting systems rely on fixed rule application schedules, discarding the original term after each rewrite. As a result, their effectiveness depends heavily on carefully designed transformation pass orderings, which are often brittle and unable to guarantee optimal results across all cases. Some systems mitigate this by using backtracking [9], where previous terms are restorable, but this approach is computationally expensive, as all alternatives must be evaluated sequentially. EqSat addresses this by applying all rewrite rules simultaneously and retaining all resulting variants. It efficiently stores these alternatives by grouping semantically equivalent terms into equivalence classes (e-classes), with each e-class containing multiple e-nodes each representing equivalent program terms. This structure enables substantial reductions in both search time and memory usage, allowing the compiler to explore the full space of program variants without requiring explicit rule application schedules. Consequently, EqSat eliminates phase-ordering problems and promotes a more composable and extensible compiler architecture.

EqSat and the ability to easily employ rewriting rules exposes a realm of optimisation capabilities that were not previously available. For example, if it is known that the application domain is deep-learning, then floating point error tolerance is likely quite high and various rewrite rules to exploit this situation could be applied. First, it is possible to use lower-precision data types such as 8-bit floats or even 4-bit floats. If the application needs to be deployed on an industrial scale that serves thousands of requests per second, or at the edge running on power-limited mobile phone processors, the power usage becomes an important factor as well. Use of integer addition to approximate the floating point multiplication for lower power scenarios is another option. The computation could be done as an approximation, maybe as a low-rank approximation via SVD compression. All of these possibilities are encodable as rewrite rules and become part of the possible solution space within the equality saturation based optimiser. Extracting the optimal solution just becomes a question of defining a cost model that is appropriate to the domain e.g. performance, power cost, code size, hardware.

3. Compiler Design

3.1. Architecture Overview

Numba v2 architecture follows the conventional structure of many compilers; a frontend for canonicalising and translating an input language into an intermediate form, a middle end for performing optimisations and a backend for translating the intermediate form into an executable. The following sections explain the process in more detail and should be read in conjunction with the following diagram.

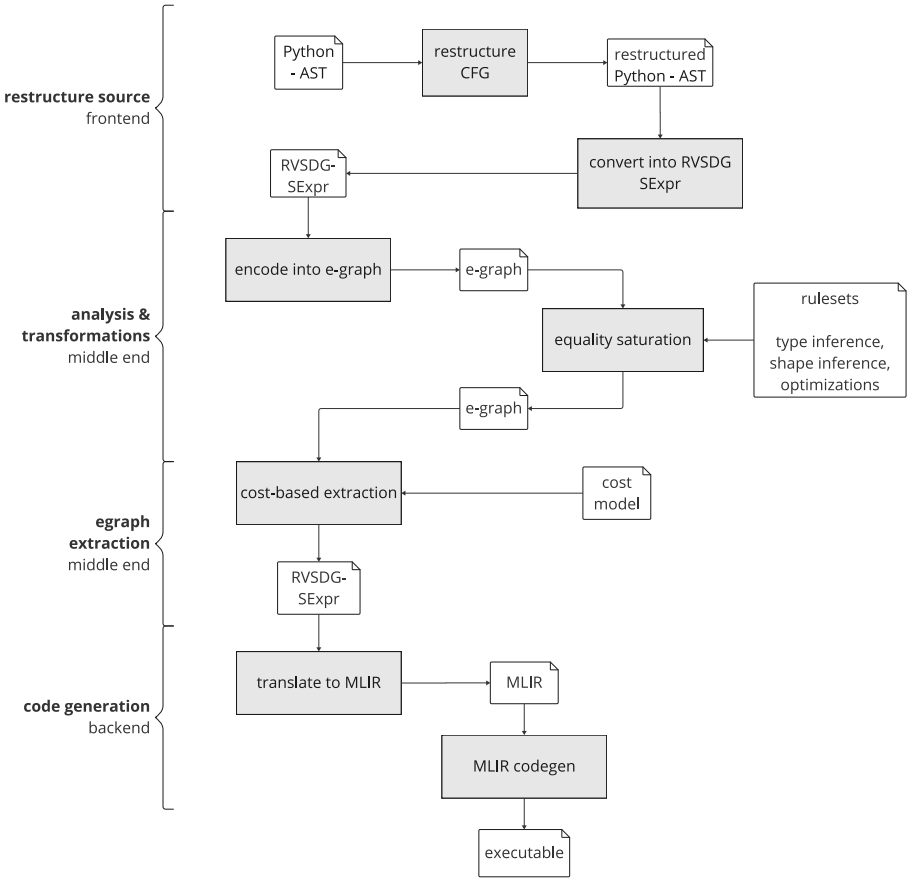


Figure 1. *Compiler architecture*

A key architectural distinction between Numba v1 and v2 lies in their treatment of middle-end optimization. Numba v1 has a minimal middle-end that handles type resolution and basic IR transformations before lowering to LLVM IR. This design places the optimization burden on LLVM's backend passes. This architectural shift enables optimizations that are fundamentally inaccessible to v1's pipeline, as demonstrated by the 19× speedup achieved in Section 14 through purely algebraic reorganization.

3.2. Converting imperative Python to functional form

As noted above, to use egglog effectively, the Python AST is first translated into an S-expression form that supports term rewriting. A direct encoding of the AST into S-expression form is insufficient, as it captures only the syntactic structure of the program. The goal, however, is to perform term rewriting based on semantic equivalence and so the syntactic form is translated into a semantic representation. An additional challenge is that Python is an imperative language with implicit state mutations. To accommodate the necessary semantics, the AST is converted into a *regionalized value-state dependence graph* (RVSDG).

RVSDG is a data-oriented program representation that can simplify compiler analysis and transformation. Unlike the common SSA-CFG IRs[4], RVSDG does not explicitly encode the control-flow graph (CFG). Prior studies (N. Reissmann, J. C. Meyer, H. Bahmann, and M.

[4]SSA-CFG IR refers to an Intermediate Representation that combines Static Single Assignment (SSA) form with an explicit Control Flow Graph (CFG) structure. It is used in LLVM.

Själander [10], D. Weise, R. F. Crew, M. Ernst, and B. Steensgaard [11], A. C. Lawrence [12]) have shown that RVSDG enables faster and simpler implementations of classical compiler passes. This reflects a broader trend towards data-dependency based compilation, even for imperative languages, as evidenced by frameworks like MLIR [13].

In RVSDG, programs are represented in terms of dataflow graphs, where edges indicate value and state dependencies, and nodes denote computation. Structured control flow is modelled using regions that encapsulate other nodes and edges, resembling block diagrams in digital circuit design. Within this structure, parallelization opportunities appear as independent groups of nodes.

There are three types of structured control flow regions: linear dependency, switches for optional paths, and tail loops. The resulting graph is an acyclic graph with nested regions, making it amenable to hierarchical analysis. Every node, including control-flow regions, in an RVSDG is an operation with incoming and outgoing "ports" for values and states.

Following the algorithms outlined in H. Bahmann, N. Reissmann, M. Jahre, and J. C. Meyer [14], the Python AST is first converted into a structurized control-flow graph[5], each imperative Python expression is then converted into an RVSDG-conforming S-expression by explicitly passing an `io` state through each operation. Any operation with side-effects receives and returns an `io` state. Initially, all Python operations are conservatively assumed to have side-effects. This assumption can later be relaxed through analysis such as type-inference, which is done using the same EqSat rule mechanism as other code transformations.

This approach differs from prior work on Program Expression Graphs (PEGs) [15], which handle effects through a single global state token. RVSDG simplifies effect handling by using explicit state edges within hierarchical regions that naturally enforce sequential ordering through structured control flow representation.

3.3. Controlflow handling and semantic raising

Encoding Python into RVSDG is conceptually similar to translating Python into a pure functional language. In this representation, all operations appear pure because state is passed as a first-class value. An effect-free operation is simply one that doesn't modify the `io` state. Many traditional compiler passes are simplified in RVSDG because of the uniform treatment of states and values. Since there is no control-flow graph, as the program is viewed in a data-oriented perspective, control-flow constructs, such as conditionals and loops, are treated like any other operations. This eliminates the need for specialized logic to handle control-flow and reduces the complexity of compiler passes.

Practically, the compiler uses an S-Expression based IR as an RVSDG based functional encoding suitable for translating into the egglog Python library bindings. At this point, the Python program is in a form in which rewrite rules can be written and subsequently applied through EqSat.

E-graph rewrite rules are akin to logic programming. They operate declaratively without specifying application strategy and are therefore inherently composable. The rewrite system uses pattern matching to identify transformable program structures and conditional guards to ensure transformation validity. Each rule specifies a source pattern to match against the e-graph and a target pattern representing the equivalent transformation, with optional conditions that must hold for the rewrite to apply. For example, the constant folding ruleset below demonstrates how conditional branches with constant predicates can be simplified:

[5]For details about this process, see the EuroScipy 2024 talk titled "Reguarlizing Python using Structured Control Flow" by Valentin Haenel

```python
@ruleset
def ruleset_const_fold_if_else(a: Term, b: Term, c: Term, operands: TermList):
    yield rewrite(
        # Define the if-else pattern to match
        Term.IfElse(cond=a, then=b, orelse=c, operands=operands),
        subsume=True,  # subsume to disable extracting the original term
    ).to(
        # Define the target expression
        # This applies region `b` (then) using the `operands`.
        Term.Apply(b, operands),
        # Given that the condition is constant True
        IsConstantTrue(a),
    )
    yield rewrite(
        # Define the if-else pattern to match
        Term.IfElse(cond=a, then=b, orelse=c, operands=operands),
        subsume=True,  # subsume to disable extracting the original term
    ).to(
        # Define the target expression.
        # This applies region `c` (orelse) using the `operands`.
        Term.Apply(c, operands),
        # Given that the condition is constant False
        IsConstantFalse(a),
    )
```

The `Term.IfElse` node represents a conditional expression in RVSDG. Simplifying it is straightforward when the condition is a constant. A pattern matching rewrite rule replaces the entire *if-else* construct with the corresponding selected branch. The logic is intuitive and compact when expressed in the declarative rule system of *egglog*. The simplicity of this transformation highlights the benefits of treating control flow as just another term in the rewrite system.

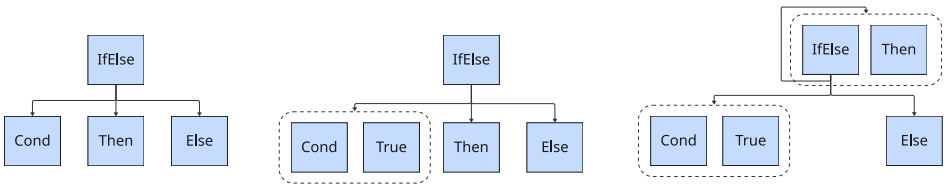


Figure 2. *This illustrates the process of applying the if-else simplification rewrite rule on the e-graph. Each blue box represents an e-node. The dotted box represents e-classes—blue-boxes inside are equivalent to each other. The left diagram is the initial state. The center diagram shows the condition (Cond) is recognized as a constant True. The right diagram shows the establishing of equivalence between the IfElse term and the Then term.*

Loops presents a greater challenge as RVSDG supports only tail-controlled loops, while Python uses constructs like while-loop and for-loop that do not naturally fit this mode. To bridge this gap to allow for more intuitive reasoning in EqSat, *semantic raising* is employed to recover the high-level Python loop constructs by matching their low-level RVSDG patterns. This allows full recovery of Python loop semantics from raw RVSDG form, enabling use of more intuitive rewrite rules.

As an example, RVSDG restructurization converts a for-loop into the following CFG structure.

```
condition = compute_init_loop_condition()
do {
    if condition {
        condition, vals = loop_body(*vals)
    }
} while condition
```

Through semantic raising, the loop structure above can be "raised into a more abstract representation:

```
Py_ForLoop(iterator, operands) { loop_body }
```

To recognize the pattern as a `for ... in range` loop, which is common in numerical code, further semantic raising can be employed. This high-level representation enables a more direct mapping to MLIR's *affine* dialect.

3.4. Limitations of EqSat

One known challenge of EqSat is that saturation can create very large e-graphs that exceed the memory limits of a given machine. In the case of Numba v2, this issue has not arisen as the project is still in its early stages. However, prior work provides insight into techniques for managing e-graph growth.

S. Thomas and J. Bornholt [16] reintroduce phases into the rewriting process to control the e-graph expansion. They break rewriting into three phases:

- Expansion phase: explores alternative unoptimized program variants; e.g. applying math identities.
- Compilation phase: converts unoptimized terms to optimized ones.
- Optimization phase: searches for better representation among optimized terms.

To prevent unbounded graph growth, they also employ cost-based pruning, using an abstract cost model to select the best intermediate program after expansion and compilation phase. Saturation in these phases is bounded by a time limit and only once improvements plateau, according to the abstract cost model, the optimization phase is applied.

T. Koehler, P. Trinder, and M. Steuwer [17] offer a different approach. Instead of optimizing directly over full program representations, they use *program sketches*—compressed abstractions of the full program—to guide optimization. This helps reduce e-graph size and keeps optimization tractable.

These concerns are particularly relevant for compilers that target low-level representations, such as ISA or machine code, where program representations can be enormous In contrast, Numba v2 applies EqSat at a much higher level. Low-level optimizations are delegated to MLIR. High-level program representations tend to be significantly smaller, and MLIR supports expressive dialects that further reduce complexity. For example, recognizing and translating a `for … in range` loop into MLIR's *affine* dialect can eliminate many low-level details and shrink the overall program graph.

3.5. Cost-based extraction

The cost-based extraction in Numba v2 is built around three guiding principles:

- **Intuitive:** it should be intuitive for users to express cost equations naturally for each operation in the cost model.

- **Stability.** Cost-computation must be stable such that the computed cost of any subprogram remains consistent across different compilations so as to ensure reproducibility.
- **Fast.** Practicality and performance are prioritized even at the expense of finding the absolute optimal solution, especially given the scale of real-world programs.

In the generalized case, costs are modelled as vectors of floats, allowing for future experimentation with multi-objective evaluation. Finding the exact minimal cost extraction from an e-graph is NP-hard [18]. An *iterative relaxation* algorithm is used to manage this complexity and to efficiently handle cycles in the e-graph, so as to permit convergence to a near-optimal solution.

3.6. Iterative relaxation algorithm

The extraction algorithm is inspired by the Bellman-Ford algorithm [19], combining dynamic programming with iterative refinement. It decomposes the extraction problem into subproblems, computing an initial best-effort solution that is progressively improved upon as more information becomes available in subsequent iterations. Unlike Dijkstra-based approaches commonly used in other e-graph systems[6], Bellman-Ford permits negative costs, though these are not currently required in the Numba v2 test cases. The iterative nature also offers a practical advantage by allowing early termination based on a time budget.

The algorithm has the following steps:

1. **Dependency graph construction:** Starting from the root e-class (the function definition term), the algorithm traces its member e-nodes. Each e-node references child e-classes, forming a dependency graph. In this graph, e-classes point to e-nodes, and e-nodes point to their child e-classes.
2. **Cost function retrieval:** For each e-node, the algorithm retrieves the associated cost function used to compute its contribution to the total cost.
3. **Identify constant values:** Constant values that participate in cost calculations are identified, such as the shape of a tensor in a matrix multiplication operation.
4. **Compute the BFS layers:** Since the dependency graph may contain cycles, a true topological sort is not possible. Instead, breadth-first search (BFS) layers are used to approximate topological ordering.
5. **Iterate until convergence:** The cost is computed bottom-up for each e-node by using the cost of its child e-classes in the e-node's cost function. If the child e-class has not yet been evaluated, its cost is temporarily treated as infinite. Each e-class aggregates the costs of its member e-nodes and selects the best. The algorithm continues until convergence. The criteria for convergence are:

    - The number of finite-cost e-nodes remains unchanged.
    - The maximum cost change falls below a predefined threshold

By construction, e-nodes form a directed acyclic graph (DAG) since they represent RVSDG terms and only reference the operands (child nodes). However, cycles can arise through e-node merging during rewriting. For example, the identity `pow(x, 1) = x` causes both expressions to reside in the same e-class, creating a cyclic self reference. These cycles are not problematic and they are resolved early in the relaxation process. Once a lower cost is found for a node, it is immediately selected as the currently optimal choice, effectively breaking the cycle.

[6] eggcc and faster_greedy_dag

3.7. Dual cost model

Accurate cost-based extraction of the e-graph requires careful handling of differing cost aggregation requirements for expression DAGs and control-flow DAGs.

In the e-graph, expressions are represented as DAGs where nodes may be shared in other expressions. Since the cost is computed locally for each operation without full visibility to the entire graph structure due to the bottom up traversal, common subexpression can be counted more than once in an expression tree. For example, `x * x * x` can redundantly count the cost of `x` multiple times (See Figure 3) , making it appear more expensive than `pow(x, 3)`. To avoid this, node-based cost accounting is performed to properly recognize shared expression nodes.

Control-flow operations such as loops require a different treatment. A loop's cost should reflect its trip count multiplied by the cost of its body. Unlike shared subexpressions in expression DAGs, the cost of the loop-body is not discounted, even if it appears in other parts of the program. This calls for edge-based cost accounting. For example, in Figure 4, the cost of `Body` should contribute to the cost of both `Loop1` and `Loop2`.

A dual cost model is adopted to resolve the conflicting situation:

- **Local cost computation.** Each e-node defines a cost equation that receives the costs of its child nodes (operands). For example, a loop operation might compute its cost as the cost of the loop body multiplied by the trip out. For simple operations like scalar addition, the cost is simply a constant. These local cost functions can be user-defined and customizable.
- **Child-dependent cost computation.** To handle shared subexpression nodes, an external cost that aggregates the costs of a node's children without duplication is computed. This requires tracking the optimal e-node for each e-class along with its total cost (local + external). During extraction, the full set of contributing e-nodes is computed to ensure that each e-node is only counted once. This approach is similar to the *FasterGreedyDagExtractor*.

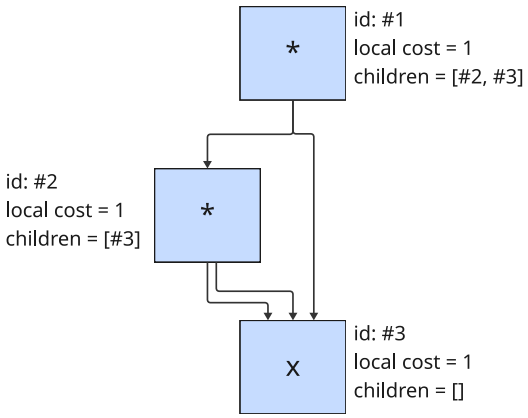


Figure 3. *This illustrates the expression x * x * x. The cost of node #1 is the local_cost#1 + local_cost#2 + local_cost#3.*

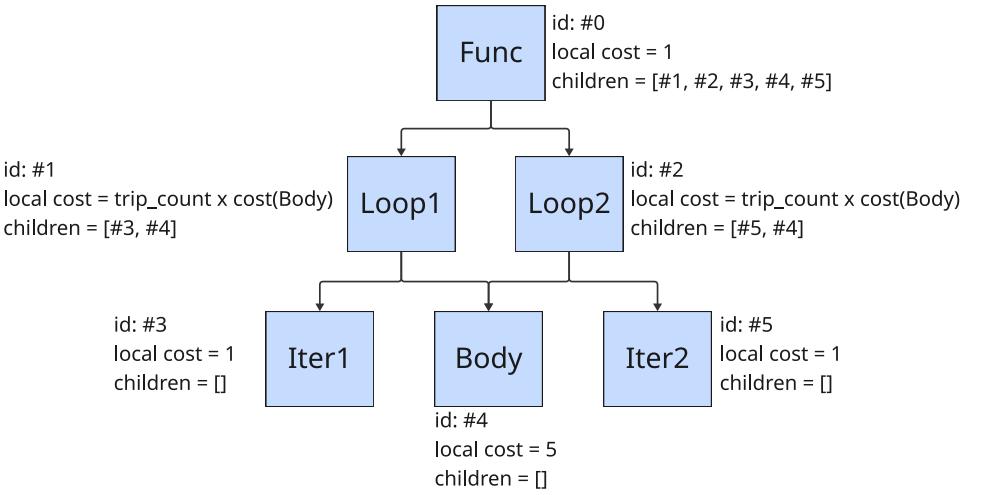


Figure 4. *This illustrates a function with two loops, for which the loop-bodies are equivalent in the e-graph. The function (#0) has a cost that is the sum of all local costs of its unique children. In Loop1 (#1) and Loop2 (#2) the cost of the Body is contributing to their local costs, respectively.*

3.8. Evaluating Scalability

A scalability analysis was conducted by scaling the matrix multiplication associativity example from Section 14 to an increasing number of operations. Here, k represents the number of matrices, and $k-1$ corresponds to the number of matrix multiplications. This subjects the compiler to a pathological scenario due to the combinatorial explosion caused by the associativity rule. Figure 5 shows that the e-graph exhibits polynomial scaling behavior. The number of e-classes shows sub-quadratic growth ($k^{1.75}$) while the number of e-nodes exhibits sub-cubic growth ($k^{2.44}$). This growth rate is substantially lower than the theoretical exponential rate because e-graphs inherently perform structural matching that compresses the representation.

Both saturation and extraction phases exhibit sub-quadratic scaling ($n^{1.8}$) w.r.t. number of e-classses. In practice, such pathological cases of long chains of associative operations are uncommon, and since associativity rules serve only optimization purposes, compilation time can be capped as needed. The sub-quadratic growth ensures EqSat remains tractable for realistic numerical programs.

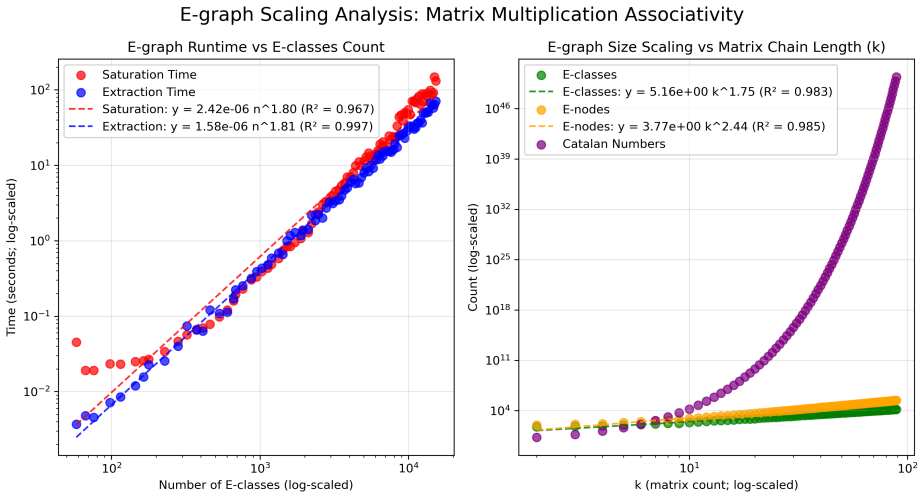


Figure 5. *(a) Left: saturation and extraction time scaling w.r.t. e-graph size. (b) Right: e-graph size scaling w.r.t. matrix-multiplication chain length.*

3.9. Further considerations

This work represents an early stage in the substantial development work of Numba v2. While initial results are promising, future efforts will need to scale to more realistic examples, focus on refining the techniques, and conduct deeper analysis.

