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

Published Jul 10, 2025

Correspondence to
Marco Colonna
marco.colonna@cern.ch

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.

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.

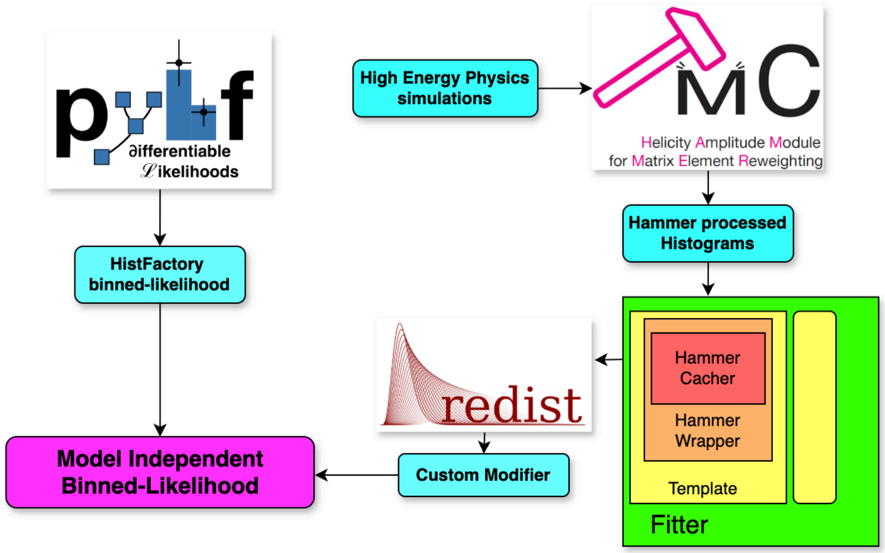
pyhf
∂ifferentiable
ℒikelihoods
HistFactory binned-likelihood
High Energy Physics simulations
Helicity Amplitude Module for Matrix Element Reweighting
Hammer processed Histograms
Hammer Cacher
Hammer Wrapper
Template
Fitter
redist
Custom Modifier
Model Independent Binned-Likelihood

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:

```json
{
  "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:

```python
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 pseudo-dataset 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.

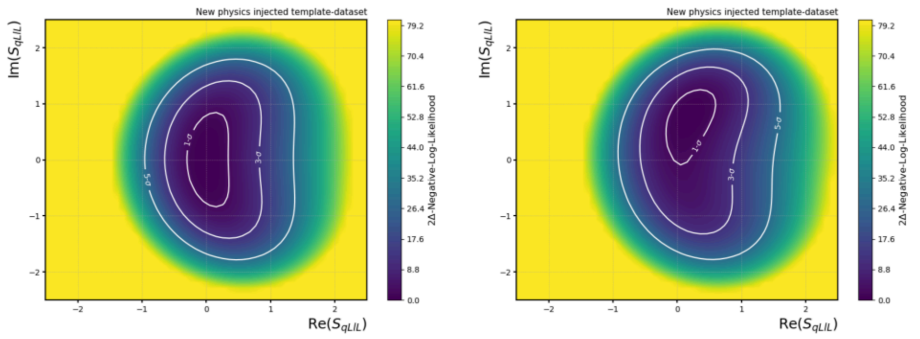

Figure 2. *Negative-Log-Likelihood phase-space scan of the scalar Wilson Coefficient on a template-dataset with no NP injection (left) and on a template dataset with the NP injection of* $Re(S_{qLlL}) = 0.2$ *and* $Im(S_{qLlL}) = 0.8$ *(right). The confidence levels correspontent to 1, 3 and 5 standard 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.

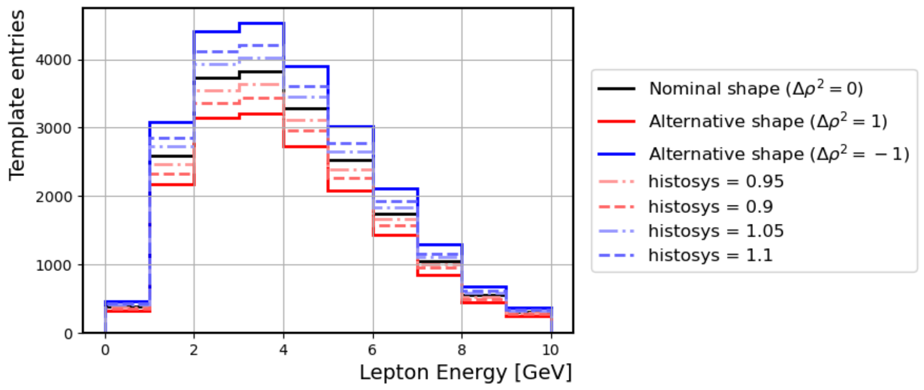

Figure 3. *Distributions of a 1-dimensional* pyhf *template for different values of the* histsys *modifier, defined with respect to two different injection of values of the* ρ^2 *FF parameter in HAMMER. Redist-HAMMER allows the integration of the theoretical uncertainty on the Form Factor parameter as 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