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

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