Larger and whole-program examples

To better evaluate the effectiveness and scalability of this technique, testing on more complex and realistic programs is needed. As part of this effort, the next goal is to compile a NumPy-based implementation of a small LLaMa based LLM [20].

Extending to multi-dimensional cost models. The cost-based extraction algorithm is designed to support n-dimensional cost vectors, enabling multi-objective optimization (e.g. runtime, memory usage, floating-point accuracy, power). So far, the examples have primarily used 1-dimensional cost, but future work will explore richer cost models.

4. Case Studies

4.1. GELU Approximation

Traditional compilers have limited ability to relax floating-point precision. Many expose a fast-math flag that enables basic operation reassociation or using fused multiply-add instruction, but they typically lack more advanced capabilities such as replacing transcendental functions with cheaper approximations.

In domains like deep learning, it's often possible to speed up programs by reducing the precision of specific functions. Activation functions are a good candidate for this kind of optimization.

As a concrete example, GELU (Gaussian Error Linear Unit), which is a common in some transformer architectures [21], is used.

GELU equation:

$$\mathrm{GELU}(x) \approx 0.5x\left(1+\tanh\left(\sqrt{\frac{2}{\pi}}(x+0.044715x^3)\right)\right) \tag{2}$$

The Python implementation:

```
def gelu_tanh_forward(x):
    dt = np.float32
    erf = np.tanh(np.sqrt(dt(2) / dt(np.pi)) * (x + dt(0.044715) * x**3))
    return dt(0.5) * x * (dt(1) + erf)
```

Pade44 approximation is used to simplify the tanh(x) function:

$$tanh(x) \approx \frac{10x^3 + 105x}{x^4 + 45x^2 + 105} \tag{3}$$

In egglog, the rewrite rule for it is expressed as below:

```
rewrite(Npy_tanh_float32(x)).to(
    div(
        add(mul(flt(10), pow(x, liti64(3))), mul(flt(105), x)),
        add(
            add(pow(x, liti64(4)), mul(flt(45), pow(x, liti64(2)))),
            flt(105),
        ),
    )
)
```

The corresponding formal inference rule representation of the rewrite rule is:

$$\frac{\text{expr} = \text{Npy_tanh_float32}(x)}{\text{expr} \to \frac{10x^3+105x}{x^4+45x^2+105}} \tag{4}$$

This can combine with a simplification of the pow(x, constant) term with the identity:

$$x^{ival} = x \cdot x^{\text{ival}-1} \tag{5}$$

As egglog rules:

```
rewrite(pow(x, lit64(ival))).to(
    mul(x, pow(x, lit64(ival - 1))),
    ival >= 1,
)
rewrite(pow(x, lit64(i64(0))), subsume=True).to(
    Npy_float32(Term.LiteralF64(float(1))),
)
```

The corresponding formal inference rule representation of the rewrite rule is:

$$\frac{\text{expr} = \text{pow}(x, \text{ival}), \quad \text{ival} \geq 1}{\text{expr} \to \text{mul}(x, \text{pow}(x, \text{ival} - 1))} \tag{6}$$

$$\frac{\text{expr} = \text{pow}(x, \text{i64}(0))}{\text{expr} \to \text{f32}(1.0)} \tag{7}$$

Egglog saturates these rules, so there is no need to specify an ordering or how many times to apply them. For example, `pow()` calls are automatically expanded into multiplications after the applying the pade44 approximation for `tanh()`.

As a result, user programs can be optimized without requiring direct modifications. Each rewrite rule is also reusable across different programs. Performance evaluation demonstrates 9× speedup for superoptimized code using MLIR code generation compared to Numba v1, which fails to achieve speedup over vanilla Python due to computation dominated by transcendental functions.

Two ways are envisioned for users to take advantage of this kind of program rewriting.

The first is by manual selection from a catalog of rewrite rules. Users will need domain knowledge to choose rules that are appropriate for their use case.

The second is to use a cost model to guide the selection of approximation functions. The cost model can be extended to account not only for runtime performance but also for floating-point accuracy.

4.2. Associativity Rule of Matrix Multiply

In the introduction, there was discussion on how the associativity of matrix multiplication can be exploited to choose a more efficient grouping for performance. Here, the process is demonstrated using Numba v2.

In egglog, the associativity rule can be spelled as:

```
rewrite(MatMul(MatMul(ary0, ary1), ary2)).to(
    MatMul(ary0, MatMul(ary1, ary2))
)
```

The corresponding formal inference rule representation of the rewrite rule is:

$$\frac{\text{expr} = \text{MatMul}(\text{MatMul}(\text{ary0}, \text{ary1}), \text{ary2})}{\text{expr} \rightarrow \text{MatMul}(\text{ary0}, \text{MatMul}(\text{ary1}, \text{ary2}))} \tag{8}$$

Saturation recursively applies the associativity rule to a succession of matrix multiplication and generate all possible groupings.

Shape and type inference rules for the matrix multiplication is expressed as follows:

```
rule(
    # array desc propagation
    ary2 == MatMul(ary0, ary1),
    ad0.toType() == TypeVar(ary0).getType(),
    ad1.toType() == TypeVar(ary1).getType(),
    ad0.ndim == i64(2),
    ad1.ndim == i64(2),
    m := ad0.dim(0),
    n := ad1.dim(1),
    ad0.dim(1) == ad1.dim(0),
    ad0.dtype == ad1.dtype,
).then(
    # set output dims
    ad2 := ArrayDescOp(ary2),
    set_(TypeVar(ary2).getType()).to(ad2.toType()),
    set_(ad2.ndim).to(2),
    set_(ad2.dim(0)).to(m),
    set_(ad2.dim(1)).to(n),
    set_(ad2.dtype).to(ad0.dtype),
)
```

The corresponding formal inference rule representation of the rewrite rule is:

$$\frac{\begin{array}{c}\text{ary2} = \text{MatMul}(\text{ary0}, \text{ary1}),\\ \text{ary0. ndim} = 2, \quad \text{ary1. ndim} = 2,\\ m = \text{ary0. shape}[\,0\,], \quad n = \text{ary1. shape}[\,1\,],\\ \text{ary0. shape}[\,1\,] = \text{ary1. shape}[\,0\,],\\ \text{ary0. dtype} = \text{ary1. dtype}\end{array}}{\begin{array}{c}\text{ary2. ndim} \rightarrow 2,\\ \text{ary2. shape}[\,0\,] \rightarrow m,\\ \text{ary2. shape}[\,1\,] \rightarrow n,\\ \text{ary2. dtype} \rightarrow \text{ary0. dtype}\end{array}} \tag{9}$$

Finally, to recognize that a matrix-multiplication has known shape, the followings rule is used:

```
rule(
    ary2 == MatMul(ary0, ary1),
    ad0.toType() == TypeVar(ary0).getType(),
    ad1.toType() == TypeVar(ary1).getType(),
    ad0.ndim == i64(2),
    ad1.ndim == i64(2),
    Dim.fixed(shapeM) == ad0.dim(0),
    Dim.fixed(shapeN) == ad0.dim(1),
    Dim.fixed(shapeN) == ad1.dim(0),
    Dim.fixed(shapeK) == ad1.dim(1),
    ad0.dtype == ad1.dtype,
).then(
    # Convert to MatMul_KnownShape
    union(ary2).with_(
        MatMul_KnownShape(ary0, ary1, shapeM, shapeN, shapeK)
    ),
)
```

The corresponding formal inference rule representation of the rewrite rule is:

$$\frac{\begin{array}{c}\text{ary2} = \text{MatMul}(\text{ary0}, \text{ary1}),\\ \text{ary0.\ ndim} = 2, \quad \text{ary1.\ ndim} = 2,\\ \text{shapeM} = \text{ary0.\ shape}[\,0\,],\\ \text{shapeK} = \text{ary0.\ shape}[\,1\,],\\ \text{shapeK} = \text{ary1.\ shape}[\,0\,],\\ \text{shapeN} = \text{ary1.\ shape}[\,1\,],\\ \text{ary0.\ dtype} = \text{ary1.\ dtype}\end{array}}{\text{ary2} \rightarrow \text{MatMul_KnownShape}(\text{ary0}, \text{ary1}, \text{shapeM}, \text{shapeN}, \text{shapeK})} \tag{10}$$

To guide the extraction, the cost for the matrix-multiplication is expressed as follows:

```
match op, tuple(children):
    case "MatMul_KnownShape", (_lhs, _rhs, m, n, k):
        return self.get_equation(
            lambda lhs, rhs, *_, m, n, k: 2 * m * n * k,
            constants=dict(m=m, n=n, k=k),
        )
```

The `MatMul_KnownShape` operation is assigned the cost of $2mnk$. Using the `constants` keyword, the cost equation can access to the literal values for `m`, `n` and `k`, rather than the cost of that constant terms themselves.

For testing, the Python source generation backend is used along with the cost-based extraction to optimize the following succession of matrix multiplications:

```
def f0(arr0, arr1, arr2, arr3):
    return arr0 @ arr1 @ arr2 @ arr1 @ arr2 @ arr3
```

with inputs like:

```
M = 200
N = 100
K = 200
arr0 = np.random.random((M, N))
arr1 = np.random.random((N, K))
arr2 = np.random.random((K, N))
arr3 = np.random.random((N, 1))
```

In researching this technique, prototypes for Numba v2 were developed. When applying an equality saturation base optimiser (along with appropriate associatively rewrite rules and cost models) to the matrix multiplication associativity problem above, it successfully

found a solution that might well have been a "guess" that someone hand optimising the code would have made:

```
def f1(arr0, arr1, arr2, arr3):
    x = arr1 @ arr2
    return arr0 @ (x @ (x @ arr3))
```

This version assumed that the reused of the `arr1 @ arr2` term would reduce the overall computation time. Additionally, the right associativity grouping should take advantage of a very small dimension in `arr3` (`arr3.shape[1]`).

Following this an interesting event occurred, the rewrite rules were improved slightly and as a result a new solution was found that had even better performance. The new version is listed below:

```
def f2(arr0, arr1, arr2, arr3):
    return arr0 @ (arr1 @ (arr2 @ (arr1 @ (arr2 @ arr3))))
```

The newly extracted version does not leverage the duplicated term. Instead it uses the full right associativity grouping.

At this point the evaluation of necessary applied mathematics was undertaken to check the solutions and gain understanding into the performance characteristics. It turned out that the problem space used in the examples depended heavily on the value of `k` (the inner dimension of some of the GEMMs), and the compiler was actually identifying this in practice and selecting different solutions based on this value!

The performance of these three versions are evaluated by running in regular Python interpreter with NumPy. The result is summarized in the following table:

Function	Time	Speedup
f0	956μs	1×
f1	209μs	4×
f2	49μs	19×

The automatically extracted version (`f2`) is 4x faster than the hand-optimized version (`f1`), and 19x faster than the original (`f0`). This demonstrates how even a simple expression can lead to significant performance gains through careful reorganization of computation.

However, performing this kind of optimization manually is not ideal, as selecting the best ordering requires searching through a combinatorial space of possible arrangements. Moreover, in realistic applications with multiple abstraction layers, such expressions typically emerge only after aggressive inlining, which depends on thorough static analysis of the program. Manual optimization undermine the benefits of abstractions, making the program harder to read, maintain, and improve. Numba v2 will enable this kind of superoptimization automatically.

A comprehensive performance evaluation using grid search across different matrix sizes demonstrates significant performance improvements over the original unoptimized code. Figure 6 shows an average speedup of 2.25× achieved purely through algebraic reorganization without machine code generation.

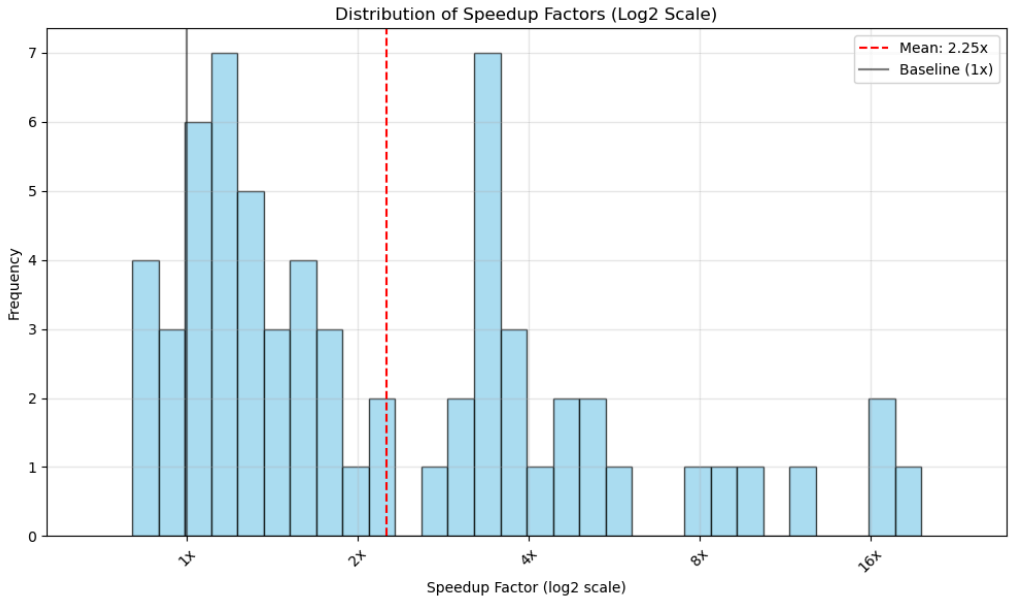


Figure 6. *Performance distribution of superoptimized programs using matrix-multiplication associativity rules across different matrix sizes.*

4.3. High-level Tensor Rewrite

TASO [9] automatically generates graph substitution rules for tensor computations in deep neural networks (DNNs). Once these rules are generated, they are applied to the DNN compute graph using a cost-based backtracking search, serving as a form of superoptimization. TENSAT [22] builds on this approach by replacing the sequential search with EqSat, which reduces search time by 48× and produces better optimization results. In this case study, a graph substitution rule inspired by TASO and TENSAT within the compiler is implemented.

Consider this Python source code:

```
def original_mma(input_1, input_2, input_3, input_4):
    out1 = np.matmul(input_1, input_3)
    out2 = np.matmul(input_2, input_4)
    return out1 + out2
```

A TASO/TENSAT rule suggests that the optimized output is:

```
def optimized_mma(input_1, input_2, input_3, input_4):
    concat_input = np.hstack([input_1, input_2])
    concat_weight = np.vstack([input_3, input_4])
    return np.matmul(concat_input, concat_weight)
```

The corresponding egglog rewrite rule is:

```
rewrite(
    ewadd(matmul(ary1, ary3), matmul(ary2, ary4)),
).to(matmul(hstack(ary1, ary2), vstack(ary3, ary4)))
```

The corresponding formal inference rule representation of the rewrite rule is:

$$\frac{\text{expr} = \text{ewadd}(\text{matmul}(\text{ary1}, \text{ary3}), \text{matmul}(\text{ary2}, \text{ary4}))}{\text{expr} \rightarrow \text{matmul}(\text{hstack}(\text{ary1}, \text{ary2}), \text{vstack}(\text{ary3}, \text{ary4}))} \tag{11}$$

Under the hood, the original funciton is translated as the following RVSDG-IR (some details are truncated for readability):

```
original_mma = Func (Args 'input_1' 'input_2'  'input_3' 'input_4')
$0 = Region <- !io input_1 input_2 input_3 input_4
{
  $1 = PyLoadGlobal $0[0] 'np'
  $2 = PyAttr $0[0] $1 'matmul'
  $3 = PyCall $2[1] $2[0] $0[1], $0[3]
  $4 = PyLoadGlobal $3[0] 'np'
  $5 = PyAttr $3[0] $4 'matmul'
  $6 = PyCall $5[1] $5[0] $0[2], $0[4]
  $7 = DbgValue 'out1' $3[1]
  $8 = DbgValue 'out2' $6[1]
  $9 = PyBinOp + $6[0] $7, $8
} [830] -> !io=$9[0] !ret=$9[1]
```

Internally, the curated rules perform the following steps to enable the short optimization rule above:

- Recognize `np` as a global reference to the `numpy` module.
- Recognize `np.matmul` as the matrix-multiplication function
- Transform calls to `np.matmul` as pure function calls named `matmul`.
- Perform shape and type inference to determine the output array's shape and type.
- Identify binary operation `+` of the results of the matrix multiplications as elementwise addition of matrices. The operation is pure and renamed to `ewadd`.
- Perform broadcasting to determine the output array's shape and type.

The optimized RVSDG-IR is listed below (with truncation for readability):

```
transformed_original_mma = Func (Args 'input_1' 'input_2' 'input_3' 'input_4')
$0 = Region <- !io input_1 input_2 input_3 input_4
{
  $1 = NbOp_Npy_HStack_KnownShape $0[1] $0[2] 102400 256
  $2 = NbOp_Npy_VStack_KnownShape $0[3] $0[4] 256 7680
  $3 = NbOp_MatMul_KnownShape $1 $2 102400 256 7680
} [1079] -> !io=$0[0] !ret=$3
```

Encoding TASO/TENSAT rules into the compiler is straightforward and can be automated. This allows a future where the database of optimization rules can continuously expand by integrating with automatic graph substitution discovery system such as TASO.

Domain experts can translate their optimization techniques into reusable rules, enabling the entire community to benefit from collective knowledge. This approach contrasts with the current workflow, where optimizations are developed in isolation and rarely reused.

Performance evaluation using additional distributive rules was conducted across different matrix sizes on programs of the form:

```
def original_mma(input_1, input_2, input_3, input_4):
   return input_1 @ (input_2 + input_3) @ input_4
```

Figure 7 shows performance variations ranging from 0.25× to 50× speedup with high variance and a mean of 0.97×. Performance degradations occur due to inadequate cost modeling of non-arithmetic operations, particularly matrix concatenation (`hstack`, `vstack`), which demonstrates the importance of sophisticated cost modeling.

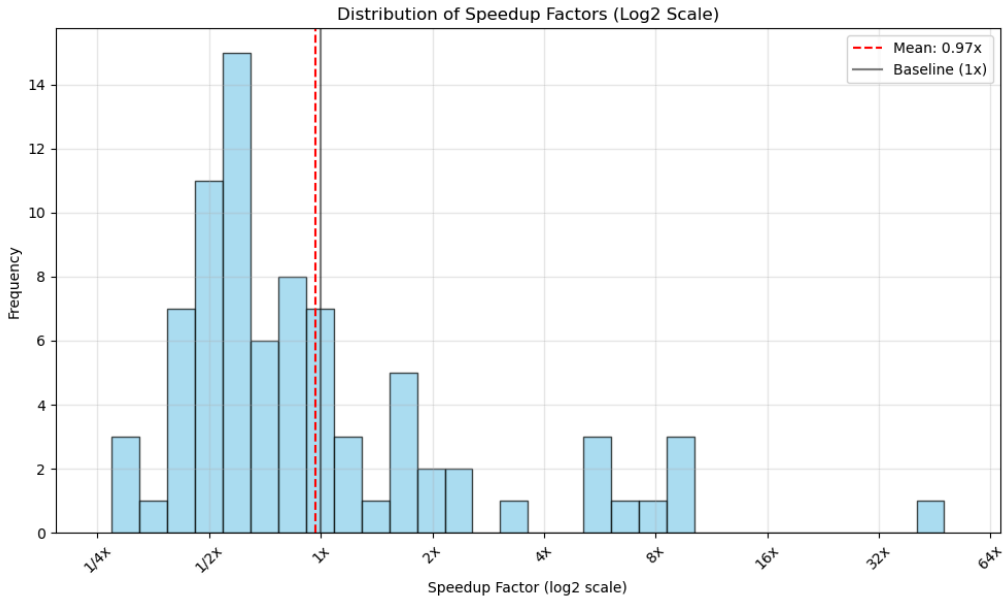


Figure 7. *Performance distribution of superoptimized programs using TASO/TENSAT-inspired tensor rewrite rules across different matrix sizes.*

5. Conclusion

This exploratory work demonstrates the use of equality saturation (EqSat) as a powerful mechanism for the superoptimization of imperative Python code. Python code is first translated into a highly structured IR and then into a functional form as RVSDG, concisely encoding programs, as an e-graph, using a data-oriented representation. This encoding makes it possible to unify general-purpose and tensor-oriented computation. Analysis rules are applied to the e-graph to reconstruct high-level semantics and type information, thus enabling semantics-aware rewrites.

The use of egglog provides an accessible way for users to write domain specific optimizations with a high degree of expressiveness. Program optimization is reframed as a problem of cost-modelling and extraction rather than heuristic-driven pass sequencing. This eliminates the traditional challenges of phase ordering, which have hindered the compilation of highly dynamic Python programs.

Several technical challenges remain—including scalability of large rewrite rule sets, and the design of multi-objective cost-based extraction strategies. Nonetheless, the early results are promising and suggest that this approach provides a powerful and extensible optimisation framework, bringing superoptimization capabilities within reach of domain experts and users alike.

References

[1] H. Massalin, “Superoptimizer: a look at the smallest program,” *ACM SIGARCH Computer Architecture News*, vol. 15, no. 5, pp. 122–126, 1987, doi: 10.1145/36177.36194.

[2] T. Dao, “FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning.” [Online]. Available: https://arxiv.org/abs/2307.08691

[3] T. Dao, D. Y. Fu, S. Ermon, A. Rudra, and C. Ré, “FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness.” [Online]. Available: https://arxiv.org/abs/2205.14135

[4] S. K. Lam, A. Pitrou, and S. Seibert, “Numba: a LLVM-based Python JIT compiler,” in *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, in LLVM '15. Austin, Texas, 2015. doi: 10.1145/2833157.2833162.

[5] C. G. Nelson, “Techniques for Program Verification.” Stanford, CA, USA, 1980.

[6] L. de Moura and N. Bjørner, “Z3: An Efficient SMT Solver,” in *Tools and Algorithms for the Construction and Analysis of Systems*, Springer Berlin Heidelberg, 2008, pp. 337–340. doi: 10.1007/978-3-540-78800-3_24.

[7] M. Willsey, Chandrakana Nandi, Yisu Remy Wang, O. Flatt, P. Panchekha, and Z. Tatlock, “Artifact for "Fast and Extensible Equality Saturation".” [Online]. Available: https://zenodo.org/record/4072013

[8] Y. Zhang *et al.*, “Better Together: Unifying Datalog and Equality Saturation.” [Online]. Available: https://arxiv.org/abs/2304.04332

[9] Z. Jia, O. Padon, J. Thomas, T. Warszawski, M. Zaharia, and A. Aiken, “TASO: optimizing deep learning computation with automatic generation of graph substitutions,” in *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, in SOSP ’19. 2019, pp. 47–62. doi: 10.1145/3341301.3359630.

