

Latest Developments in RooFit and Plans

Jonas Rembser (CERN EP-SFT) for the ROOT team

May 6 2025

WLCG/HSF Workshop 2025

ICJLab, Paris



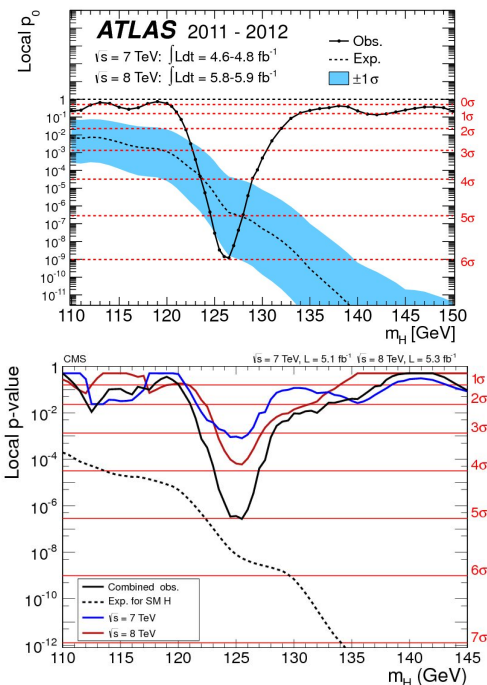
Introduction to RooFit

- ▶ **RooFit**: C++ library for statistical data analysis in ROOT
 - provides tools for model building, fitting and statistical tests
- ▶ Recent development focused on:
 - **Performance** boost (preparing for larger datasets of **HL-LHC**)
 - More **user friendly** interfaces and high-level tools

In **this presentation** we're summarizing the RooFit developments.

The covered areas:

- **Accelerating** likelihood evaluation, also on the GPU
- **Automatic Differentiation (AD)** in RooFit with Clad
- Recent **Minuit 2** improvements
- **Community** efforts in the RooFit ecosystem
- **Simulation Based Inference** (fully differentiable!)
- **Future** plans



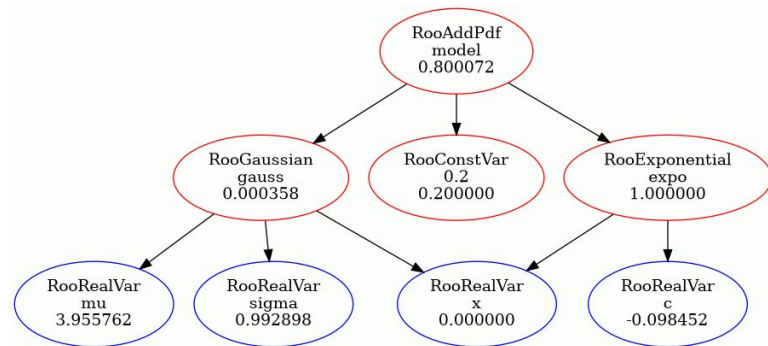
*RooFit was used in the
Higgs boson discovery!*



Why RooFit?

RooFit is serving the HEP community well because of several **key features**:

- ▶ Likelihood functions **highly optimized** for the context of minimization with Minuit
- ▶ It takes care of **analytical normalization** integrals where possible
- ▶ **User-extendible** framework that can cover a wide range of use cases
 - *Binned* likelihood fits
 - *Unbinned* likelihood fits
- ▶ Sharing of **statistical workspaces** thanks to ROOTs powerful IO system



Conditional pdf example:

$$p(x|y) = \frac{p(x, y)}{p(y)} = \frac{p(x, y)}{\int p(x, y) dx}$$

Observable subdomain example:

$$p(x|\text{subrange}) = p(x) \frac{\int_{\text{full}} p(x) dx}{\int_{\text{subrange}} p(x) dx}$$