[10] N. Reissmann, J. C. Meyer, H. Bahmann, and M. Själander, “RVSDG: An Intermediate Representation for Optimizing Compilers,” 2019, doi: 10.48550/ARXIV.1912.05036.

[11] D. Weise, R. F. Crew, M. Ernst, and B. Steensgaard, “Value dependence graphs: representation without taxation,” in *Proceedings of the 21st ACM SIGPLAN-SIGACT symposium on Principles of programming languages - POPL ’94*, in POPL ’94. 1994, pp. 297–310. doi: 10.1145/174675.177907.

[12] A. C. Lawrence, “Optimizing Compilation with the Value State Dependence Graph,” 2007. [Online]. Available: https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-705.pdf

[13] C. Lattner *et al.*, “MLIR: Scaling Compiler Infrastructure for Domain Specific Computation,” in *2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, 2021, pp. 2–14. doi: 10.1109/cgo51591.2021.9370308.

[14] H. Bahmann, N. Reissmann, M. Jahre, and J. C. Meyer, “Perfect Reconstructability of Control Flow from Demand Dependence Graphs,” *ACM Transactions on Architecture and Code Optimization*, vol. 11, no. 4, pp. 1–25, 2015, doi: 10.1145/2693261.

[15] M. B. Stepp, “Equality saturation: engineering challenges and applications.” USA, 2011.

[16] S. Thomas and J. Bornholt, “Automatic Generation of Vectorizing Compilers for Customizable Digital Signal Processors,” in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, in ASPLOS ’24. 2024, pp. 19–34. doi: 10.1145/3617232.3624873.

[17] T. Koehler, P. Trinder, and M. Steuwer, “Sketch-Guided Equality Saturation: Scaling Equality Saturation to Complex Optimizations of Functional Programs.” [Online]. Available: https://arxiv.org/abs/2111.13040

[18] G. Sun, Y. Zhang, and H. Ni, “E-Graphs as Circuits, and Optimal Extraction via Treewidth.” [Online]. Available: https://arxiv.org/abs/2408.17042

[19] R. Bellman, “On a Routing Problem,” *Quarterly of Applied Mathematics*, vol. 16, no. 1, pp. 87–90, 1958.

[20] S. Park, “llama3.np: pure NumPy implementation for Llama 3 model.” [Online]. Available: https://github.com/likejazz/llama3.np

[21] D. Hendrycks and K. Gimpel, “Gaussian Error Linear Units (GELUs).” [Online]. Available: https://arxiv.org/abs/1606.08415

[22] Y. Yang, P. M. Phothilimtha, Y. R. Wang, M. Willsey, S. Roy, and J. Pienaar, “Equality Saturation for Tensor Graph Superoptimization.” [Online]. Available: https://arxiv.org/abs/2101.01332

Open Access | https://doi.org/10.25080/rpaa9684

SciPy 2025

July 7 - July 13, 2025

Proceedings of the 24th Python in Science Conference
ISSN: 2575-9752

Challenges and Implementations for ML Inference in High-energy Physics

Sanjiban Sengupta[1,2], and **Lorenzo Moneta**[1]

[1]CERN, Switzerland, [2]The University of Manchester, United Kingdom

Abstract

At CERN, the European Organization for Nuclear Research, machine learning models are developed and deployed across a wide spectrum of applications, from data analysis and event reconstruction to real-time classification in trigger systems. These systems must operate with extremely high efficiency, as experiments at the Large Hadron Collider (LHC) at CERN generate enormous data streams every second, requiring rapid filtering of irrelevant events to isolate the most promising collisions. With the upcoming High-Luminosity phase of the LHC, collision rates, and therefore data volumes, will increase dramatically, placing even greater demands on the design, optimization, and deployment of machine learning models for fast and reliable inference.

This paper surveys the diverse approaches to machine learning inference currently adopted and under development at CERN to meet these challenges. We highlight methods employed by the four major LHC experiments, as well as complementary tools and frameworks designed for high-throughput environments. Particular focus is given to the unique challenges of running ML models in production at CERN, such as latency constraints, hardware optimization, and integration with large-scale computing infrastructure, and the solutions being pursued to ensure robust performance for present operations and future upgrades of the collider.

Keywords machine learning, optimization, heterogeneous architectures, keras, pytorch, onnx

1. Introduction

Machine learning inference is the process of applying a trained model to new data in order to generate predictions. In high-energy physics (HEP), efficient inference is essential for large-scale production workflows, where billions of collision events must be processed rapidly and accurately. Unlike many other domains, HEP workflows are deeply integrated into C++-based software frameworks, which form the backbone of event reconstruction, simulation, and data analysis. As a result, seamless integration of ML inference into C++ environments is crucial to ensure compatibility, performance, and maintainability within existing large-scale computing infrastructures.

In addition, effective thread management is vital for exploiting ML models in multi-threaded settings, ensuring scalability and optimal performance in massive data-processing pipelines. In many HEP applications, inference must be performed at the event level (for instance, classifying particle collision events in real time at the LHC trigger system), often in single-batch mode, while still meeting strict requirements on both computational speed and memory efficiency. Addressing these challenges is therefore central to enabling fast, reliable, and resource-efficient ML inference within complex scientific workflows.

Published Jul 10, 2025

Correspondence to
Sanjiban Sengupta
sanjiban.sengupta@cern.ch

Open Access

Copyright © 2025 Sengupta & Moneta. This is an open-access article distributed under the terms of the Creative Commons Attribution 4.0 International license, which enables reusers to distribute, remix, adapt, and build upon the material in any medium or format, so long as attribution is given to the creator.

2. Background

2.1. Machine Learning for High-energy Physics

Machine learning has become a central component of HEP research, where experiments generate petabytes of complex and noisy data each year [1]. In this domain, ML provides powerful tools to address the challenges of scale, dimensionality, and heterogeneity inherent to detector data. A key application lies in signal-versus-background discrimination [2], where supervised learning algorithms are trained to identify rare events of interest, such as potential new particles or exotic decay channels against overwhelming amounts of Standard Model background processes. Beyond classification, ML is also used for particle identification [3] and track reconstruction [4], enabling physicists to infer particle properties from detector signals with greater accuracy.

Another crucial area is trigger systems, which must make real-time decisions on whether to record an event in less than a millisecond. Here, ML models are increasingly deployed on specialized hardware (e.g., FPGAs, GPUs) [5] to optimize triggers by rapidly distinguishing potentially interesting collisions from noise. ML also plays a major role in fast simulation and generative modeling [6], where deep generative networks replicate computationally expensive detector simulations, drastically reducing computation times while preserving fidelity. In parallel, unsupervised and anomaly detection methods [7] are being explored to uncover unexpected patterns in data, providing model-independent avenues to search for physics beyond the Standard Model. At a broader scale, ML contributes to detector calibration, uncertainty estimation, and data compression, improving resource efficiency and the reliability of physics results.

Despite the success of widely adopted frameworks such as TensorFlow [8] and PyTorch [9], deploying ML inference in C++ based HEP environments presents unique challenges. These frameworks are primarily designed around their native Python ecosystems and model formats, limiting flexibility when integrating externally trained models. Using TensorFlow in a C++ environment is challenging, introduces heavy dependencies, and provides limited control over thread management, making deployment complex for use cases like single-event evaluation that require extremely fast, one-at-a-time processing of individual collision events, as is common in HEP workflows. PyTorch offers LibTorch, a C++ interface that is more lightweight and easier to integrate. However, extensions such as PyTorch Geometric are not supported in libtorch, as they rely on Python-heavy APIs and do not provide native C++ bindings. These constraints underscore the need for lightweight, high-performance inference solutions that integrate seamlessly into C++ based pipelines while maintaining both flexibility and efficiency.

2.2. High-Luminosity LHC

The High-Luminosity Large Hadron Collider (HL-LHC) [10] is the major upgrade of the LHC at CERN, planned to begin operation in the mid-2030s, with the goal of increasing the collider's integrated luminosity by an order of magnitude compared to its predecessor. This dramatic boost in collision rates will allow physicists to probe rare processes with unprecedented precision, improve measurements of the Higgs boson, and enhance sensitivity to potential new physics. However, the HL-LHC will also introduce significant challenges: detectors must handle higher particle densities, event pileup will reach up to 200 simultaneous interactions per bunch crossing, and data volumes will grow to exabyte scales. These requirements demand innovations in data processing, triggering, reconstruction, and storage. Machine learning will play a critical role in meeting these challenges, from enabling ultra-fast, resource-efficient trigger decisions on specialized hardware, to improving event reconstruction in high-pileup environments, accelerating detector simulations, and

guiding anomaly detection for unexpected signatures. By combining physics insight with advanced ML methods, the HL-LHC program will ensure that researchers can fully exploit the collider's enhanced discovery potential.

2.3. Next-Generation Triggers Project

An important step toward addressing the challenges of HL-LHC is the Next-Generation Triggers (NGT) project [11], a CERN-wide initiative that aims to prepare the LHC experiments for the unprecedented data rates expected in the HL-LHC era. With collisions producing up to 100 terabytes of raw data per second, traditional approaches to triggering and data reduction are insufficient. The NGT project brings together researchers across experiments and the CERN Theory Department to co-develop software and hardware strategies for real-time data processing. A central focus of NGT is the integration of machine learning inference into trigger systems in a way that is both scalable and low-latency. To achieve this, the project emphasizes the design of efficient interfaces that minimize data movement between detector front-ends, memory, and accelerators, as well as the development of portable solutions that can run seamlessly across heterogeneous architectures (CPUs, GPUs, FPGAs, and potentially AI-specific hardware).

3. Machine Learning Inference at the Experiments

3.1. ATLAS

ATLAS (A Toroidal LHC ApparatuS) is one of the two general-purpose detectors at the LHC [12], designed to study a wide range of collision events, including proton–proton and nucleus–nucleus interactions. Alongside CMS, ATLAS played a leading role in the 2012 discovery of the Higgs boson. Like the other LHC experiments, ATLAS faces the primary challenge of handling the immense volume of data generated during collisions. Its complex trigger system is tasked with rapidly selecting interesting events from this continuous data stream. Research within ATLAS [13] has shown that convolutional and recurrent neural networks may outperform traditional signal filters for estimating the energy and timing of signals in the Electromagnetic Calorimeter (ECAL)- a subdetector that measures the energy of particles, particularly electrons and photons. Advances in fast machine learning methods may also being leveraged for accurate regression of key physical quantities, leading to improved calibration of detector signals.

Machine learning may further support reconstruction algorithms, which transform raw detector signals into physics objects and phenomena, enabling higher-level analyses. Sophisticated architectures such as Transformers can be applied to study the simulated decays of particles like b-hadrons, while anomaly detection techniques are increasingly explored to isolate unexpected phenomena from well-understood backgrounds. To support these diverse applications, ATLAS employs a robust inference infrastructure to integrate trained ML models into production workflows. Its primary software framework, Athena [14], serves as the production environment for processing collision data and is built upon Gaudi, a shared software stack used by several CERN experiments.

3.1.1. *ML Inference in Athena*

Athena employs ONNXRuntime as its primary tool for machine learning inference [15], running models in the ONNX (Open Neural Network eXchange) format [16]. ONNXRuntime is particularly suitable because it allows seamless switching between different execution providers via job configurations, offering flexibility and maintainability. In addition to ONNXRuntime, ATLAS also leverages NVIDIA Triton Inference Server [17] to enable Inference-as-a-Service within the Athena framework. While ONNXRuntime serves as the main

inference backend, scalable strategies are essential to meet the increasing demands of simulation and collision data processing, including maximizing event throughput and efficiently utilizing accelerators such as GPUs.

To address these requirements, the AthenaTriton integration allows Athena to act as a Triton client, sending inference requests to a local or remote Triton server. This approach supports both online and offline computing workflows and enables scalable deployment of ML models. Triton provides multiple backend options, including ONNXRuntime and TensorRT [18], as well as custom Python and C++ backends, ensuring broad flexibility. Performance evaluations using `perf_analyzer` demonstrated that throughput scaling efficiency remained above 98% as concurrent model instances increased, while GPU utilization reached ∼45%. End-to-end tests with Athena clients further showed that three concurrent threads achieved a 2.4× speedup with strong scaling efficiency. Together, these results highlight AthenaTriton as a robust and maintainable solution for integrating ML inference into ATLAS workflows at scale.

3.2. CMS

CMS (Compact Muon Solenoid) is the other general-purpose detector [19] at the LHC. The CMS collaboration investigates a wide range of topics, from precision studies of the Standard Model to searches for extra dimensions and dark matter. Although CMS shares the same broad scientific goals as ATLAS, it differs in its technical design, particularly in the magnet system and detector configuration. CMS has been actively exploring machine learning across diverse applications [20], including convolutional neural networks to denoise faster [21], lower-quality detector simulations, CNNs for identifying hadronic tau particles, and graph neural networks for particle-flow reconstruction. More recently, the ECAL has deployed autoencoder-based anomaly detection models [22] to enhance data quality monitoring.

Like ATLAS, CMS operates its own dedicated software framework- CMSSW [23], for simulation, calibration and alignment, and reconstruction, enabling the processing and analysis of collision data by the collaboration. The primary goal of this framework and its event data model is to provide a robust environment for developing, integrating, and scaling reconstruction and analysis software, while supporting the incorporation of advanced machine learning techniques into physics workflows.

3.2.1. *ML Inference in CMSSW*

For its machine learning use cases ranging from jet tagging with the graph-convolutional model ParticleNet [24] to end-to-end reconstruction of particle hits into clusters using the GravNet architecture [25] with dynamic graph building, CMS employs both direct and indirect inference strategies. For direct inference, CMS primarily uses ONNXRuntime, which is straightforward to maintain and relatively tolerant of different GCC/CUDA versions. However, this approach faces limitations, particularly with graph neural networks, which are often not fully supported in ONNX. To address such cases, CMS has been exploring direct PyTorch support [26], especially for models that are difficult to export to ONNX or that require custom operations. To further integrate deep learning into HEP workflows, CMSSW supports both ahead-of-time and just-in-time compilation strategies, and recent work has investigated enabling inference on heterogeneous architectures via ALPAKA [27], connecting directly to PyTorch tensors.

For indirect inference, CMS uses the SONIC (Service for Optimized Network Inference on Coprocessors) approach [28] for Inference-as-a-Service capabilities. In this model, CPU-based CMSSW clients send inference requests to remote coprocessor servers, where a Triton Inference Server executes the models. Triton supports multiple model instances, dynamic

batching, and asynchronous inference requests per ML algorithm per event, while CMSSW clients continue data processing in parallel. SONIC offers flexibility across coprocessor types (GPUs, IPUs, FPGAs) and ML backends, and even supports non-ML GPU workloads through custom Triton backends. To extend this capability to large-scale distributed environments, the SuperSONIC [29] package enables Kubernetes-based deployment with load balancing, autoscaling, and rate limiting across multiple GPUs, along with observability via Prometheus, Grafana, and OpenTelemetry. Clients discover available servers through site configurations, with fallback to local CPU or GPU inference when remote resources are unavailable. Ongoing development focuses on improving robustness through retry mechanisms, expanding supported ML models, and preparing the framework for Run 3 operations and the upcoming High-Luminosity LHC era.

3.3. LHCb

The LHCb (Large Hadron Collider beauty) experiment [30] is one of the two specialized collaborations at the LHC. Its primary goal is to search for indirect evidence of new physics through studies of charge–parity violation and rare decays of beauty and charm hadrons. Like other LHC experiments, LHCb relies on machine learning models for a variety of tasks, including track reconstruction using models such as the Bonsai BDT (Boosted Decision Tree) [31], particle identification (e.g., electrons, muons, pions) [32] modeled as a multiclass classification problem with neural networks and XGBoost, and particle decay selection using topological triggers through MatrixNet [33].

Due to LHCb's specialized design, its detectors produce a readout of $\sim$5 TB/s, processed via a software-based trigger system without requiring dedicated hardware triggers [34]. The experiment employs a dual High-Level Trigger (HLT) system: HLT1 performs fast track reconstruction [35], while HLT2 executes high-fidelity reconstruction [18]. ML inference is extensively online triggers and simulation frameworks, with throughput constraints guiding the choice of methods. In the GPU-based HLT1, small MLPs are executed on NVIDIA A5000 GPUs, leveraging general-purpose libraries such as ONNX Runtime and TensorRT , which provide flexibility and standardized model support, though kernel overhead can limit performance compared to custom solutions. In the CPU-based HLT2, LHCb uses custom compile-time inference tools integrated into the Gaudi framework, enabling SIMD vectorization, compile-time optimizations, and efficient weight loading. This approach achieves 2–3$\times$ speedups in reconstruction timing and supports rapid retraining and deployment via a PyTorch-based pipeline.

For simulation, fast inference libraries including the PyTorch C++ API and ONNXRuntime are integrated to support research and unify inference across the software stack. Key ML applications include ghost track rejection, particle identification, and classification of reconstructed objects from heavy-flavor decays, typically using compact MLPs with 10–20 input features. Overall, while custom compile-time inference currently dominates online workflows due to its superior speed, ongoing efforts focus on generalizing flexible inference libraries and building maintainable, high-performance, unified ML pipelines across LHCb.

3.4. ALICE

ALICE (A Large Ion Collider Experiment) [36] is a specialized LHC experiment designed to study the physics of strongly interacting matter at extreme energy densities, where a phase of matter called the quark-gluon plasma forms. Like other LHC experiments, ALICE employs machine learning [37] for tasks such as particle identification, jet reconstruction, and simulations. Applications include neural networks for particle identification [32], BDTs for event selection, and clustering in the Time Projection Chamber (TPC), which is the central barrel detector used for tracking charged particles and performing particle identification.

ALICE's ML pipeline typically involves model development in PyTorch, followed by export to the ONNX format, and inference through the ONNXRuntime's C++ API [38]. ONNXRuntime serves as the primary inference framework, leveraging both CPU and GPU resources to manage the high data rates and occupancies of Run 3. ML applications span physics analysis tasks such as TPC particle identification, jet-pT background correction, and heavy-flavor BDT triggers, as well as online workflows like GPU-based TPC clustering at 3.5 TB/s [39]. ORT allows PyTorch models to be converted into efficient inference graphs and integrated into C++ workflows with dynamic column access for flexibility. GPU acceleration is central, with custom CUDA/ROCm stream implementations supporting multiple parallel streams per GPU, achieving orders-of-magnitude speedups compared to CPU execution. Additional developments include GAN-based fast simulations, graph networks for track matching, and generalized particle identification using multiple detectors. Despite challenges such as memory inefficiencies and multi-threaded execution issues on SLURM clusters, ALICE has built a fully functional inference framework, making ML indispensable for both online reconstruction and offline analysis in the high-rate environment of Run 3 and beyond.

4. Standardized Inference Runtimes

Examining the machine learning requirements across the experiments and their inference workflows highlights the extensive use of tools such as ONNXRuntime and TensorRT.

4.1. ONNXRuntime

ONNX is a standardized format for representing deep learning models, designed to facilitate interoperability between different frameworks such as TensorFlow and PyTorch. By providing a common representation, ONNX allows trained models to be exported and deployed across a variety of runtime environments and hardware platforms. However, ONNX cannot fully represent all model architectures, particularly those used in graph neural networks, which limits its applicability for certain HEP models.

To enable efficient inference of ONNX models, Microsoft developed ONNXRuntime, an open-source engine supporting both C++ and Python environments. ONNXRuntime runs on CPUs and GPUs, and it has been successfully integrated into HEP frameworks such as ATLAS and CMS. Its convenient C++ API and fine-grained thread control make it particularly valuable for production workflows. However, several challenges remain: certain ML operations are not supported by ONNXRuntime, which can prevent some models from being converted to ONNX altogether; inference values may vary slightly across runs; and ONNXRuntime does not natively support access to remote coprocessors- specialized hardware like GPUs and FPGAs, or the acceleration of non-ML algorithms on these devices. Despite these limitations, ONNX and ONNXRuntime provide a robust foundation for deploying machine learning models efficiently within C++-based scientific computing workflows, particularly when combined with complementary tools like TensorRT for GPU optimization.

4.2. TensorRT

NVIDIA TensorRT is an inference engine ecosystem tailored for high-performance deep learning inference on NVIDIA GPUs. It includes both a runtime and a suite of model-optimization tools, such as the TensorRT-LLM library, Model Optimizer, and Cloud services, for producing highly efficient, low-latency inference engines. TensorRT accelerates neural network inference by applying a range of optimizations: it performs precision calibration to support lower-precision formats like FP16, INT8, FP8, and INT4, significantly improving throughput and reducing memory usage; layer and tensor fusion, which merges multiple operations into fewer GPU kernels; kernel auto-tuning, which selects optimal GPU kernels per hardware; and dynamic tensor memory management, which optimizes memory

allocation during inference. It supports importing models from popular deep learning frameworks, such as TensorFlow, PyTorch, Caffe, MXNet, or through ONNX formats, offering both C++ and Python APIs for integration and optimization. TensorRT achieves up to 36× faster inference compared to CPU-only baselines and even up to 40× higher throughput with sub-7 ms latency in real-time use cases such as embedded systems and data centers.

5. HEP-developed Inference Frameworks

5.1. SOFIE

To address the challenges of efficient machine learning inference in C++ environments, the ML4EP team at CERN have been developing SOFIE (System for Optimized Fast Inference code Emit) [40], a tool within ROOT/TMVA [41] designed to generate optimized C++ code from trained ML models. SOFIE is capable of converting models in ONNX format to its own Intermediate Representation. Additionally, it provides support to parse TensorFlow/Keras and PyTorch models, as well as message-passing Graph Neural Networks from DeepMind's Graph Nets library [42]. The key advantage of SOFIE is its ability to produce standalone C++ code that can be directly invoked within C++ applications with minimal dependencies, requiring only Basic Linear Algebra Subroutines (BLAS) for numerical computations. This makes integration seamless for high-energy physics workflows and other computationally demanding applications. Moreover, the generated code can be compiled at runtime using ROOT [43], [44] Cling Just-In-Time compilation, allowing for flexible execution, including within Python environments. By eliminating the need for heavyweight machine learning frameworks during inference, SOFIE offers a highly efficient and easily deployable solution for ML model evaluation. While initially SOFIE struggled for more complex models, thereby taking more time and memory during inference as mentioned in [45], several optimizations have recently been applied to accelerate their process. These optimizations [46] include efficient reuse of intermediate memory, fusing multiple operators in the model graph, node eliminations, etc. Recent benchmarking suggests SOFIE to be

5.2. hls4ml

hls4ml (High-level synthesis for Machine Learning) [47] is an open-source software–hardware codesign workflow designed to make machine learning implementations on energy-efficient hardware accessible to domain scientists. It translates trained neural networks from common ML frameworks such as TensorFlow, PyTorch, and QKeras into digital hardware implementations using high-level synthesis (HLS) for FPGAs and ASICs. By supporting techniques like quantization-aware training (where networks are trained while simulating reduced numerical precision, improving robustness to low-bit representations), pruning, and tunable parallelization, hls4ml enables the creation of low-latency and low-power ML accelerators optimized for specific scientific applications. The framework provides Python APIs, visualization tools, and bit-accurate emulation (a software execution mode that reproduces the exact fixed-point arithmetic of the target hardware), helping non-experts explore trade-offs in latency, throughput, power, and resource usage. Initially developed for real-time, low-latency tasks in high-energy physics, hls4ml has expanded to broader low-power applications, offering end-to-end workflows for both FPGA and ASIC backends, and significantly reducing the design cycle.

5.3. PQuant

PQuant [48] is a tool designed for end-to-end hardware-aware model compression, aimed at training and optimizing compressed neural networks for deployment under strict hardware constraints. It automates model wrapping, training, and cleanup, requiring no deep knowledge of compression techniques from the user. The library supports pruning

and quantization of weights and biases, with hyperparameters fully configurable through a simple config file. Users can enable or disable pruning and quantization, choose pruning methods, and specify quantization parameters such as bit-width (default or layer-specific), symmetric or asymmetric quantization, and optional hard quantization for certain activations. Training is organized into distinct phases: pretraining, training, and fine-tuning managed by a generic training function where users only need to provide training and validation functions. PQuant allows fine-grained control, such as disabling pruning for specific layers or using different quantization strategies across layers, and it supports iterative training with weight rewinding. Results on models like SmartPixels and ResNet20 demonstrate its ability to significantly reduce parameters while maintaining accuracy. Future developments include TensorFlow support, integration with hls4ml, and automated hyperparameter optimization.

5.4. Conifer

Conifer [49] is a framework designed to map decision forests onto FPGA firmware, enabling extremely low-latency and high-throughput inference. Decision forests, which are ensembles of decision trees, remain valuable for machine learning in edge and resource-constrained environments because they are lightweight, robust, and fast. Conifer integrates with popular training tools and translates trained models into FPGA implementations, exploiting parallelism and pipelining to minimize latency. It supports both HLS and VHDL (Very High-Speed Integrated Circuit Hardware Description Language) backends, with latencies in the 10–100 ns range depending on depth and number of trees. Applications already demonstrated include medical image segmentation on embedded devices, real-time filtering in tracking detectors, and particle reconstruction in high-energy physics experiments. To address the challenge of model reconfiguration without lengthy synthesis cycles, Conifer also introduces the Forest Processing Unit (FPU), a dynamic architecture where boosted decision tree models are represented as data and efficiently evaluated across parallel tree engines. This makes Conifer a practical tool for scenarios requiring real-time predictive performance in scientific and embedded systems.

6. Conclusion

Machine learning inference has become a central component of data processing in CERN's experiments. As this survey has outlined, the challenges of scale, latency, and heterogeneous architectures are being tackled through a range of solutions, from widely used standardized runtimes such as ONNXRuntime and TensorRT to experiment-specific strategies embedded within their major software frameworks. While each experiment has developed inference pipelines adapted to its detector design and computing environment, they share common requirements: seamless integration with C++ based workflows, efficient use of heterogeneous hardware, and robustness under extreme data rates.

Looking ahead to the High-Luminosity LHC era, these requirements will only grow more demanding, with increasing event complexity and data volumes. Alongside established runtimes, a number of research and development efforts, such as SOFIE- which generates optimized standalone C++ code for model inference, and hls4ml, for FPGA-based acceleration are being explored for improved performance and hardware efficiency.

Overall, the developments surveyed here demonstrate how machine learning inference has become very important in high-energy physics. These efforts for scalable and efficient inference pipelines will be essential to the upcoming physics research of the HL-LHC and future experiments.

Acknowledgement

This work has been funded by the Eric & Wendy Schmidt Fund for Strategic Innovation through the CERN Next Generation Triggers project under grant agreement number SIF-2023-004.

References

[1] P. Clarke *et al.*, "Big data in the physical sciences: challenges and opportunities," *ATI Scoping Report*, 2016.

[2] X. Vidal, L. Dieste, and Á. Suárez, "How to Use Machine Learning to Improve the Discrimination between Signal and Background at Particle Colliders," *Applied Sciences*, vol. 11, p. 11076, 2021, doi: 10.3390/app112211076.

[3] M. Karwowska, M. Jakubowska, Ł. Graczykowski, K. Deja, and M. Kasak, "Particle identification with machine learning in ALICE Run 3." [Online]. Available: https://arxiv.org/abs/2309.07768

[4] D. Baranov, S. Mitsyn, P. Goncharov, and G. Ososkov, "The Particle Track Reconstruction based on deep Neural networks," *EPJ Web of Conferences*, vol. 214, p. 6018, 2019, doi: 10.1051/epjconf/201921406018.

[5] A. Coccaro, F. Armando Di Bello, S. Giagu, L. Rambelli, and N. Stocchetti, "Fast neural network inference on FPGAs for triggering on long-lived particles at colliders," *Machine Learning: Science and Technology*, vol. 4, no. 4, p. 45040, 2023, doi: 10.1088/2632-2153/ad087a.

[6] M. Kita, J. Dubiński, P. Rokita, and K. Deja, "Generative Diffusion Models for Fast Simulations of Particle Collisions at CERN." [Online]. Available: https://arxiv.org/abs/2406.03233

[7] K. M. Fraser, "Anomaly Detection in Particle Physics." [Online]. Available: https://indico.cern.ch/event/1198609/contributions/5366508/attachments/2653550/4595418/Fraser_LHCP_2023_Talk.pdf

[8] M. Abadi *et al.*, "TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems." [Online]. Available: https://www.tensorflow.org/

[9] A. Paszke *et al.*, "PyTorch: an imperative style, high-performance deep learning library," in *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, Red Hook, NY, USA: Curran Associates Inc., 2019. doi: 10.48550/arXiv.1912.01703.

[10] G. Apollinari, O. Brüning, T. Nakamoto, and L. Rossi, "High Luminosity Large Hadron Collider HL-LHC." [Online]. Available: https://cds.cern.ch/record/2120673

[11] Next-Generation Triggers Project, "Next Generation Triggers - Public proposal." [Online]. Available: https://doi.org/10.17181/nke6y-e3957

[12] The ATLAS Collaboration *et al.*, "The ATLAS Experiment at the CERN Large Hadron Collider," *Journal of Instrumentation*, vol. 3, no. 8, p. S8003, 2008, doi: 10.1088/1748-0221/3/08/S08003.

[13] J. Gonski, "Learning by machines, for machines: Artificial Intelligence in the world's largest particle detector." [Online]. Available: https://atlas.cern/Updates/Feature/Machine-Learning

[14] J. Elmsheuser, "ATLAS ATHENA Configuration." [Online]. Available: https://indico.bnl.gov/event/20123/contributions/78799/attachments/48664/82783/c_190723.pdf

[15] Y.-T. Chou *et al.*, "AthenaTriton: A Tool for running Machine Learning Inference as a Service in Athena." [Online]. Available: https://indico.cern.ch/event/1338689/contributions/6010068/

[16] J. Bai, F. Lu, K. Zhang, and others, "ONNX: Open Neural Network Exchange." [Online]. Available: https://github.com/onnx/onnx

[17] X. Ju, "Machine Learning Inference in ATLAS." [Online]. Available: https://indico.cern.ch/event/1565886/contributions/6595773/attachments/3100320/5493246/20250708%20IML%20ATLAS%20ML%20inference.pdf

[18] NVIDIA Corporation, "NVIDIA TensorRT 8.4 Developer Guide (v8.4.3)," 2024. [Online]. Available: https://docs.nvidia.com/deeplearning/tensorrt/archives/tensorrt-843/pdf/TensorRT-Developer-Guide.pdf

[19] The CMS Collaboration *et al.*, "The CMS experiment at the CERN LHC," *Journal of Instrumentation*, vol. 3, no. 8, p. S8004, 2008, doi: 10.1088/1748-0221/3/08/S08004.

[20] G. Kasieczka and J.-R. Vlimant, "ML overview in CMS, HEP, and beyond." [Online]. Available: https://indico.cern.ch/event/1101433/contributions/4813873/

[21] S. Banerjee *et al.*, "Denoising Convolutional Networks to Accelerate Detector Simulation," *Journal of Physics: Conference Series*, vol. 2438, no. 1, p. 12079, 2023, doi: 10.1088/1742-6596/2438/1/012079.

[22] CMS Collaboration, "CMS develops new AI algorithm to detect anomalies." [Online]. Available: https://home.cern/news/news/experiments/cms-develops-new-ai-algorithm-detect-anomalies

[23] The CMS Collaboration, "CMSSW: CMS Offline Software (GitHub repository)." [Online]. Available: https://github.com/cms-sw/cmssw

[24] H. Qu and L. Gouskos, "Jet tagging via particle clouds," *Physical Review D*, vol. 101, no. 5, 2020, doi: 10.1103/physrevd.101.056019.

[25] S. R. Qasim *et al.*, "End-to-end multi-particle reconstruction in high occupancy imaging calorimeters with graph neural networks," *The European Physical Journal C*, vol. 82, no. 8, 2022, doi: 10.1140/epjc/s10052-022-10665-7.

[26] D. Valsecchi and Y. Yao, "ML inference in CMS." [Online]. Available: https://indico.cern.ch/event/1565886/contributions/6595778/attachments/3100394/5493369/25_07_04%20-%20IML%20meeting%20-%20CMS%20ML%20inference%20report.pdf

[27] E. Zenker *et al.*, "Alpaka - An Abstraction Library for Parallel Kernel Acceleration," 2016. doi: 10.48550/arXiv.1602.08477.

[28] A. Hayrapetyan *et al.*, "Portable Acceleration of CMS Computing Workflows with Coprocessors as a Service," *Computing and Software for Big Science*, vol. 8, no. 1, 2024, doi: 10.1007/s41781-024-00124-1.

[29] Y.-T. Chou *et al.*, "SuperSONIC: Cloud-Native Infrastructure for ML Inferencing." [Online]. Available: https://indico.cern.ch/event/1496673/contributions/6637919/attachments/3127011/5546579/SuperSONIC%20FastML.pdf

[30] The LHCb Collaboration *et al.*, "The LHCb Detector at the LHC," *Journal of Instrumentation*, vol. 3, p. S8005, 2008.

[31] V. V. Gligorov and M. Williams, "Efficient, reliable and fast high-level triggering using a bonsai boosted decision tree," *JINST*, vol. 8, p. P2013, 2013, doi: 10.1088/1748-0221/8/02/P02013.

[32] Derkach, Denis, Hushchyn, Mikhail, and Kazeev, Nikita, "Machine Learning based Global Particle Identification Algorithms at the LHCb Experiment," *EPJ Web Conf.*, vol. 214, p. 6011, 2019, doi: 10.1051/epjconf/201921406011.

[33] T. Likhomanenko, P. Ilten, E. Khairullin, A. Rogozhnikov, A. Ustyuzhanin, and M. Williams, "LHCb Topological Trigger Reoptimization," *Journal of Physics: Conference Series*, vol. 664, no. 8, p. 82025, 2015, doi: 10.1088/1742-6596/664/8/082025.

[34] M. van Veghel, "ML inference in LHCb." [Online]. Available: https://indico.cern.ch/event/1565886/contributions/6595775/attachments/3100442/5493491/iml_cern_inference_meeting_mvanveghel.pdf

[35] J. van Tilburg, "Track simulation and reconstruction in LHCb." 2005.

[36] K. Aamodt and others, "The ALICE experiment at the CERN LHC," *JINST*, vol. 3, p. S8002, 2008, doi: 10.1088/1748-0221/3/08/S08002.

[37] R. Haake, "Machine Learning in Alice." [Online]. Available: https://indico.nikhef.nl/event/1122/contributions/574/attachments/300/350/2018-04-05_ML_ALICE.pdf

[38] C. Sonnabend, "ML Inference in ALICE." [Online]. Available: https://indico.cern.ch/event/1565886/contributions/6595776/attachments/3099713/5492835/IML_08072025.pdf

[39] J. The ALICE Collaboration Alme *et al.*, "The ALICE TPC, a large 3-dimensional tracking device with fast readout for ultra-high multiplicity events," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 622, no. 1, pp. 316–367, 2010, doi: 10.1016/j.nima.2010.04.042.

[40] S. An, L. Moneta, S. Sengupta, A. Hamdan, F. Sossai, and A. Saxena, "C++ Code Generation for Fast Inference of Deep Learning Models in ROOT/TMVA," *Journal of Physics: Conference Series*, vol. 2438, no. 1, p. 12013, 2023, doi: 10.1088/1742-6596/2438/1/012013.

[41] K. Albertsson *et al.*, "TMVA User's Guide (Version 4.3.0)," 2020. [Online]. Available: https://root.cern.ch/download/doc/tmva/TMVAUsersGuide.pdf

[42] P. W. Battaglia *et al.*, "Relational inductive biases, deep learning, and graph networks." [Online]. Available: https://arxiv.org/abs/1806.01261

[43] R. Brun and F. Rademakers, "ROOT – An Object Oriented Data Analysis Framework," *Nuclear Instruments and Methods in Physics Research Section A*, vol. 389, no. 1–2, pp. 81–86, 1997, doi: 10.1016/S0168-9002(97)00048-X.

[44] P. Canal *et al.*, "root-project/root: v6-36-04." [Online]. Available: https://doi.org/10.5281/zenodo.16944732

[45] L. Moneta, I. Panagou, S. Sengupta, N. Shah, and P. Wollenhaupt, "Benchmark Studies of Machine Learning Inference using SOFIE." [Online]. Available: https://indico.cern.ch/event/1338689/contributions/6016194/

[46] S. Sengupta, L. Moneta, O. Sirikova, P. Kasar, and S. Akash, "TMVA SOFIE: Enhancements in ML Inference through graph optimizations and heterogeneous architectures." [Online]. Available: https://indico.cern.ch/event/1488410/contributions/6561436/attachments/3130795/5555028/ACAT25-SOFIE.pdf

[47] J. Duarte and others, "Fast inference of deep neural networks in FPGAs for particle physics," *JINST*, vol. 13, no. 7, p. P7027, 2018, doi: 10.1088/1748-0221/13/07/P07027.

[48] R. O. Niemi *et al.*, "PQuant: A Tool for End-to-End Hardware-Aware Model Compression." [Online]. Available: https://indico.cern.ch/event/1502120/contributions/6505976/attachments/3074062/5439367/PQuant_23052025-3.pdf

[49] S. Summers, “Under the Canopy: Exploring Conifer for Low-Latency Decision Forests on FPGAs.” [Online]. Available: https://indico.cern.ch/event/1381060/contributions/5923280/attachments/2875746/5037307/conifer_fdf.pdf

Open Access | https://doi.org/10.25080/ncmh8429

SciPy 2025

July 7 - July 13, 2025

Proceedings of the 24th Python in Science Conference
ISSN: 2575-9752

Unlocking the Missing 78%: Inclusive Communities for the Future of Scientific Python

Noor Aftab[1]

[1]IBM Champion, PyData Speaker, Chair of NumFOCUS CoC WG

Abstract

Women account for **15–22 %** of the global data science workforce and less than **3 %** of commits to core Python repositories [1], [2], [3]. This "missing 78 %" drains talent, innovation, and economic value scientific Python. We first quantify the gap, then unpack its root causes, and finally propose the Visibility–Invitation–Mechanism (VIM) framework, as a practical remedy. A mixed methods case study—IBM Women in AI (WAI) User Group shows VIM in action: membership jumped from 17 to 38 within a week of its March 2025 launch (↑124 %).

Keywords scientific python, diversity, inclusion, gender gap, open source, community, VIM framework

1. Introduction

Python dominates contemporary data science and scientific computing, yet women are conspicuously scarce across its contributor base and professional user community. Longitudinal GitHub analyses show women's share of commits to high-impact Python projects rising from ≈ 2 % in 2008 to 4.9 % in 2021 [2]. Labour-market surveys place women at ≈ 20 % of data-science roles globally, lower than many adjacent STEM domains [1], [4].

1.1. Problem Statement — Why the VIM lens?

Building on organisational-learning and DEI literature, we treat inclusion as a three-pillar system: **Visibility** (whose work is seen), **Invitation** (who feels welcome), and **Mechanism** (what structural supports exist). Python's globally distributed, volunteer-heavy culture makes it an ideal proving ground—breakdowns in any pillar surface quickly as stalled pull requests, skewed conference rosters, or talent attrition.

Origin of this study. Preliminary findings were shared at PyData Global 2024 in the talk *"The Missing 78 %"* (5 Dec 2024). Feedback from that session—requests for actionable next steps—directly motivated formation of the *IBM Women in AI (WAI) User Group* two weeks later. The group now serves as a living laboratory to test VIM-aligned interventions, the results of which feed back into this paper.

A skewed contributor pool constrains cognitive diversity, perpetuates algorithmic bias, and threatens the sustainability of volunteer-driven projects. Gender-diverse technical teams outperform homogeneous teams on defect resolution, product novelty, and profitability [5], [6], [7]. Conversely, uniform developer cohorts embed discriminatory patterns into socio-technical artefacts—from résumé-screening engines to clinical devices [8], [9].

1.2. Research Objectives and Questions

Table 1. *Research questions*

Published Jul 10, 2025

Correspondence to
Noor Aftab
noor.aftab@gmail.com

Open Access

Copyright © 2025 Aftab. This is an open-access article distributed under the terms of the Creative Commons Attribution 4.0 International license, which enables reusers to distribute, remix, adapt, and build upon the material in any medium or format, so long as attribution is given to the creator.

ID	Research question
RQ1 (Scale)	How large is the gender gap in scientific-Python / data-science?
RQ2 (Barriers)	What systemic factors keep women out?
RQ3 (Interventions)	Which actions measurably close the gap?
RQ4 (Translation)	How can communities roll those actions out in practice?

1.3. Methodological Overview

Our embedded case-study design enables triangulation of large-scale secondary data with fine-grained community observations. Quantitatively, we analyze membership counts, demographic self-report, and structured survey items to detect early engagement signals. Qualitatively, we code open-ended survey comments and the PyData talk transcript to surface emergent themes. This mixed-methods approach improves complementarity—numerical trends provide breadth, while narrative data add explanatory depth. Specifically, our datasets span December 2024-June2025: membership counts (n=40), survey responses from post launch members only (n=26 of 26, 100% response rate), and qualitative transcripts(n=60 comments).

Ethical considerations. All WAI membership data are voluntary and stored under GDPR-compliant protocols. Identifiable information is aggregated or anonymized before analysis; gender and role fields are self-declared. The study received a low-risk determination from the IBM Research Ethics Committee (ID WAI 2025 04).

Secondary statistics are drawn from peer-reviewed literature and public dashboards Table 1 to locate the case in a broader empirical context.

1.4. Paper Structure

Literature Review surveys the literature; Systemic Barriers maps explanatory mechanisms; Operationalizing the VIM Framework: The IBM Women in AI User Group details the case study; A Roadmap for Fostering Inclusion in Data, AI, and Open-Source Communities synthesizes findings into the VIM roadmap; Conclusion and Future Directions outlines research and policy implications.

Table 2. *Snapshot of the Gap (latest available data)*

Domain	Women's Share	Source
GitHub Python core commits	2–3 %	[2]
Global AI workforce	22 %	[1]
US data-analytics roles	26 %	[10]

2. Literature Review

This section synthesizes empirical findings on women's underrepresentation in scientific-Python, open-source software (OSS), data science, and AI through the Visibility–Invitation–Mechanism (VIM) lens. We first quantify the gender gap (2.1), then unpack its systemic drivers (2.2), examine downstream impacts (2.3), review proven inclusion interventions (2.4), and finally position VIM relative to related frameworks (2.5).

2.1. Participation Metrics

Table 3. *Women's share across data/AI (latest)*

Segment (2024 unless noted)	Women %	Source

Python core repository commits	2–3	[2]
OSS contributors (≥ 10 commits)	5.4	[3]
Global AI talent pool	22	[1]
US data-analytics roles	26	[10]
STEM R&D researchers (G20)	22	[11]
Nat-sci papers with ≥ 1 woman author (2025)	27	[12]

*Pandemic years widened the publishing gap by **7 pp** (2019 arrow.r 2020) [13].*

2.2. Explanatory Factors

Underrepresentation arises from interlocking barriers:

1. **Pipeline attrition** – mentorship scarcity predicts lower retention [13].
2. **Gatekeeping** – pull-request acceptance drops when gender is visible; review latency is 13 % higher for women [14], [15].
3. **Psychological safety** – 72 % of women in tech report "bro-culture" microaggressions [16].
4. **Time poverty** – unpaid OSS labor conflicts with caregiving burdens [17].
5. **Algorithmic bias** – historic bias in training data propagates exclusion [18].

2.3. Outcome Differentials

- **Innovation:** Gender-heterogeneous inventor teams achieve higher patent-citation impact [19].
- **Quality:** No gender difference in Python code quality (PEP 8 compliance) [20].
- **Economics:** Exec-level gender diversity arrow.r +25 % profitability premium; boards with ≥ 3 women arrow.r +66 % ROI [21], [22].

2.4. Evidence on Effective Interventions

Table 4. *High-impact inclusion programs*

Initiative	Pillars	Key Features	Reported Outcomes
NumFOCUS DISC / PyData	V, I	Scholarships; contributor leads	120 awards; CZI onboarding pipelines [23]
Ada Developers Academy	I, M	Tuition-free; paid internships	97 % grad rate; 94 % placement; 152 % salary uplift [24]
Outreachy	I, M	Paid OSS internships; one-on-one mentorship	92 % women interns; 64 % PoC [25]
TechWomen	V, I, M	Immersive mentorship	> 80 % alumnæ mentor girls in STEM locally [26]

Proven programs hit multiple VIM pillars; integrated designs, paid stipends, and affinity networks drive the strongest, most sustained impact.

2.5. Related Frameworks and Positioning of VIM

Two lenses dominate inclusion work today: **Communities of Practice (CoP)** explains social learning but lacks structural fixes, while **CHAOSS DEI Metrics** supplies quantitative dashboards yet can slip into "checklist" mode. Our contribution, **Visibility–Invitation–Mechanism (VIM)**, blends CoP's community focus with CHAOSS's rigorous metrics **and**

adds funded levers—mentorship stipends, “good-first-issue” curation—to turn insight into action.

Table 5. *Frameworks compared across five dimensions*

Framework	Social Dimension	Structural Dimension	Actionability	Metrics	Subjective Experience
CoP	Strong	Weak	Moderate	Weak	Moderate
CHAOSS DEI	Weak	Strong	Strong	Strong	Weak
VIM	Strong	Strong	Strong	Moderate	Strong

2.6. Visual Illustrations

The following figures visually illustrate the conceptual synthesis Figure 1 and dimensional strengths [fig:dimensional] of the frameworks, offering both theoretical and practical comparisons.

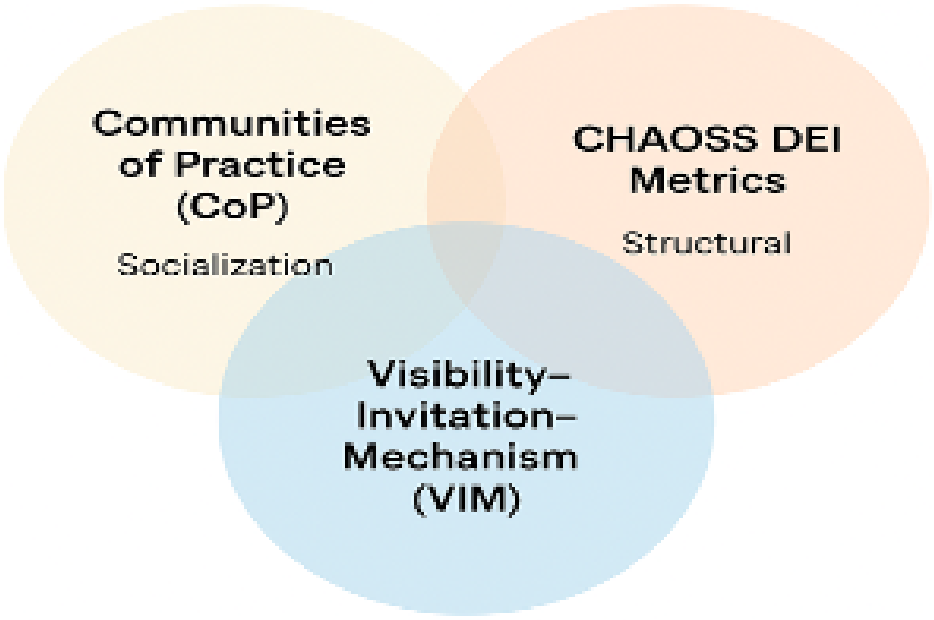


Figure 1. *Conceptual synthesis. Venn diagram illustrating the conceptual overlap and synthesis of CoP, HAOSS DEI, and VIM frameworks, with VIM positioned at the intersection and extending beyond.*

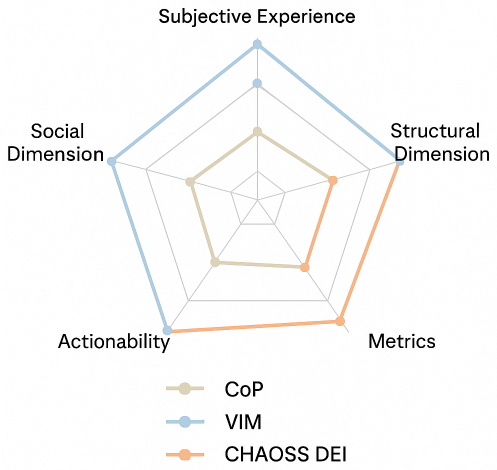


Figure 2. *Dimensional strengths. Radar chart comparing CoP, CHAOSS DEI, and VIM across five inclusion dimensions: Social, Structural, Actionability, Metrics, and Subjective Experience.*

3. Systemic Barriers

Five mutually reinforcing hurdles keep women from fully participating in scientific Python. Each erodes one or more pillars of the Visibility–Invitation–Mechanism (VIM) framework. These barriers mirror broader patterns in open-source participation, underscoring that the challenges are systematic rather than isolated.

Table 6. *How five barriers erode VIM pillars*

#	Barrier	Main VIM Pillar Hit	How it Shows Up (Metric)
1	Pipeline attrition	Invitation	Women earn 20 % of OECD ICT degrees and hold 22 % of G20 STEM posts [1], [11]
2	Gatekeeping	Visibility	Women's PRs wait 13 % longer; acceptance drops if gender is visible [14], [15]
3	Psychological-safety gaps	Visibility	72 % of women in tech report "bro culture" micro-aggressions [16]
4	Time and care load	Invitation	Women perform ∼76 % of unpaid care, limiting volunteer contributions [27]
5	Algorithmic feedback loops	Mechanism	60 % of audited tools downgrade résumés with female names [18]

3.1. Causal Chain

GitHub data on 600 k users show a loop: pipeline leaks shrink the pool of senior female reviewers arrow.r women's PRs merge 13 % slower and quit sooner (18 % vs 35 % active after 10 yrs) arrow.r hiring algorithms see fewer female exemplars and keep filtering them out [2].

4. Operationalizing the VIM Framework: The IBM Women in AI User Group

4.1. From "The Missing 78 %" Talk to a Live Community

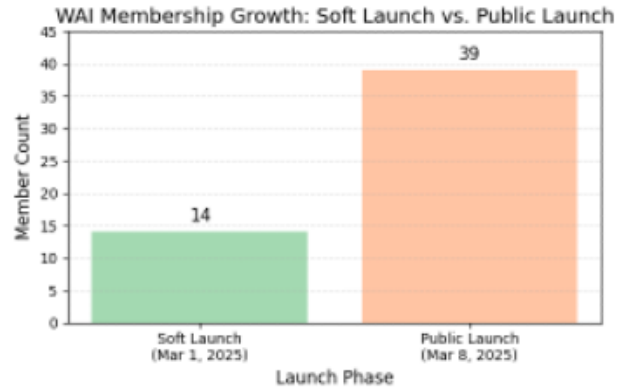
On 5 December 2024 at PyData Global, we presented "The Missing 78 %," revealing that women represent only 22 % of AI professionals and < 3 % of core Python contributors. Audience demand for actionable next steps led to the rapid formation of the IBM Women in AI (WAI) User Group, founded two weeks later to put our Visibility–Invitation–Mechanism (VIM) framework into practice.

Table 7. *Phased Launch Timeline (Dec '24–Mar '25 arrow.r Mar '25)*

Phase	Key Dates	Activities & Metrics
Soft Launch	16 Dec 2024 – 1 Mar 2025	Invite-only Slack/GitHub rehearsal; baseline survey (14/14 responses)
Public Launch	8 Mar 2025	LinkedIn + email: +21 sign ups (10% via LinkedIn)
Today		39 Active Members; live lab for VIM

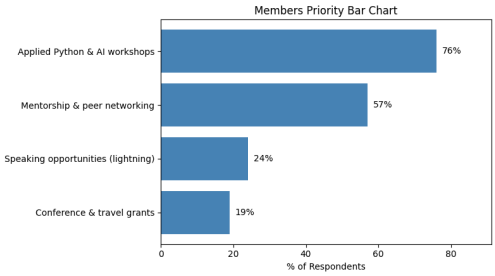
4.2. Phased Rollout & Membership Growth

Membership grew from **14** (Soft Launch) to **39** (Public Launch) in one week—a **124 % jump**—demonstrating immediate interest. Importantly, retention was effectively 100% in the first month, with 85% of the members remaining active month-to-month. While this high rate may partly reflect the same sample size, it signals strong early commitment beyond the launch spike.



4.3. Member-Needs Survey: Shaping Python & AI Offerings

In mid-March 2025, we surveyed all **39** WAI members (21 responses, 54 % response rate). Ranked priorities:



Key takeaways:

1. **Mechanism:** 76 % want hands-on AI/Python labs
2. **Invitation:** 57 % seek mentor pairs/peer networking
3. **Visibility:** 24 % lightning talks; 19 % travel grants

These findings confirm a focus on **Mechanism**, **Invitation**, and **Visibility** in WAI programming.

4.4. VIM-Aligned Program Mechanisms & Phasing

Building on VIM and survey results, we distilled four initiatives. Each maps to Visibility (V), Invitation (I), and Mechanism (M), with timing, targets, and A Roadmap for Fostering Inclusion in Data, AI, and Open-Source Communities linkage.

Table 8. *Phase-I rollout (Mar 2025 – Dec 2025) with Sec 5 links*

Initiative	Pillars	What Happens	Kick-off	Phase I Target	Sec 5 Action
Applied Workshops	M	Monthly AI/Python labs	Sept ‘25	6 sessions ≥ 70 % attendance	**A3** A3
Mentor-Match 2.0	I	Bi-monthly mentor/mentee pairing	Oct ‘25	10 active pairs	**A4** (Sec 5.5)
Lightning-Talk Series	V	Quarterly 15 min talks (YouTube/ blog)	Feb ‘26	2 talks in H1 ‘26	**A5** (Sec 5.6)
Travel Grants Program	I, V	Micro-grants for conferences + post-trip share-out	Dec ‘26	3 awards by Q1 ‘26	**A5** (Sec 5.6)

Legend: V = Visibility I = Invitation M = Mechanism

Phasing rationale: Phase I programs launch in 2025–26, paced by member demand (76 % want hands-on labs; 57 % mentorship) and aim for six workshops, ten mentor pairs, two lightning talks, and three travel grants within 12 months.

4.5. Ongoing Evaluation & Continuous Improvement

WAI runs a **quarterly scorecard** + **feedback loop** to keep each pillar on track.

Table 9. *Core KPIs (captured quarterly)*

Dimension	Sample Metrics
Engagement	Workshop attendance %; mentor-mentee check-ins; talk submissions
Contribution	First-time PRs; docs added; demo deployments
Equity	Speaker gender ratio; awardee demographics; cross-pillar participation

Retention	6- / 12-month churn; repeat event attendance

Qualitative channels: Post-event NPS surveys; Slack sentiment polls; biannual focus groups
Improvement cycle: *Action arrow.r Measure arrow.r Adapt*—data triggers topic shifts, pairing tweaks, or grant-size recalibrations each quarter.

WAI's phased, member-driven rollout grew the group from 14 arrow.r 39 in six months, showing how VIM's action-insight-adapt loop converts research into real impact.

5. A Roadmap for Fostering Inclusion in Data, AI, and Open-Source Communities

(June 2025 – onward; builds directly on Operationalizing the VIM Framework: The IBM Women in AI User Group's Phase I pilots.)

The Visibility–Invitation–Mechanism framework—and its concrete realization via WAI's four Phase I initiatives (Operationalizing the VIM Framework: The IBM Women in AI User Group)—provides a scalable blueprint for closing the "Missing 78 %" gap Section 5. To expand beyond WAI's single-community proof of concept, we outline ten practical Actions. The first four mirror WAI's Phase I pilots; the remaining six build on those foundations and stretch into broader partnerships, metrics, and shared learning.

5.1. The Imperative for Systemic Change

Open-source AI and data projects power global innovation, yet women frequently represent fewer than 10 – 15 % of contributors—and far less among maintainers Section 5. This imbalance risks embedding bias in critical technologies. Systemic, data-driven inclusion is a necessity, not a luxury.

5.2. Action 1: Develop and Distribute an Open Inclusion Toolkit

(Piloted in WAI: baseline survey instrument; CoC draft; best-practice templates—see Section 12 – Section 14)

Objective: Empower any Data/AI project with turnkey, research-informed resources to launch inclusion initiatives.
Implementation:

1. GitHub toolkit repository containing:
    - Sample Codes of Conduct (CoC) with enforcement guidance
    - Validated needs-assessment surveys (modeled on WAI's baseline)
    - "Good first issue" annotation guidelines
    - Mentorship program blueprints (peer, group, one-on-one)
    - Metrics dashboards for attendance, engagement, and diversity
2. Community Contributions: Encourage maintainers to upload case studies, templates, and translations.

Impact: Lowers the barrier for any Data/AI project to adopt evidence-based inclusion strategies (building on WAI's early toolkit usage in Section 12).

5.3. Action 2: Establish a "VIM-Ready" Project Index

(Piloted in WAI: Slack/GitHub onboarding guidelines; WAI's own VIM alignment—see Section 15.)

Objective: Help underrepresented developers locate welcoming, well-supported open-source projects.
Implementation:

- Curated YAML registry of Data/AI repositories meeting minimum VIM criteria:
  - Visibility: Clear, up-to-date documentation and contribution instructions
  - Invitation: Defined mentor contacts or newcomer liaisons
  - Mechanism: Enforced CoC and accessible grievance processes
- Badge program: Offer a "VIM-Ready" badge for READMEs to signal inclusion readiness.

Impact: Provides newcomers with trusted entry points and incentivizes projects to maintain inclusive standards.

5.4. Action 3: Coordinate Regular Inclusion & Contribution Sprints

(WAI Pilot: "Applied Workshops," Apr 2025 – Sep 2025; see Section 15, Action 3 above.)

Objective: Create low-barrier, supportive events that onboard new contributors to Data/AI projects.
Implementation:

- Quarterly virtual sprints focused on documentation, test coverage, and labeling "good first issues."
- Mentorship stations using the Open Inclusion Toolkit to pair volunteers with newcomers for real-time guidance.
- Cross-project themes (e.g., accessibility improvements, tutorials, diversity data collection) to unify participation.

Impact: Enables structured, supportive experiences that mirror WAI's Applied Workshops in Phase I: immediate contributions, skill building, and community bonds.

5.5. Action 4: Forge Strategic Partnerships for Talent Development

(WAI Pilot: "Mentor-Match 2.0," Jun 2025 – Sep 2025; see Section 15, Action 4 above.)

Objective: Build a multi-sector support network—industry, academia, nonprofits—to sustain diverse Data/AI talent pipelines.
Implementation:

1. Industry Engagement: Encourage companies to:
   - Sponsor open-source mentorship hours within employee volunteer programs
   - Offer internships/fellowships targeting inclusive project contributions
   - Publicize transparent pay and promotion practices rewarding community engagement
2. Academic Collaboration: Integrate open-source contributions and sprint participation into Data/AI curricula and capstone courses.
3. Nonprofit Synergy: Partner with PyLadies, R-Ladies, Black in AI, Data Umbrella, etc., for joint events, shared resources, and broader outreach.

Impact: Replicates and scales WAI's Mentor-Match 2.0, fostering supportive peer networks and professional development across the broader ecosystem.

5.6. Action 5: Champion Inclusive Community Events & Conferences

(WAI Pilot: "Lightning-Talk Series," Jul 2025 & Oct 2025; see Section 15, Action 5 above.)

Objective: Transform Data/AI gatherings—online and in person—into welcoming, equitable forums that expand networks and skills.
Implementation:

- Travel grants & childcare stipends to lower financial/logistical barriers for attendees.
- Speaker diversity initiatives: Blind review, proactive recruitment of underrepresented speakers, and mentorship for first-time presenters.
- Accessibility services: Captioning, sign-language interpretation, sensory-friendly spaces.
- Inclusive networking: Structured mixers (e.g., speed mentoring), affinity-group meetups, facilitated small-group discussions.

Impact: Mirrors WAI's Lightning-Talk Series, providing underrepresented voices with visibility and real-world speaking experience.

5.7. Action 6: Adopt Ecosystem-Wide Inclusion Metrics

(WAI Pilot: Quarterly "Engagement/Contribution/Equity/Retention" metrics, Apr 2025 – ongoing; see Section 16, Action 7 above.)

Objective: Establish transparent benchmarks and accountability for inclusion efforts across projects and organizations.
Implementation:

- Inclusion scorecard capturing:
  - Contributor demographics (voluntary, anonymized)
  - Time-to-first-contribution for newcomers
  - Gender/racial diversity in leadership and maintainers
  - Adoption of VIM-Ready practices
- Public reporting: Encourage quarterly publication of scorecards—progress, challenges, next steps.

Impact: Shifts inclusion from a "nice to have" to a measurable shared responsibility—mirroring WAI's own quarterly metrics cycle.

5.8. Action 7: Cultivate Shared Learning & Iteration

(WAI Pilot: Biannual focus groups and post-event surveys, Apr 2025 – ongoing; see Section 16.)

Objective: Keep inclusion strategies adaptive by surfacing lessons learned and evolving best practices in real time.
Implementation:

- Case study repository: Aggregate anonymized program results (surveys, attendance, qualitative feedback) into the Open Inclusion Toolkit.
- Annual "Inclusion in Data/AI" symposiums: Community leaders share innovations, pitfalls, and emerging tools.
- Meta-mentorship for maintainers: Train project leads on fostering inclusive cultures, conflict resolution, and data-driven decisions.

Impact: Transforms the ecosystem into a living lab—continuously improving and sharing effective inclusion tactics, just as WAI's feedback loops did.

5.9. Community Participation & Contribution Paths

(Built on WAI's pilot projects in Section 15; expanded for general audience.)

To transform strategy into action, provide women—and all newcomers—with clear, ready-to-use entry points. Below is a curated list of repositories actively seeking diverse contributors, all with beginner-friendly labels and inclusive cultures:

Project	Highlights
NumPy	“good first issue” labels; active “NumPy Mentors” channel; quarterly newcomer sprints (Action 3 pilot).
scikit-learn	“help-wanted” board; monthly contributor office hours; automated CoC enforcement via CLA bot (Action 1 pilot).
Pandas-GUI	“easy” label for UI bugs; rotating time-zone community calls; explicit mentorship contacts (Action 4 pilot).
TensorFlow Docs	Documentation sprints; onboards 20+ first-time writers/year; CoC with ombudsperson (Action 1 & 7 pilot).
MLFinLab	Financial-ML library with “newcomer track”; pair-programming sessions; inclusive commit guidelines (Action 3 pilot).

Why it matters: By spotlighting repositories that adopt VIM-Ready practices, we lower intimidation and accelerate skill development, just as WAI’s own pilots did Operationalizing the VIM Framework: The IBM Women in AI User Group.

5.10. Partner Companies & Organizations

(Scaling WAI’s pilot partnerships from Section 15 into a broad coalition.)

IBM’s VIM-driven launch is evolving into an industry-wide inclusion platform. Early adopters and committed partners include:

Company/Org	Commitment
IBM	Host WAI User Group; maintain open-source toolkit; link inclusion work to internal recognition and compensation.
Google (OSPO)	Co-host quarterly sprints; integrate the inclusion toolkit into Google’s Open-Source Programs Office.
Microsoft	Launch “Women in AI + .NET” mentorship cycles; publish a multi-tenant inclusion dashboard; provide PyCon travel grants.
NumFOCUS	Embed “VIM-Ready” badge across PyData, Dask, FastAPI ecosystems; coordinate cross-project onboarding sprints.
Intel	Sponsor “Inclusion in AI” webinars; provide hardware stipends for workshop participants.

Scaling tip: Establish an Inclusion Council—a rotating committee of HR, open-source program leads, and community managers from each partner—to share governance, coordinate metrics, and drive cross-company accountability (Action 4 scaled).

5.11. Projects & Communities Adopting VIM Practices

(Demonstrating Operationalizing the VIM Framework: The IBM Women in AI User Group’s impact beyond WAI.)

Several Data/AI communities have already piloted VIM-aligned elements. Sharing these success stories inspires others and illustrates best practices:

Project/Community	Adoption

Project/Community	Adoption
R-Ladies (global)	Forked CoC templates; labeled 37 “good first issues” during Feb 2024 sprint; peer-led mentor groups (Action 1 & 3 pilot).
Black in AI	Deployed WAI’s baseline survey for their 2025 mentorship cohort; published anonymized findings in Jun 2025 (Action 2 pilot).
PyData Global	Introduced “VIM-Ready” badge for accepted talks; prioritized first-time speakers in 2025 CFP round (Action 5 pilot).
SciPy	Added “newcomer hours” in Jun 2025 mini-summit; tracked diversity metrics in all session lists (Action 6 pilot).
Data Umbrella	Incorporated inclusion dashboard into volunteer onboarding; automated “welcome guide” distribution (Action 7 pilot).

Next steps: Encourage each adopter to submit a brief case study (2 – 3 slides) on their experience—tools used, challenges, and impact metrics—so we can refine the toolkit further (feeds back into Action 1 & 7).

5.12. Feasibility and Time Horizons

While all seven actions are critical, they differ in feasibility and time horizon:

- **Short-term / Plug-and-play (0–6 months):** Actions 1, 3, and 5 can be implemented quickly with existing templates and event formats.
- **Medium-term / Community strengthening (6–18 months):** Actions 2, 6, and 7 require coordination and adoption cycles but provide self-sustaining feedback once established.
- **Long-term / Institutional support (18+ months):** Action 4 requires sustained buy-in across industry, academia, and nonprofits to scale impact.

6. Conclusion and Future Directions

This final section synthesizes the primary contributions of our study, offering a thorough reflection on the methodological innovations employed, the constraints encountered during the research, and comprehensively lays out a forward-looking agenda designed to guide future research and practice in fostering inclusive digital communities. We conclude by presenting concrete implications derived from our findings and issuing a collective call to action, urging stakeholders to collaboratively drive sustained and meaningful inclusion within Data, AI, and open-source communities worldwide.

6.1. Key Findings

- **Scale of the Gap**: The research reaffirms that women consistently make up only about 22% of AI professionals and a starkly lower figure of under 3% of core Python contributors. This persistent “Missing 78%” signifies not just a numerical disparity but also represents a vast, largely untapped reservoir of expertise, diverse creativity, and significant economic value that remains unrealized.
- **VIM Framework Effectiveness:** The study demonstrates that coordinating strategic efforts across Visibility initiatives (such as lightning talks to showcase emerging talent and inclusion dashboards to track progress), Invitation mechanisms (like the Mentor-Match program and the curation of “good first issues” for newcomers), and Mechanism enablers (including hands-on technical workshops and the consistent enforcement of Codes of Conduct) can rapidly bootstrap engagement and foster a welcoming environment. The WAI User Group’s notable growth from an initial 14

foundational members to 39 active participants within a brief three-month period vividly exemplifies this powerful synergy and the framework's potential.
- **Data-Driven Alignment:** Our Member-Needs Survey, which achieved a robust 54% response rate, provided crucial insights, confirming that a significant 76% of members prioritize applied, hands-on workshops, and 57% value mentorship opportunities. Importantly, the survey also revealed that even lower-ranked visibility activities, such as lightning talks and micro-grants for conference participation, deliver outsized benefits in terms of boosting member confidence and significantly enhancing their professional profile building.
- **Ecosystem Leverage:** The engagement of early partners—including prominent organizations like IBM, Google OSPO, Microsoft, NumFOCUS, and Intel—powerfully demonstrates that an open, multi-tenant toolkit and a shared recognition model can effectively scale best practices far beyond individual organizational boundaries. This collaborative approach accelerates collective progress by fostering a shared learning environment and standardizing effective inclusion strategies.

6.2. Novel Methodology

- Integrated Mixed Methods: A key methodological innovation was the combination of large-scale secondary data analyses (including GitHub commit histories and broad workforce surveys) with rich, community-level qualitative and quantitative data (derived from Airtable questionnaires, Slack communication logs, and detailed event metrics). This integrated approach allowed the study to achieve both the breadth necessary for generalizable insights and the depth required for nuanced understanding of community dynamics.
- Operationalization of VIM: By systematically sequencing pillar-specific pilots within the VIM framework—each validated through iterative feedback loops with the community and tracked via quantitative scorecards—we successfully forged a replicable playbook. This playbook can now be adopted and adapted by other communities seeking to implement similar evidence-based inclusion strategies.
- Rapid Research-Into-Action Cycle: The ability to convert audience questions and identified needs at the PyData Global conference into a fully functioning inclusion incubator, the WAI User Group, merely two weeks later vividly illustrates an agile, impact-oriented scholarship model. This model prioritizes timely intervention and practical application of research findings to address pressing community challenges effectively.

6.3. Limitations

- Cohort Size & Diversity: With an initial 39 members, the WAI User Group's cohort provides meaningful early signals and qualitative insights; however, achieving broader generalizability across different types of communities and diverse geographical regions will necessitate studies involving larger and more varied samples.
- Short-Term Evaluation: The study's six-month evaluation horizon effectively captures rapid initial growth and high levels of member satisfaction but, by its nature, does not yet measure crucial long-term outcomes such as sustained retention rates, member promotion trajectories, or tangible salary impacts over extended periods.
- Self-Selection Bias: It is acknowledged that early adopters of new initiatives like WAI are often highly motivated and enthusiastic individuals. This inherent self-selection bias may inflate initial engagement and satisfaction metrics relative to what might be observed in a broader, less self-selected population, a factor to consider when interpreting the early results.

- Contextual Dependencies: The WAI User Group significantly leverages IBM's existing platforms, established resources, and brand recognition. Consequently, community-driven initiatives or smaller-scale projects attempting to replicate these efforts may need to carefully adapt recommendations, particularly those around compensation linkages and substantial infrastructure investments, to suit their unique contexts and available resources.

6.4. Future Research Agenda

- Longitudinal Career Tracking: Future efforts should monitor WAI alumni and VIM-Ready contributors over an extended period of 1–3 years to rigorously assess the long-term impacts of participation on key career indicators such as job promotions, salary increases, academic publication records, and continued engagement in open-source software participation. This will provide deeper insights into the sustained value of VIM interventions.
- Cross-Ecosystem Replication: It is important to test and refine the VIM framework in adjacent technical domains—including burgeoning fields like DevOps, cloud-native development, data visualization, and the scientific R programming community—to validate the relevance of its core pillars and to surface any necessary domain-specific adaptations for optimal effectiveness.
- AI-Enabled Personalization: The development of ML-driven tools presents an exciting opportunity to optimize mentor-mentee pairing (e.g., through advanced skill clustering, granular interest matching, and personality compatibility assessments) and to build predictive models that can identify early disengagement risks among members, thereby enabling proactive and targeted interventions.
- Deep Qualitative Studies: There is a need to conduct in-depth ethnographic and interview-based research to carefully unpack how micro-cultural factors within communities (such as the sense of identity safety, the prevalence and impact of microaggressions, and implicit biases) interact with structural VIM interventions, informing more nuanced and culturally sensitive Mechanism designs.
- Compensation & Incentives Experiments: Future research should pilot and rigorously evaluate various financial reward structures—such as direct stipends, integration of inclusion work into performance bonuses, and targeted micro-grants for contributions—to precisely quantify their effects on sustained member participation, individual and team performance, and overall equity outcomes within communities.

6.5. Implications for Maintainers and Organizations

- **Open-Source Maintainers:**
    - Visibility: Actively adopt "VIM-Ready" badges to signal an inclusive environment, consistently curate "good first issues" specifically for newcomers, and make it a practice to spotlight newcomer success stories in release notes and community communications to foster encouragement and role modeling.
    - Invitation: Offer regular, accessible office hours for Q&A, establish peer-mentorship circles to provide support, and implement automated welcome bots to streamline initial interactions and significantly reduce onboarding friction for new members.
    - Mechanism: Clearly articulate and consistently enforce an accessible Code of Conduct; further, appoint an independent ombudsperson or a neutral third-party for conflict resolution and transparently share summaries of incident resolutions to build and maintain community trust.
- **Corporate Stakeholders:**

    - Performance & Compensation: Formally integrate inclusion-related contributions into employee performance evaluations—awarding tangible career credit, corresponding merit increases, and financial bonuses for activities such as workshop leadership, active mentorship, and valuable open-source software contributions. This signals that such work is valued.
    - Resource Allocation: Actively sponsor multi-tenant inclusion councils that span across departments or even companies, provide necessary resources like cloud credits or hardware stipends for contributors, and fund cross-company sprints or hackathons focused on inclusion to effectively lower financial and logistical barriers to participation.
    - Transparency & Accountability: Commit to publishing quarterly inclusion dashboards that benchmark VIM-aligned metrics (e.g., diversity in leadership, contributor engagement from underrepresented groups) against peer organizations or industry standards to foster healthy competition, shared accountability, and collective learning across the ecosystem.
- **Academic & Nonprofit Partners:**
    - Curriculum Integration: Proactively embed open-source contribution projects, the use of inclusion toolkits, and active participation in community sprints directly into Data Science and AI courses, as well as capstone projects, to strategically align academic skill acquisition with real-world impact and collaborative practices.
    - Joint Programming: Collaborate with community-focused nonprofits (e.g., PyLadies, R-Ladies, Black in AI, and similar organizations) to co-market and co-host inclusion sprints, workshops, and events, thereby maximizing outreach to diverse talent pools and optimizing the sharing of valuable resources.
    - Funding & Fellowship: Establish and offer targeted scholarships and internship fellowships that are explicitly tied to contributions to open-source software and demonstrated leadership in inclusion initiatives, thereby directly linking academic credentials and opportunities with meaningful community engagement and impact.

7. Call to Action

The challenge posed by the “Missing 78%” is not an insurmountable or permanent void but rather a compelling call to action—an opportunity to mobilize our collective expertise, supportive structures, and sustained willpower to effect lasting change. We invite all stakeholders—individuals, communities, corporations, and institutions—to actively participate in this transformation:

- **Adopt & Contribute to the Toolkit:** Fork the Open Inclusion Toolkit available on GitHub, thoughtfully tailor it to your specific organizational or community context, and importantly, share back any enhancements made—such as new survey instruments, improved onboarding guides, or refinements to codes of conduct—to benefit the wider ecosystem.
- **Display the “VIM-Ready” Badge:** By displaying this badge, signal clearly to diverse contributors that your project or organization demonstrably meets established inclusive standards across visibility, invitation, and mechanism pillars, doing so in a transparent and verifiable manner that builds confidence.
- **Join the Inclusion Council:** Become an active part of a rotating, cross-company and cross-community consortium. This council will play a key role in governing shared inclusion metrics, coordinating collaborative sprints, and broadly amplifying success stories to inspire wider adoption of best practices.

- **Host or Participate in Sprints:** Commit to organizing or participating in regular quarterly inclusion sprints, dedicated mentorship days, and engaging lightning-talk events—these can be held either within your own community or in broader collaboration with other organizations to maximize impact and knowledge sharing.
- **Publish Your Metrics & Case Studies:** Transparently report your organization's or community's inclusion scorecards and provide detailed, honest reflections on what strategies worked, what didn't, and the lessons learned along the way, thereby enriching the shared evidence base for effective inclusion practices.

Together, by systematically and thoughtfully weaving the principles of Visibility, Invitation, and Mechanism (VIM) into every layer of the Data, AI, and open-source ecosystems—from the daily practices of individual contributors to the strategic imperatives of global corporations—we can collectively strive to turn the "Missing 78%" not just into a closed gap, but into a historic and sustained overrepresentation of diverse, empowered innovators who are driving ethical, robust, and equitable technologies forward for the benefit of all.

References

[1] Boston Consulting Group, "What's keeping women out of data science?," 2020. [Online]. Available: https://www.bcg.com/publications/2020/what-keeps-women-out-data-science

[2] D. Rossi and S. Zacchiroli, "Worldwide gender differences in public code contributions," *arXiv*, 2022, [Online]. Available: https://arxiv.org/abs/2202.07278

[3] Idaho State University, "1 in 20 Programmers on Open-Source Software Projects Are Women, ISU Study Says." [Online]. Available: https://localnews8.com/isu/2023/06/24/1-in-20-programmers-on-open-source-software-projects-are-women-isu-study-says

[4] The Alan Turing Institute, "Women in Data Science and AI." [Online]. Available: https://www.turing.ac.uk/research/research-programmes/public-policy/public-policy-themes/women-data-science-and-ai

[5] R. Lorenzo, N. Voigt, M. Tsusaka, M. Krentz, and K. Abouzahr, "How Diverse Leadership Teams Boost Innovation," 2018. [Online]. Available: https://www.bcg.com/publications/2018/how-diverse-leadership-teams-boost-innovation

[6] C. Díaz-García, Á. González-Moreno, and F. J. Sáez-Martínez, "Gender Diversity within R&D Teams: Its Impact on Radicalness of Innovation," *Innovation: Management, Policy & Practice*, vol. 15, no. 2, pp. 149–160, 2013, doi: 10.5172/impp.2013.15.2.149.

[7] M. M. Fuentes-Fuentes, A. M. Bojica, and M. Ruiz-Arroyo, "Gender Diversity, Inclusive Innovation and Firm Performance," *Sustainable Development*, vol. 31, no. 5, pp. 3622–3638, 2023, doi: 10.1002/sd.2615.

[8] G. Desvaux, S. Devillard, and S. Sancier-Sultan, "Women Matter: Gender Diversity, a Corporate Performance Driver," 2007. [Online]. Available: https://www.mckinsey.com/capabilities/people-and-organizational-performance/our-insights/gender-diversity-a-corporate-performance-driver

[9] B. Trinkenreich *et al.*, "Pots of Gold at the End of the Rainbow: What is Success for Open Source Contributors?," *IEEE Transactions on Software Engineering*, vol. 48, no. 10, pp. 3940–3953, 2022, doi: 10.1109/TSE.2021.3108032.

[10] Harnham, "Diversity in Data Report," 2023. [Online]. Available: https://www.harnham.com/did-deep-dive

[11] UNESCO, "G20: UNESCO Calls for Greater Gender Equality in Science." [Online]. Available: https://www.unesco.org/en/articles/g20-unesco-calls-greater-gender-equality-science

[12] "Gender gap in research publishing is improving — slowly," *Nature*, 2025, doi: 10.1038/d41586-025-00553-x.

[13] T. C. Dennehy and N. Dasgupta, "Female Peer Mentors Early in College Increase Women's Positive Academic Experiences and Retention in Engineering," *Proceedings of the National Academy of Sciences*, vol. 114, no. 23, pp. 5964–5969, 2017, doi: 10.1073/pnas.1613117114.

[14] J. Terrell *et al.*, "Gender Bias in Open Source: Pull Request Acceptance of Women Versus Men," *PeerJ Computer Science*, vol. 3, p. e111, 2017, doi: 10.7717/peerj-cs.111.

[15] S. Sultana, A. Turzo, and A. Bosu, "Code Reviews in Open Source Projects: How Do Gender Biases Affect Participation and Outcomes?," *arXiv*, 2022, [Online]. Available: https://arxiv.org/abs/2210.00139

[16] WomenTech Network, "Women in Tech Stats 2025." [Online]. Available: https://www.womentech.net/women-in-tech-stats

[17] S. Gammage, "Gender Equality and the Costs of Unpaid Caregiving," *IMF Finance & Development*, 2019, [Online]. Available: https://www.imf.org/en/Publications/fandd/issues/2019/03/gender-equality-and-costs-of-unpaid-caregiving-gammage

[18] Built In, “AI Bias: What It Is and How to Prevent It.” [Online]. Available: https://builtin.com/artificial-intelligence/ai-bias

[19] A. Maddi and Y. Gingras, “Gender Diversity in Research Teams and Citation Impact in Economics and Management,” *arXiv*, 2020, [Online]. Available: https://arxiv.org/abs/2011.14823

[20] S. Brooke, “Programmed Differently? Testing for Gender Differences in Python Code Quality,” *Journal of ComputerMediated Communication*, vol. 29, no. 1, p. zmad49, 2023, doi: 10.1093/jcmc/zmad049.

[21] V. Hunt, S. Prince, S. DixonFyle, and L. Yee, “Delivering Through Diversity,” 2018. [Online]. Available: https://www.mckinsey.com/featured-insights/diversity-and-inclusion/delivering-through-diversity

[22] “The Tipping Point: Women on Boards and Financial Performance,” 2016. [Online]. Available: https://www.msci.com/www/research-paper/the-tipping-point-women-on/0474298723

[23] PyData, “Diversity Scholarships.” [Online]. Available: https://pydata.org/diversity-inclusion/

[24] CIO Magazine, “Ada Developers Academy: A No-Cost Career Program that Aims to Diversify IT.” [Online]. Available: https://www.cio.com/article/308608/ada-developers-academy-a-no-cost-career-program-that-aims-to-diversify-it.html

[25] Outreachy, “Internships Supporting Diversity in Tech.” [Online]. Available: https://www.outreachy.org/

[26] “Evaluation Report: TechWomen,” 2020. [Online]. Available: https://eca.state.gov/files/bureau/final_report__v2_techwomen140d0420f0211-for_web.pdf

[27] Organisation for Economic Co-operation and Development, “Unpaid Care Work: The Missing Link in Gender Equality,” 2020. doi: 10.1787/1f3fd03f-en.

Open Access | https://doi.org/10.25080/xydg9873

SciPy 2025

July 7 - July 13, 2025

Proceedings of the 24th Python in Science Conference
ISSN: 2575-9752

Enhancing Curve Fitting with SciPy: Interactive Spline Modeling and Reproducibility with SplineCloud

Vadym Pasko[1,2]

[1]SplineCloud, [2]Kyiv Aviation Institute

Abstract

Curve fitting is a fundamental task in data science, engineering, and scientific computing, enabling researchers to extract meaningful relationships from data. However, selecting and tuning the right fitting model for complex, noisy, or multidimensional data remains a significant challenge. SciPy plays a critical role in addressing these challenges by providing robust spline fitting methods that offer flexibility and precision. Yet, fine-tuning spline parameters, ensuring stability in extrapolation, and sharing fitted models for reproducibility remain open problems.

In order to address these challenges, we developed SplineCloud - an open platform that provides interactive spline fitting capabilities and uses SciPy on the backend. SplineCloud allows the construction, analysis, and exchange of spline-based regression models using SciPy's `interpolate` module. SplineCloud's curve fitting tool extends the capabilities of SciPy spline fitting methods by enabling researchers to fine-tune spline parameters: knot vector, control points, and degree interactively, and providing the ability to quickly analyze the accuracy of the models. Models constructed on the platform obtain their unique identifiers and become reusable in code, fostering better collaboration and knowledge transfer. Reusability of spline curves and underlying datasets in code is enabled via the open-source SplineCloud client library called `splinecloud-scipy`, which is also based on SciPy. The proposed approach of interactive cloud-based fitting improves data processing workflows, allows separating data preparation and approximation routines from the main code, and brings FAIR principles to the curve fitting, enabling researchers to construct and share libraries of empirical data relations.

Keywords curve fitting, spline fitting, FAIR, empyrical relations, data analysis, regression models

Published Jul 10, 2025

Correspondence to
Vadym Pasko
vadym@splinecloud.com

Open Access

Copyright © 2025 Pasko. This is an open-access article distributed under the terms of the Creative Commons Attribution 4.0 International license, which enables reusers to distribute, remix, adapt, and build upon the material in any medium or format, so long as attribution is given to the creator.

1. Introduction

The adoption of FAIR data principles [1] has fundamentally transformed research practices across scientific disciplines. Initially designed to enhance research data management and increase transparency, these principles now extend to a broad range of digital scientific artifacts, including computational models, source code, 3D models, and other digital objects. Widespread adoption of FAIR principles fosters improved reproducibility and facilitates the integration and extension of scientific results. However, not all digital objects achieve the same level of FAIRness, and many platforms lack support for the diverse range of data formats and objects used in science and engineering.

This paper focuses on enhancing reusability — and thereby reproducibility — in the context of regression modeling, a technique used to mathematically describe the relationship between dependent and independent variables. This process, commonly known as curve

fitting, is widely employed in data science, engineering, and scientific computing, particularly for processing experimental data and extracting interpretable models for further analysis. Of particular practical interest is spline fitting, which effectively models complex empirical relationships without explicit underlying mathematical laws. A formal overview of splines and spline fitting is provided in Some Theoretical Background Behind Splines and Spline Fitting With SciPy.

SciPy's `interpolate` module offers extensive functionality for constructing splines, from simple interpolating curves to advanced smoothing splines and parametric representations. Its spline fitting methods provide robust algorithms widely used to model complex, noisy, or multidimensional data dependencies. Nonetheless, the capabilities of splines cannot be fully leveraged through purely programmatic interfaces. In contrast, modern graphical modeling software with interactive spline manipulation affords a level of control and intuitive adjustment not achievable by traditional automatic fitting methods. This, together with a problem of reusability of obtained models, their accessibility, and interoperability, leaves a way for thorough considerations towards alternative approaches to the curve fitting process.

These considerations eventually led to the development of a dedicated platform for interactive spline fitting, which incorporates FAIR principles and adapts them to the regression models. Built upon SciPy's routines, SplineCloud enables collaborative, transparent, and reusable curve fitting[1]. Sections 4 through 8 of this paper detail the similarities and distinctions between SciPy's and SplineCloud's approaches, illustrating a novel paradigm for collaborative data fitting.

2. Some Theoretical Background Behind Splines

Splines are piecewise-defined functions used extensively in numerical analysis, computer-aided geometric design, and data fitting. The fundamental idea behind the spline fitting process is to construct a smooth function that matches a set of data points or satisfies a set of constraints, while preserving computational efficiency and numerical stability. Splines can be represented in several forms, each suitable for different applications: Piecewise Polynomial, Hermite, Bézier, B-spline, NURBS (Non-Uniform Rational B-Splines). Amongst these forms, a special place takes B-Spline representation - where the spline function (or curve) is expressed as a linear combination of basis functions.

A distinction must be made between spline functions and parametric spline curves:

- **A spline function** is a scalar-valued function $S(x)$, defined over a single independent variable x, typically used in interpolation or regression of scalar data.
- **A parametric spline curve** defines a vector-valued mapping from a scalar parameter t to a multidimensional space:

$$\boldsymbol{C}(t) = (x(t), y(t), z(t), ...) \tag{1}$$

where each coordinate function $x(t), y(t), ...$ is a spline function in t. This is the common representation in geometric modeling and computer graphics.

A particularly efficient and widely used spline representation is the B-spline (basis spline). B-splines are defined via a set of control points $\boldsymbol{P}_i$ and a knot vector t_i, and provide local control, numerical stability, and efficient evaluation. A B-spline curve of degree k is defined as:

$$\boldsymbol{C}(t) = \sum_{i=0}^{n} \boldsymbol{P}_i B_{i,k}(t) \tag{2}$$

[1]First described in Mastering Curve Fitting. How to Fit Complex Data with Parametric Splines

where $\boldsymbol{P}_i$ are control points, $B_{i,k}(t)$ are the B-spline basis functions of degree k, defined recursively by the Cox–de Boor formula [2], [3]:

$$B_{i,0}(t) = \tag{3}$$

$$B_{i,k}(t) = \frac{t - t_i}{t_{i+k} - t_i} B_{i,k-1}(t) + \frac{t_{i+k+1} - t}{t_{i+k+1} - t_{i+1}} B_{i+1,k-1}(t) \tag{4}$$

In (4) possible divisions by zero are resolved by the convention that 'anything divided by zero is zero' [4]. This formulation separates geometry (control points) from basis functions, allowing flexible manipulation and efficient computation of spline curves.

Parametric splines are widely used in computer graphics, computer-aided design (CAD), and more general geometric modeling, where the goal is to design and manipulate accurate and smooth curves and surfaces in two or three dimensions. Their unique properties allow for high shape flexibility, local control over curve shape and its continuity. An intuitive visual explanation of splines, different forms of their representation and unique properties is given in the video by Freya Holmér [5].

However, despite their geometric advantages, parametric splines are rarely used in statistical data fitting and regression analysis. The primary limitation is the parameterization problem: for observed data points (x_i, y_i), there is no universally optimal method to assign parameter values t_i that correspond meaningfully to the underlying data relationship. While heuristic approaches such as chord-length or centripetal parameterization are commonly used, the choice can significantly influence the quality of the resulting fit [6]. Additionally, parametric splines do not provide the direct functional relationship $y = f(x)$ that is typically required for prediction and statistical inference in data analysis contexts. This is why spline functions are often a more popular choice.

3. Spline Fitting With SciPy

SciPy provides a robust and flexible set of spline fitting tools for both **interpolation** and **approximation** through the `scipy.interpolate` module. In terms of univariate fitting, SciPy supports several spline fitting methods, including:

Interpolating splines (Figure 1 a) - exact fit to the data. This constructs a spline $S(x)$ such that:

$$S(x_i) = y_i \quad \text{for all } i \tag{5}$$

and ensures continuity of first and second derivatives (C^2 continuity for cubic splines).

Smoothing splines (Figure 1 b) - approximate fit with smoothness penalty. The smoothing spline minimizes the penalized least-squares objective:

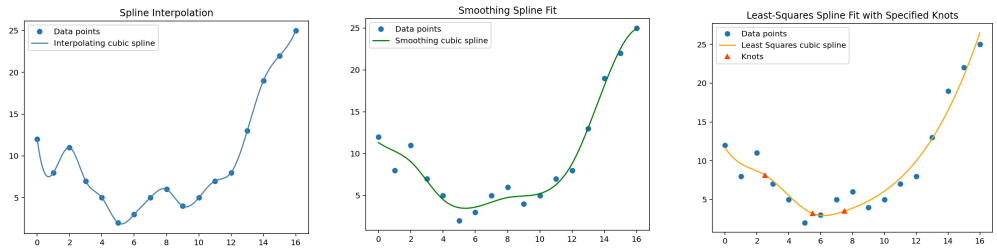
$$\min_S \left\{ \sum_{i=1}^{n} (y_i - S(x_i))^2 + \lambda \int_a^b (S''(x))^2 dx \right\} \tag{6}$$

where λ is a regularization parameter related to s.

Least squares splines (Figure 1 c) - approximate fit with squared residual minimization penalty:

$$\min_S \sum_{i=1}^{n} (y_i - S(x_i))^2 \tag{7}$$

Parametric splines. Used to fit looped curve, isolines, or 3D curves.



(a) Simple interpolating cubic spline. **(b) Smoothing cubic spline.** **(c) Least-Squares cubic spline.**

Figure 1. *Spline fitting results with a use of SciPy's* `interpolate` *methods.*

3.1. Pain Points of Pure Programmable Spline Fitting

SciPy provides a programmatic interface to robust spline fitting methods suitable for a variety of tasks, particularly when working with complex experimental or statistical data. However, despite offering multiple fitting methods and adjustable parameters, selecting an appropriate combination can be challenging. As noted by [7], issues such as overfitting and extrapolation control lack straightforward solutions and often require multiple iterations, either with visual inspection or advanced scripting.

The mathematical formulation of B-splines—especially parametric B-splines and NURBS—enables fine control over curve shapes through manipulation of control points and knot vectors. While this functionality is widely leveraged in computer graphics and CAD applications, it is not supported in SciPy.

Introducing interactive control over curve geometry can improve fitting accuracy and reduce the time spent tuning parameters in search of acceptable results from automatic fitting routines.

Another limitation is the lack of portability and reproducibility of fitted models. SciPy spline objects are tightly coupled to the Python runtime and local environment. Although spline parameters (order, knot vector, and coefficients) can be exported, reconstructing models from these components requires an understanding of B-spline structure. Model serialization is also possible, but may not be practical in all scenarios.

These limitations hinder reproducibility and lead to duplicated effort, where researchers repeatedly perform the same fitting procedures on shared datasets, reimplementing logic and parameters independently. Even when code and data are shared, fitted models often remain environment-specific and are difficult to reuse across projects.

Overcoming these challenges requires workflows in which spline models can be edited interactively, stored independently from source code and raw data, and reused across tools and teams.

These needs motivated the development of SplineCloud, a platform that extends SciPy's spline fitting with interactivity, transparency, and full model lifecycle support.

4. Introducing SplineCloud

SplineCloud is an open platform for formalized knowledge exchange. It is designed to make technical data, like results of simulations, tests and modeling, more accessible and reusable in computer code. Its architecture and instruments help reduce the routine overhead of data collection, processing, and model reconstruction in code.

SplineCloud is free to use for everyone, publishing open data, however, its source code is closed (as of now). Repositories with limited access are a part of the platform's monetization model. All tools are free to use without limitations (except repository size, which as of now is limited to 3 GB). The platform's main features are: plot digitizer, advanced spline fitting tool, open API, and client libraries for Python and MATLAB (libraries for other languages are planned for future implementation).

The philosophy behind the platform is in representing data relations in the form of spline models, which can be reused in code. In this way, SplineCloud can be considered as a repository-based open library of functional relations. The typical user journey on the platform is given in Figure 2.

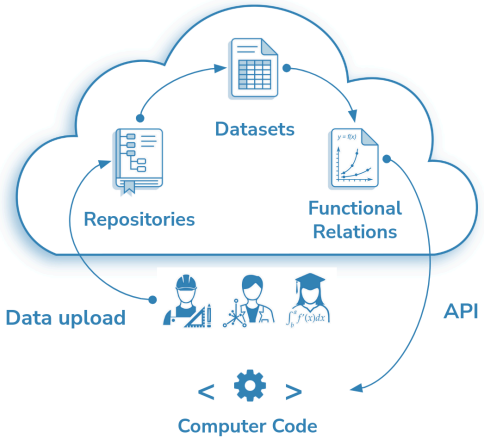


Figure 2. *SplineCloud workflow scheme: upload data to repository, define datasets, create data relations using spline fitting tool, share and access curves and datasets in code via API.*

The platform's data structure is given in Figure 3. Each repository may contain multiple data files (uploaded by authors), which serve as sources for datasets; authors and other users can define subsets (clean data) related to datasets. Subsets are used for the construction of data relations, which capture relationships between columns in subsets. Parametric spline curves are used to model these relations.

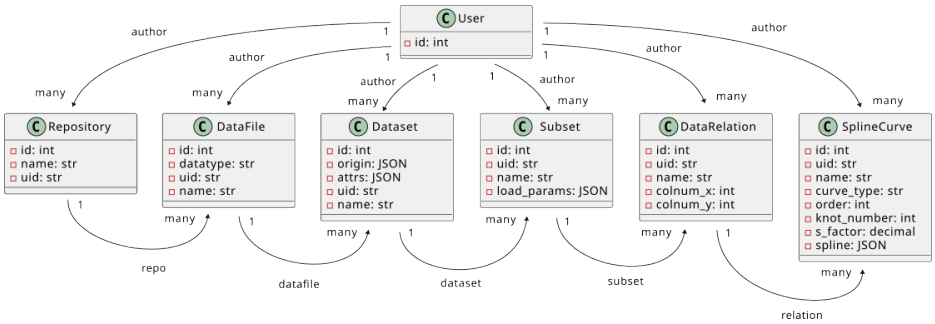


Figure 3. *SplineCloud objects relationship diagram*

Spline fitting tool, a main instrument of the platform, is built on top of SciPy's interpolate module and provides a visual interface to three main spline fitting methods described in Interactive Spline Fitting Workflow: Interpolating Splines, Smoothing Splines, Least Squares Splines.

The web interface of the spline fitting tool is powered by D3.js and JavaScript build of verb-nurbs - an open-source library for NURBS modeling. The RESTful API and an open-source Python client library (splinecloud-scipy) allow the reuse of data objects and spline curves

in code. In this way, SplineCloud addresses the key limitations of traditional spline fitting workflows by offering the following core capabilities:

- Interactive Spline Modeling;
- Reusability and Interoperability of Models;
- Model Traceability.

5. Interactive Spline Fitting Workflow

Interactivity in the curve fitting process significantly simplifies and accelerates model construction. It enables users to identify overfitting, discontinuities, or extrapolation issues early, and make real-time adjustments. This is especially valuable in parametric spline fitting, where fine-tuning the curve shape and knot configuration often requires iterative, visual feedback. In this section, we will take a look at how these capabilities are implemented in SplineCloud and how the interactive curve fitting approach can be complementary to the programmatic data processing workflows.

5.1. Data Preparation

According to the workflow presented in Figure 2, data has to be uploaded to the existing or new repository. It can be a text file, a spreadsheet, or an image containing a plot. In a case of text data, a default dataset will be created automatically and a subset of data can be identified by adjusting data loading options. Datasets will be created automatically for each sheet in the spreadsheet source file. An interactive plot digitizer tool will be displayed for the image file to help extract data ranges from plots (Figure 4).

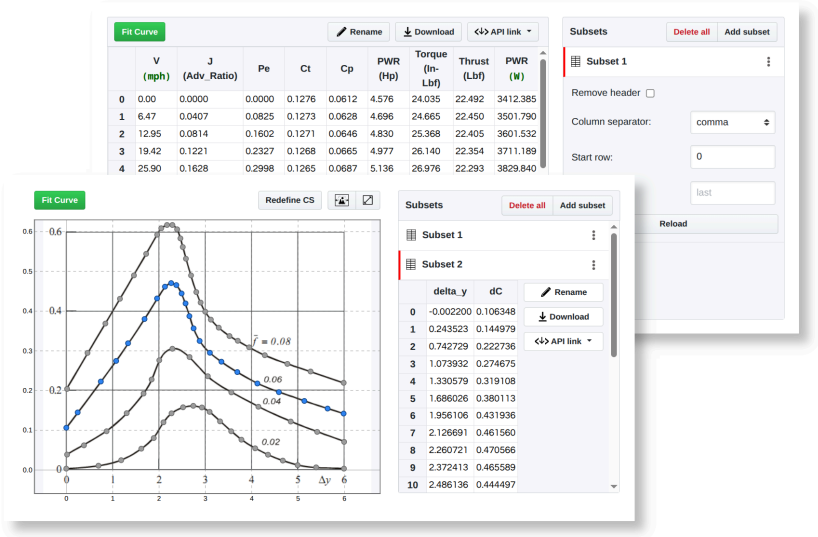


Figure 4. *SplineCloud datasets extracted from the tabular data file and plots*

After identifying clean subsets, a default Data Relation object will be created after entering a curve fitting mode (either by clicking the Fit Curve button or going into the Relations tab).

5.2. Automatic Spline Fitting

By adding a first curve, an initial spline fit is generated using a smoothing splines method (Figure 5).

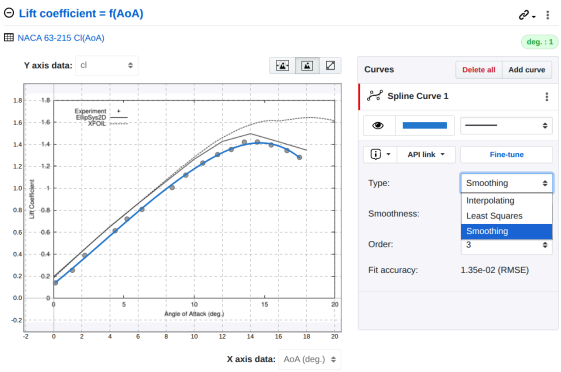


Figure 5. *Spline fitting tool*

SplineCloud's smoothness parameter is a relative parameter used to calculate SciPy's (FITPACK's) smoothing factor s defined as:

$$\sum_{i=1}^{n} w_i(y_i - S(x_i))^2 \leq s \tag{8}$$

The transition from the relative smoothing parameter to SciPy's s-factor is implemented in three steps:

1. building the least squares fit with the minimal possible number of knots (zero internal knots);
2. calculating the actual smoothing factor s_{max} for this fit using formula (8);
3. multiplying the relative smoothing parameter by this value: $s_{\text{scipy}} = s_{\max} \cdot s_{\text{rel}}$

This approach improves user experience - instead of guessing each time the correct absolute value (which depends on the scale of data points), it is more intuitive to use relative values. By selecting several values for one data range the developed feedback instructs the more appropriate values for another curve of a different scale.

Least Squares fitting is implemented as an alternative to smoothing splines and is built on top of SciPy's `LSQUnivariateSpline` class. For simplicity, a uniform knot vector is constructed and passed to the class constructor using the number of internal knots from the user input. However, there is an option to adjust the knot vector interactively and use least squares fitting for the given non-uniform knot vector. This capability is implemented in the fine-tuning mode (see Section 4).

SplineCloud also supports interpolating splines, implemented via hardcoded $s = 0$ in `UnivariateSpline` class constructor.

5.3. Fine-Tuning. Interactive Adjustments of Control Points and Knot Vector

A principal enhancement over the automatic fitting approach is SplineCloud's Fine-Tuning functionality. It provides the ability to visually adjust control points and knot vectors of fitted splines. As proved in many cases, this interface enables users to achieve curve refinements that exceed the capabilities of SciPy's automatic fitting algorithms in terms of smoothness and accuracy. Moreover, the visual interface to knot vectors provides control over curve continuity in the specific regions. This helps in modelling complex data behavior with steep changes in main trends.

In SplineCloud, all spline curves are represented as parametric B-splines or NURBS. As mentioned in Some Theoretical Background Behind Splines, a parametric B-spline curve is defined as a vector-valued function (2). This function, however, can be decomposed into its scalar components:

$$S^{(x)}(t) = \sum_{i=0}^{n} c_i^{(x)} B_{i,k}(t)$$
$$S^{(y)}(t) = \sum_{i=0}^{n} c_i^{(y)} B_{i,k}(t) \tag{9}$$

In simpler words, parametric spline curves can be defined by two distinct univariate spline functions, $x(t)$ and $y(t)$, sharing a common knot vector (Figure 6). This representation allows for the modeling of complex geometries, including loops, sharp transitions, and high-curvature regions.

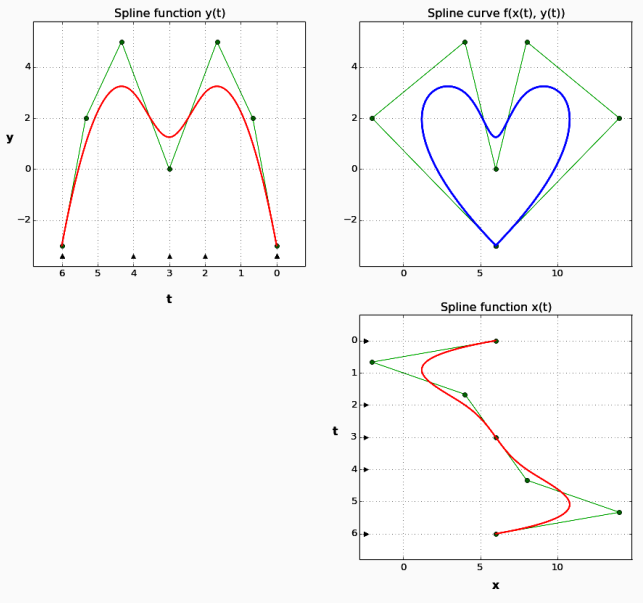


Figure 6. *Parametric spline curve as a combination of two spline functions.* Black triangles along parameter axes represent knots. Control polygons and control points are displayed in green color

5.3.1. *Interactive Editing of Control Points*

The beauty of parametric splines is in the ability to have an intuitive control over the curve shape by modifying the control polygon: moving, adding, and removing control points, increasing or decreasing their weights. This is impossible for spline functions, since they are defined through scalar coefficients, not vectors.

However, it is important to mention that for parametric spline curves coordinates of the control points $P_i = [x_i, y_i]$ are mathematically equivalent to the spline coefficients for the component functions $S^{(x)}(t)$ and $S^{(y)}(t)$, so that:

$$\boldsymbol{C}(t) = \sum_{i=0}^{n} \boldsymbol{P}_i \, B_{i,k}(t), \quad \text{where } \boldsymbol{P}_i = \begin{bmatrix} c_i^{(x)} \\ c_i^{(y)} \end{bmatrix} \tag{10}$$

Due to the local support property of B-spline basis functions, moving a single control point influences the shape of the curve only within a limited range of the parameter domain, providing localized editing capabilities.

In the Fine-Tune mode, users can interactively drag control points to reshape the curve (Figure 7). This mirrors the behavior of spline modeling in professional CAD environments, where designers sculpt geometry directly. The influence of each control point depends not only on its position but also on the associated basis function and, optionally, its weight.

Assigning higher weights to a control point increases its pull on the curve, bringing it closer to the point's location, converting the B-Spline to NURBS.

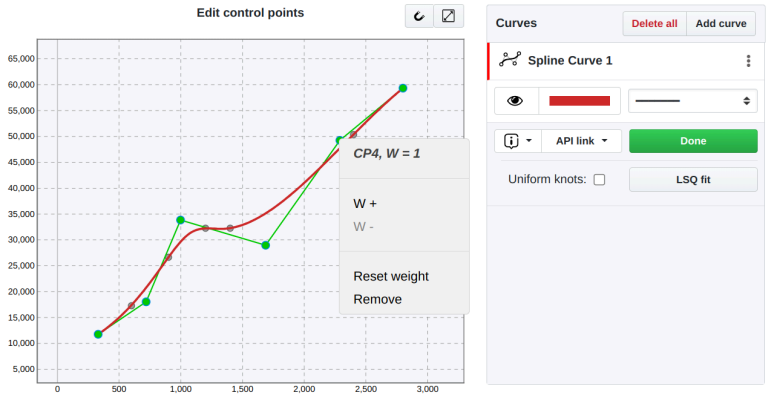


Figure 7. *Adjusting control points of the spline curve in the Fine-Tune mode*

This form of user interaction provides precise control over the spline's shape and is particularly advantageous when fitting complex datasets where automated routines produce unsatisfactory results.

5.3.2. *Dynamic Knot Vector Adjustment*

The knot vector plays a central role in determining the structure and properties of the resulting spline curve. Given a spline of degree k, the knot vector $\{t_0, t_1, ..., t_m\}$ is a non-decreasing sequence of real numbers, typically ranging from 0 to 1 (but not necessarily). Each interval $[t_i, t_{i+k+1})$ corresponds to a region over which a particular B-spline basis function $B_{i,k}(t)$ has support, meaning that each basis function is non-zero over at most $k + 1$ knot spans. The location and multiplicity of knots affect several critical properties of the resulting spline:

- **Continuity**. The number of continuous derivatives at a knot t_i is $k - m_i$, where m_i is the multiplicity of that knot. Repeated knots reduce the smoothness of the spline at that knot location. Specifically, if a knot has multiplicity m, then the continuity of the spline at that knot is reduced to C^{k-m}, where k is the degree of the spline. That is, the spline remains $(k - m)$-times continuously differentiable, and all higher derivatives are discontinuous.
- **Flexibility**. Adding more knots increases the local adaptability of the spline, allowing it to better follow variations in the data. In the B-spline formulation, the number of basis functions, and therefore the number of control points n is always $T - k - 1$, where k is the spline degree, T is the number of knots. As more interior knots are introduced, the number of basis functions increases, providing additional degrees of freedom for shaping the curve while maintaining the required continuity.

In traditional fitting methods such as `LSQUnivariateSpline` in SciPy, the knot vector must be either provided manually or generated heuristically, which typically requires some kind of iterative approach in finding an optimal (usually quasi-optimal) knot vector. SplineCloud removes this complexity by exposing the knot vector as an editable structure in the Fine-Tune mode, where users can manipulate knots directly and observe their impact on the spline in real time (Figure 8).

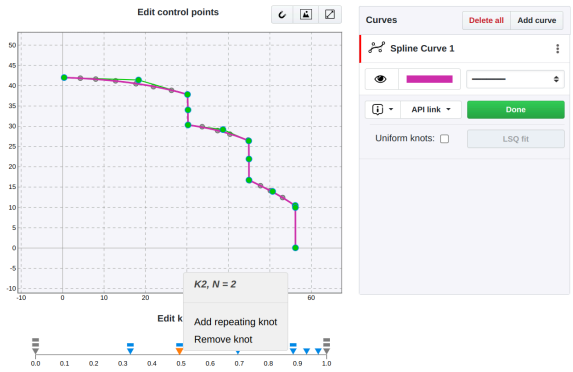


Figure 8. *Adjusting the knot vector of the spline curve in the Fine-Tune mode*

In particular, SplineCloud enables users to insert and remove knots, relocate internal knots, add and remove duplicate knots to control continuity and introduce geometric features such as cusps, kinks, or plateaus.

Uniformly spaced knot vectors typically produce curves with more predictable and symmetric behavior, which is easier to interpret visually and adjust interactively. In contrast, non-uniform vectors can help in fitting local irregularities with higher precision.

5.4. Fitting Errors

Comparison of the different fitted models requires a well-defined error metric that quantifies the discrepancy between the predicted and actual data. Out of many different metrics it is worth to mention the most common: Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE).

$$\mathrm{MAE} = \frac{1}{n}\sum_{i=1}^{n} \mid y_i - S(x_i) \mid \tag{11}$$

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n} (y_i - S(x_i))^2 \tag{12}$$

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n} (y_i - S(x_i))^2} \tag{13}$$

where y_i are the actual data values, $S(x_i)$ are the corresponding predicted values from the curve, n is the total number of data points.

5.4.1. *Estimating Fitting Errors in SciPy*

In SciPy's spline fitting, smoothing splines support weighted data points, with fitting errors estimated using the corresponding weighted residuals. If weights w_i are provided (or implicitly set to 1), the weighted squared residuals are computed as:

$$R_i = (w_i(y_i - S(x_i)))^2 \tag{14}$$

The smoothing condition is formulated as:

$$\sum_{i=1}^{n} R_i \leq s \tag{15}$$

where s is the smoothing factor. This formulation is used internally by functions such as `UnivariateSpline` and `splrep`, where the goal is to balance curve fidelity with smoothness.

The `UnivariateSpline` class includes a method called `get_residual()`, which returns the weighted sum of squared residuals used in the spline construction process. Here is an ex-

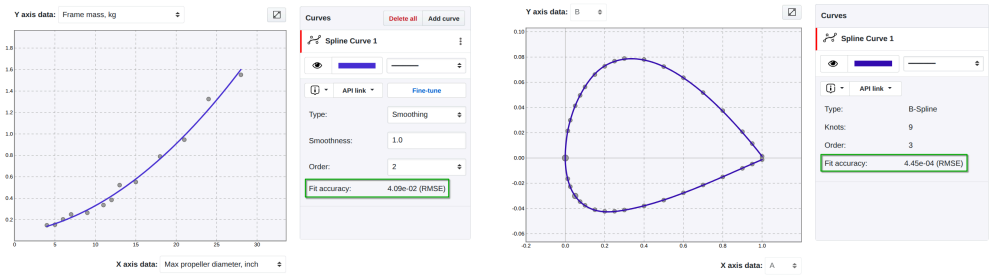
ample of estimating the fitting error using the built-in `get_residual()` and custom estimator function:

```python
import numpy as np
from scipy.interpolate import UnivariateSpline

# Generate synthetic data
x = np.linspace(0, 10, 100)
y = np.sin(x) + 0.2 * np.random.randn(100)

# Fit smoothing spline
spline = UnivariateSpline(x, y, s=5.0)

# Evaluate spline at input points
y_fit = spline(x)

# RMSE using SciPy's get_residual()
residual = spline.get_residual()
rmse_scipy = np.sqrt(residual / len(x))

# RMSE computed manually
rmse_manual = np.sqrt(np.mean((y - y_fit)**2))

print(f"RMSE (SciPy):  {rmse_scipy:.5f}")
print(f"RMSE (Manual): {rmse_manual:.5f}")
```

```
RMSE (SciPy):  0.22361
RMSE (Manual): 0.22361
```

5.4.2. *Estimating Fitting Errors in SplineCloud*

In SplineCloud, the RMSE is calculated automatically for all curves. The `Fit accuracy` hint updates after each curve modification (Figure 9 a, b). This allows tracking the change of fitting error while applying different fitting parameters, comparing different models and fine-tuned curves against automatically fitted models.



(a) RMSE estimation in SplineCloud.

(b) RMSE evaluated via shortest distance residuals.

Figure 9. *Simple interpolaing cubic spline.*

For complex curve shapes, especially those with closed loops (Figure 9 b), the error between a data point and the curve should not be measured only along the vertical axis. Instead, the true discrepancy is the Euclidean distance from each data point to the nearest point on the curve. This approach is implemented in SplineCloud to estimate fitting errors of parametric curves that approximate data in non-ascending order via $RMSE_{sd}$ - root mean squared shortest distance error:

$$\mathrm{RMSE}_{sd} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(w_i d_i)^2} \tag{16}$$

where d_i is the shortest distance between a data point and the spline curve. Evaluation of this distance is implemented via SciPy's `optimize` module, and particularly a direct method is used as one of the most stable and fastest global optimization algorithms (based on the in-house analyses).

6. Reusability and Reproducibility with SplineCloud

Reusability and reproducibility are closely related concepts in scientific research, both essential for ensuring that data-driven results can be reliably verified, extended, and applied across different contexts and by independent researchers.

Reusability refers to the capacity of data, models, or computational results to be used beyond their original context, either by the same researcher at a later time or by others pursuing related work.

Reproducibility denotes the ability of independent researchers to replicate the results of a study using the same input data, methods, and conditions. In computational science, this typically involves the complete transparency of the modeling pipeline, including data preprocessing, parameter tuning, and evaluation metrics.

In conventional curve fitting workflows implemented via a programmable interface, fitted models are often tightly coupled with the original data and code used to generate them. While the source data and fitting scripts may be published, the resulting models themselves are rarely stored or shared as independently reusable objects. As a result, anyone wishing to replicate or build upon a previous curve fitting problem must re-execute the entire fitting process, including selecting the method, tuning parameters, and validating the fit.

This leads to the following limitations:

- **Redundant effort**. Researchers across teams or domains often repeat fitting procedures for the same datasets.
- **Poor reproducibility**. Deviations in fitting parameters and algorithms across various programming environments may cause discrepancies in reproduced models and, as a consequence, discrepancies in modelling results.
- **Poor interoperability**. There is no widely adopted convention for serializing and sharing regression models across tools and environments. As a result, transferring models typically requires manual extraction of fitting parameters or curve construction data (like spline parameters). This process is error-prone and demands significant additional effort to reuse models accurately in code.

By extending the principles of FAIR data to regression models, it becomes clear that it is not sufficient to publish raw data and code alone - intermediate or final research outputs such as fitted models (or response surface models) should also be accessible in standardized, referenceable form.

6.1. Improving Reusability and Reproducibility of Spline Models

SplineCloud addresses these challenges by decoupling the spline model from the fitting code and treating it as a first-class, shareable object. The platform is open for open data and supports persistent storage of fitted curves as structured entities that include:

- Unique IDs and web links;
- API links that return spline data and related datasets and subsets;
- A metadata layer, including authorship, tags, units, and other associated context (improvements in progress).

Once users create spline models interactively, they become instantly accessible to the broad public, including anonymous users. Reusability and reproducibility are ensured by both the open REST API and client libraries.

An official Python client, `splinecloud-scipy`, is based on SciPy and allows for fetching spline data using the spline UIDs and recreating the model in code. Reusable spline model has critical methods for evaluating spline in the form $y = f(x)$, loading underlying data, and assessing fit accuracy (using one of the methods, listed in Section 5). This enables a new level of interoperability: a spline curve fitted by one researcher can be discovered on the platform, imported and evaluated in another researcher's codebase, without downloading and importing the original data and refitting it.

Let's first take a look at the client library basic usage scenarios and then discuss the library structure and how splines are recreated and evaluated.

6.1.1. *Python client*

`splinecloud-scipy` is a library that can be installed from the Python Package Index:

```
pip install splinecloud-scipy
```

As of now, the library provides two main functions: `load_spline()` and `load_subset()`, which should be used to fetch spline models and underlying subsets:

```
from splinecloud_scipy import load_spline
spline = load_spline(<curve_uid>)
```

The curve UID can be taken from SplineCloud - an API link dropdown on the Curves toolbox (Figure 10 a). `load_spline()` returns an instance of the `ParametricUnivariateSpline` class, which allows for evaluating the spline as a function of x.

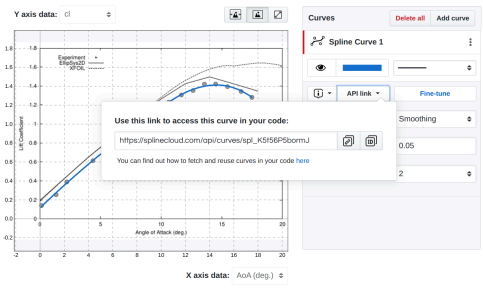
```
import numpy as np
X = np.linspace(0, 20, 100)
Y = spline.eval(X, extrapolate=True)
```

The spline object allows for loading underlying data:

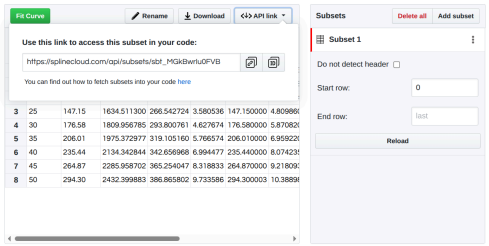
```
columns, table = spline.load_data()
```

The same result can be achieved by explicitly loading a corresponding subset via its UID. Similarly to the spline curve, the subset UID can be taken from the SplineCloud API dropdowns (Figure 10 b). Both methods return a tuple with a list of column names and a NumPy array with subset data.

```
from splinecloud_scipy import load_subset
columns, table = load_subset(<subset_uid>)
```



(a) Accessing the curve API link.



(b) Accessing the subset API link.

Figure 10. *Accessing the curve and subset API links.*

More information on the library usage can be found in the library code repository README section.

The `splinecloud-scipy` library is built around two core classes: `ParametricUnivariateSpline` - for the construction of the parametric splines from the curve data (degree, control points, and knot vector) and `PPolyInvertible` - a helper class defined to enable solving the spline curve as a function of x values. Both classes extend SciPy's classes, namely: `interpolate.UnivariateSpline` and `interpolate.PPoly`.

`ParametricUnivariateSpline` is the main class of the `splinecloud-scipy` client library. Its instance is returned by the `load_spline()` function. It defines a 2D parametric spline curve based on two univariate spline functions — one for the $x(t)$ and one for the $y(t)$ dependencies, that share a common knot vector and spline degree.

The `ParametricUnivariateSpline` object is callable - it takes a parameter value and returns the corresponding x and y values by calling internal objects `self.spline_x()` and `self.spline_y()` - instances of SciPy's `UnivariateSpline` class. This part, however, requires refactoring, since it uses private methods to construct `UnivariateSpline` from the knot vector, coefficients, and degree. There is a plan to use the `BSpline` class instead of `UnivariateSpline` here:

```
self.spline_x = si.UnivariateSpline._from_tck(tck_x)
self.spline_y = si.UnivariateSpline._from_tck(tck_y)
```

Evaluation of the spline in the form $y(x)$ is implemented in the `eval()` method. This method creates a piecewise polynomial representation of spline functions x(t) and y(t) using the custom `PPolyInvertible` class. The methods of this helper class allow for finding the interval containing the *x-value*, then the polynomial is solved for the *t-value*. This parameter value is then used in the spline function $y(t)$ to find the desired *y-value*.

Polynomial representations are constructed inside the private method `_build_ppolyrep()`, called on the `ParametricUnivariateSpline` initialization.

```
def _build_ppolyrep(self):
    self.spline_x.ppoly = PPolyInvertible.from_splinefunc(self.spline_x, extrapolate=True)
    self.spline_y.ppoly = PPolyInvertible.from_splinefunc(self.spline_y, extrapolate=True)
```

The `ParametricUnivariateSpline` class also provides a method for evaluating fit accuracy using one of the three methods discussed in Section 5.

```
RMSE = spline.fit_accuracy(table, method="RMSE")
```

The client library introduces basic capabilities that allow for fetching, recreating, and evaluating spline models in code. Its structure, however, is a bit complicated: conversion to polynomial representation may be omitted in future versions, and other approaches may be used to evaluate the inverse relation $t(x)$ or solve the relation $y(x)$ directly. Contribution for this and other improvements is welcome.

Nevertheless, the implemented approach is covered with tests, supports extrapolation, and shows good performance (no reportable benchmarking was performed).

6.2. Implications and Limitations of SplineCloud

The interactive, collaborative and reusable curve fitting enabled by SplineCloud minimizes redundant work and enhances the reliability and continuity of model-based research outputs. However, the platform currently does not support other classes of regression models such as polynomial fits, rational functions, exponential models, or machine learning-based regressors. This limitation is not technical in nature but a deliberate, temporary design decision, rooted in both theoretical and practical considerations.

Spline models, particularly B-splines and their parametric forms, possess a standardized and well-defined mathematical representation. This makes them ideally suited for platform-independent storage, manipulation, and code-level reuse. In contrast, arbitrary analytical or statistical regression models lack the generalization inherent to splines and the support of all possible modeling techniques is an overwhelming challenge, at least for the current stage of the project.

From a technical point of view, spline-based models are especially effective for representing empirical data derived from physical experiments, simulations, and measurements — scenarios where an accurate and differentiable fit is often more valuable than an interpretable analytical expression. SplineCloud is thus particularly suited for constructing surrogate models, processing and interpolating results of numerical simulations (like wind tunnel tests), and reusing such models in downstream analysis, control, or optimization tasks.

However, users requiring analytical model validation (e.g., fitting custom functions to verify physical laws or derive closed-form expressions) will find SplineCloud insufficient for their needs. In such cases, traditional tools, including SciPy, remain more appropriate.

Despite this limitation, SplineCloud introduces the novel capability of building shareable repositories of empirical relations. The catalogs of such relations can serve as reference data and model sources in experimental domains such as fluid dynamics, structural mechanics, or thermodynamics, where precomputed curves and data-driven models are frequently reused. For example, aerodynamic design processes often rely on aggregated wind tunnel results; SplineCloud provides an infrastructure to formalize, store, and exchange such data in a reproducible and programmatically accessible way.

7. Use Cases and Applications

Among the many possible applications of SplineCloud, three main types of scenarios are especially worth highlighting:

1. situations where interactive curve fitting improves accuracy and/or reduces time spent on analysis;
2. cases where making modeling results accessible and reusable enhances research reproducibility;

3. workflows where empirical data is structured as a library of functions for use in automation.

While the examples presented here are related to engineering, the described patterns and approaches can be applied in other domains.

7.1. Modeling Step-drawdown Water Pumping From Boreholes

Step-drawdown tests are performed to determine the optimal pumping rate for water boreholes by measuring how water levels respond to progressively increasing discharge rates. Traditional analysis methods assume that relationships between pumping rates and water level drop should be predictable and follow simple mathematical models, but experimental data often reveals complex non-linear behaviors that standard equations cannot capture. When conventional models fail to fit the observed data, researchers tend to develop more sophisticated mathematical frameworks.

G. Van Tonder, J. Botha, and J. Van Bosch [8] developed a model for predicting drawdown behavior in step-drawdown tests based on experimental data from South Africa's Karoo formations. This model can be useful in determining sustainable pumping rates for boreholes. While relying on physical properties of fractured rock aquifers, flow dimensions, and elasticity parameters, the model still exhibits underfitting for some pumping rates. Spline fitting can produce significantly better accuracy and does not require a complex analytical background if practical use is the purpose.

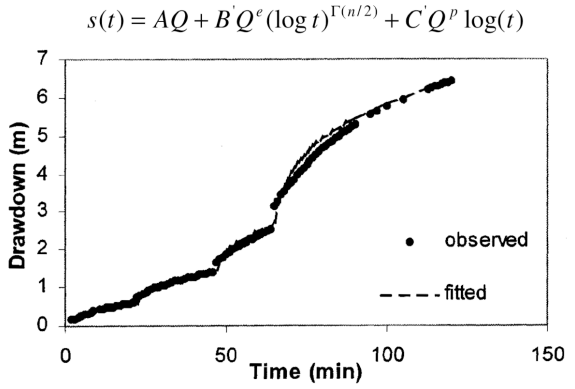
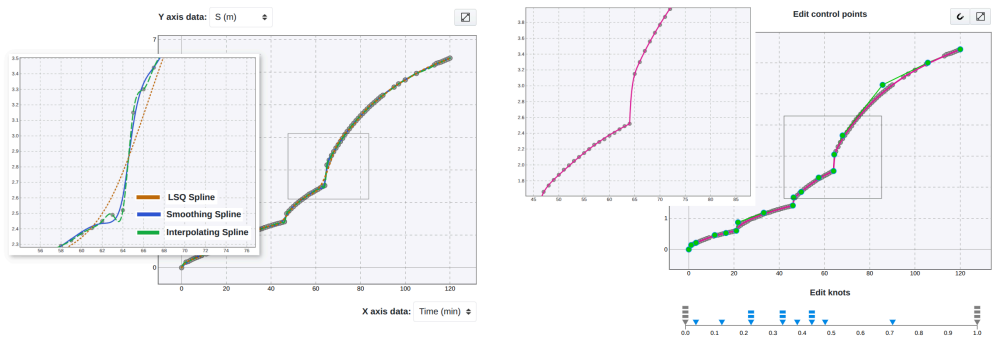


Figure 11. *empirical model fitted to experimental data on the step-drawdown test performed on Borehole UP16 [8]*

Nevertheless, in the case of relations with clearly expressed step changes, fitting becomes a challenge, even for splines. All automatic spline fitting models exhibit overfitting or underfitting (Figure 11). This is where SplineCloud's Fine-Tuning mode can be helpful. By introducing breaks in continuity in the mode change points, and manipulating control points and knot vector, it is possible to manually fit a curve[2] to the complex stepped data (Figure 12).

[2] Link to curves fitted to the discharge rates from the test data from [8]



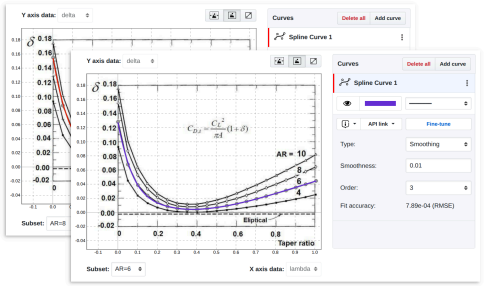
(a) Auto fitted spline models.

(b) Fine-tuned spline model.

Figure 12. *Spline models fitted to experimental data on the step-drawdown test data.*

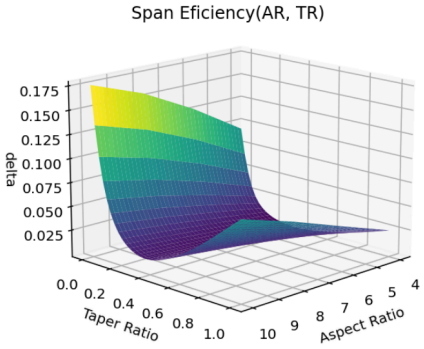
7.2. Reconstructing Empirical Relations for Wing Span Efficiency Factor

When designing an aerial vehicle, preliminary estimates of wing lift and drag forces are essential. These calculations are often based on empirical data obtained from wind tunnel experiments or numerical simulations. Such data is typically processed and generalized across different combinations of relative design parameters. One example is the estimation of the induced drag coefficient for tapered wings. Amongst various approaches [9], empirical relationships for the span efficiency parameter δ, used in induced drag calculations, were proposed by B. W. McCormick [10] in graphical form.



(a) Auto fitted spline models.

```
delta_curves = {
  "AR=10": load_spline("spl_Nb5m075aHe42"),
  "AR=8":  load_spline("spl_YUtYmXHXXK5Q"),
  "AR=6":  load_spline("spl_VsE31rlqsGtH"),
  "AR=4":  load_spline("spl_lvdZRtwoGU6T")
}
```



(b) Fine-tuned spline model.

Figure 13. *Spline models fitted to empirical relations of the span efficiency parameter δ.*

Using SplineCloud, data from such plots can be digitized and spline models created — see Figure 13 (a). These models[3] can then be reused programmatically to construct a response surface model using linear interpolation between curves (see Figure 13 b). This enables direct evaluation of the δ parameter and ultimately the induced drag coefficient for different wing planforms. The resulting models are shareable and reusable, enabling other researchers to avoid repeating the same steps and improving reproducibility.

[3] Data relations constructed using plot for efficiency factor [10]

7.3. Building Reusable Libraries of Airfoils' Performance Curves

Airfoil performance data are fundamental for aerodynamic analyses in a wide range of aerospace applications, including wind turbines, unmanned aerial vehicles, and civil and transport aircraft. Usually, an objective in such analyses is the selection of airfoils appropriate to specific operational conditions. This task typically involves evaluating large sets of airfoils and their associated performance data, obtained either through wind tunnel experiments or numerical simulations. Resources like Airfoil Tools provide access to the results of simulations for a huge library of airfoils, and PDFs with airfoil performance data can be found in many university libraries: e.g. DTU library published Wind Turbine Airfoil Catalogue [11] that contains results of wind tunnel tests and numerical analyses for a collection of airfoils used in wind turbines.

A common limitation of these resources is their lack of programmatic accessibility and reusability. Airfoil Tools does not expose its datasets via an API, and extracting numerical values from plots in PDF documents remains a manual and time-consuming operation. Despite this, processing, fitting, and analysis of airfoil performance data are routine tasks in aerospace engineering and are frequently duplicated across research groups.

The recent approaches based on neural networks, aiming to predict aerodynamic coefficients directly from airfoil geometry, are bypassing the need for raw experimental or simulation data. For example, the AirfoilNet model [12] uses image-based representations for end-to-end prediction of lift and drag. While effective for automated prediction, such models emphasize performance over physical interpretability and reusability.

Alternatively, SplineCloud can be used to create and host airfoil performance curves fitted with splines, offering tracable, interpretabile and reusable approach. The collection of curve IDs from such repositories can be shared publicly to simplify access and reuse in custom scripts and optimization algorithms Figure 14.

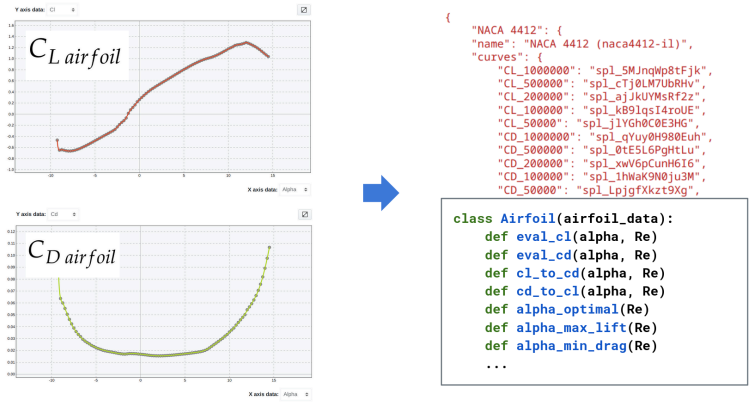


Figure 14. *Accessing and reusing airfoil performance curves in Python*

Several airfoil repositories are already available on the platform[4] and are actively used in applications such as the Wing generator web application developed for demonstration purposes Figure 15 . Upon selecting an airfoil, the application loads lift and drag curves, along with profile geometry from SplineCloud. These data are then used to compute aerodynamic loads, wing bending, and generate a 3D model of the wing using the CQ-UAV library.

[4]SplineCloud repositories containing tag 'airfoils'

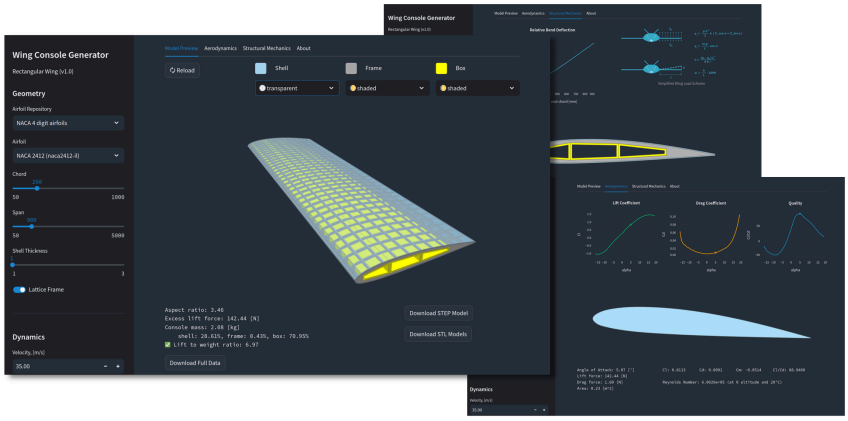


Figure 15. *Wing console generator web application that reuses airfoil performance curves from SplineCloud repositories*

8. Summary and Future Directions

This paper presented a collaborative and transparent approach to curve fitting, enabled by SplineCloud — an open platform that extends SciPy's interpolate module with interactive spline modeling and reproducibility features.

The paper contains an overview of the spline fitting approaches implemented in SciPy and SplineCloud, illustrating the practical utility of spline-based regression for addressing complex real-world problems across diverse scientific and engineering domains, especially in problems where flexible and accurate curve fitting is essential for extracting meaningful relationships from experimental and simulation data.

As described in Interactive Spline Fitting Workflow of this paper, SplineCloud extends the capabilities of SciPy and addresses key limitations of traditional programmatic spline fitting by providing visual control over spline parameters, enabling fine-tuning through interactive manipulation of control points and knot vectors. The application of FAIR principles to regression modeling enhances model reusability and promotes reproducibility in data analysis workflows. By decoupling spline models from their original fitting code and providing programmatic access through the client libraries, the platform enables researchers to build shareable repositories of empirical relations, reducing redundant effort for data processing and enhancing the reliability of model-based research outputs.

This is proved by the use case studies, given in Use Cases and Applications, that illustrate the platform's effectiveness in addressing real-world challenges, including: complex data fitting from hydrogeological experiments; improving reusability of models based on historical data in aerodynamics; and the creation of digital libraries of empirical relations to support engineering analysis and design automation in aerospace applications.

Future development directions include expanding the platform's capabilities to support multivariate splines, API and client libraries improvements, and features required for collaboration, metadata structuring, provenance, traceability and transparency. SplineCloud's mission and its approach align with the growing emphasis on open science and reproducible research practices in computational science and engineering. As the platform matures, it has the potential to become widely adopted in collaborative data-driven modeling, enabling researchers to discover, validate, and build upon existing results, saving time and resources for creativity and innovation.

Acknowledgments

This work was supported by funding from the European Union under the Seeds of Bravery initiative, as part of the Horizon Europe program for supporting Ukrainian innovation. Prior support was provided by CRDF Global, whose contribution played an essential role in the early development of the platform.

The development of SplineCloud would not have been possible without the dedication and expertise of the team. The author extends sincere gratitude to Olena Onyshchenko, Maksym Valyn, Daria Prokaza, Taras Talimonchuk, and Alina Nehoda for their valuable contributions to engineering, design, and research efforts.

References

[1] M. D. Wilkinson *et al.*, "The FAIR Guiding Principles for scientific data management and stewardship," *Scientific Data*, vol. 3, no. 1, 2016, doi: 10.1038/sdata.2016.18.

[2] M. G. COX, "The Numerical Evaluation of B-Splines," *IMA Journal of Applied Mathematics*, vol. 10, no. 2, pp. 134–149, 1972, doi: 10.1093/imamat/10.2.134.

[3] C. de Boor, "On calculating with B-splines," *Journal of Approximation Theory*, vol. 6, no. 1, pp. 50–62, 1972, doi: https://doi.org/10.1016/0021-9045(72)90080-9.

[4] T. Lyche and K. Mørken, *Spline Methods. Draft*. Department of Informatics, Centre of Mathematics for Applications, University of Oslo, [Online]. (last visited on 02/15/2018), 2008, p. 230. [Online]. Available: http://www.uio.no/studier/emner/matnat/ifi/INF-MAT5340/v07/undervisningsmateriale/

[5] F. Holmér, "The Continuity of Splines." [Online]. Available: https://youtu.be/jvPPXbo87ds?si=NbPi7mdrAJJQCUvG

[6] E. Lee, "Choosing nodes in parametric curve interpolation," *Computer-Aided Design*, vol. 21, no. 6, pp. 363–370, 1989, doi: https://doi.org/10.1016/0010-4485(89)90003-1.

[7] V. Pasko, "Approximation by Spline Functions and Parametric Spline Curves with SciPy." [Online]. Available: http://vadym-pasko.com/blog/2015/03/06/spline-approx-scipy.html

[8] G. Van Tonder, J. Botha, and J. Van Bosch, "A generalised solution for step-drawdown tests including flow dimension and elasticity," *Water SA*, vol. 27, no. 3, 2001, doi: 10.4314/wsa.v27i3.4978.

[9] D. S. M. Ni\textcommabelow ťa, "Estimating the Oswald Factor from Basic Aircraft Geometrical Parameters," *Deutscher Luft- und Raumfahrtkongress 2012, Berlin*, p. 19, 2012, [Online]. Available: https://www.dglr.de/publikationen/2012/281424.pdf

[10] B. W. McCormick, *Aerodynamics, Aeronautics, and Flight Mechanics*. John Wiley & Sons, 1979, p. 652. [Online]. Available: https://rexresearch1.com/FlightMechanicsLibrary/AerodynamicAeronauticsFlightMechanics.pdf

[11] F. Bertagnolio, *Wind turbine airfoil catalogue*. Denmark. Forskningscenter Risoe. Risoe-R No. 1280(EN): DTU Library, 2001, p. 151. [Online]. Available: https://backend.orbit.dtu.dk/ws/portalfiles/portal/7728949/ris_r_1280.pdf

[12] Z. Cai, Z. Fan, and T. Liu, "Efficient aerodynamic coefficients prediction with a long sequence neural network." [Online]. Available: https://arxiv.org/abs/2403.14979