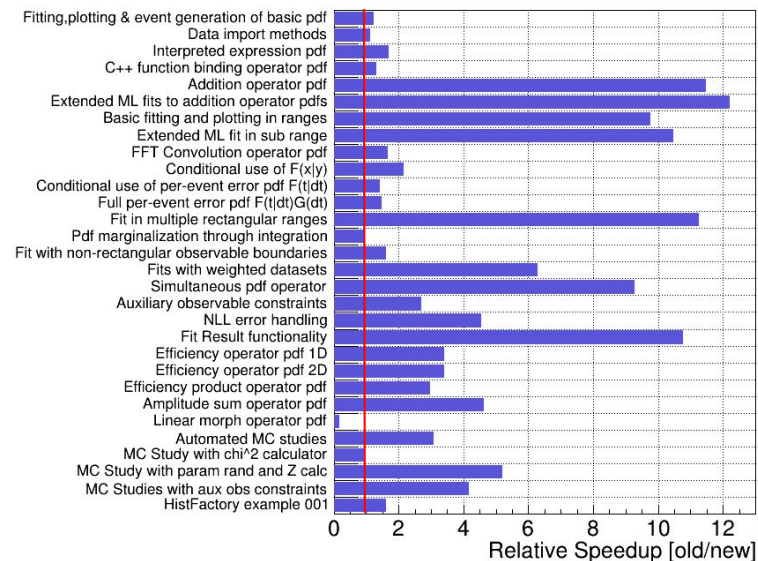
Accelerating likelihood evaluation



RooFits new vectorizing likelihood evaluation

- ▶ **Performance matters!**
RooFit workflows in experiments can run for hours or even days on the grid
- ▶ RooFits new likelihood evaluation has been presented on several occasions already
- ▶ Speedup for almost all tests from a combination of:
 - Vectorized** evaluation
 - Optimized computation graphs
 - Less function calls
- ▶ Average speedup of **4.4x**
- ▶ **New:** the vectorizing CPU backend is the **default since ROOT 6.32** released in 2024!
 - Showing it's possible to modernize significant parts of production software “in flight”

RooFit/HistFactory stress tests: speedup of NLL minimization by using BatchMode("cpu")

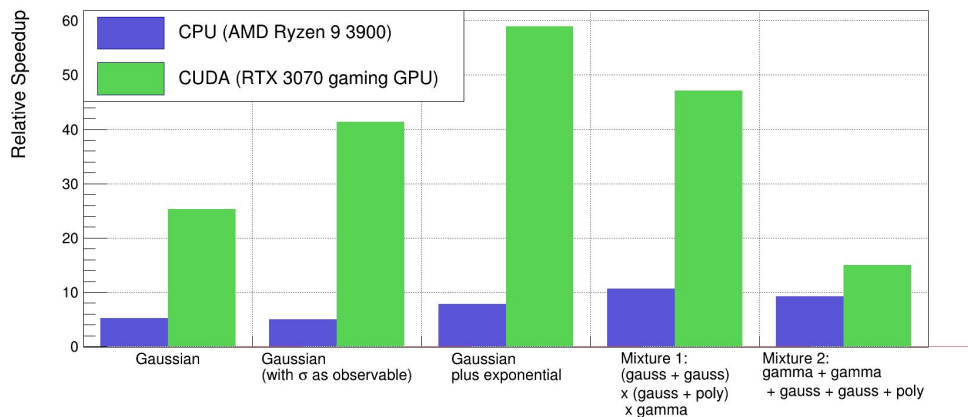


Results obtained with ROOT 6.28.04 from [CHEP 2023](#).

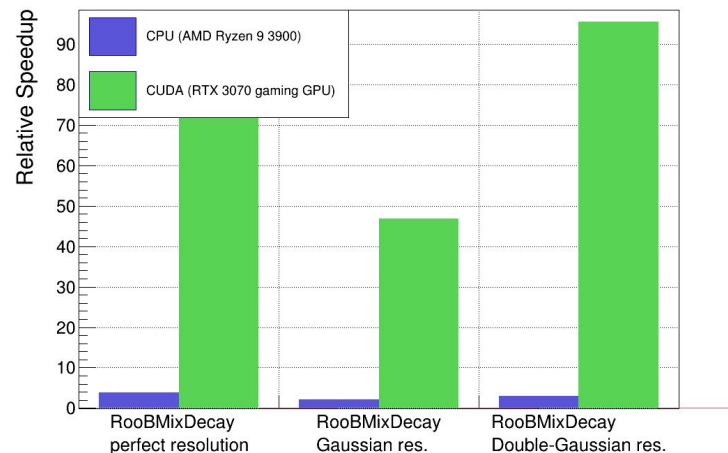


Great speedup possible for unbinned fits with many events with the **CUDA backend**:

RooFit: speedup in benchmark fits with BatchMode() relative to old RooFit (1 million events)



RooFit: speedup in benchmark fits with BatchMode() relative to old RooFit (1 million events)



- **New in 2025:**
 - Experiments making an effort to support RooFit CUDA in their environments
- Future plans:
 - **Numerical integrals** on the GPU
 - Support for **user functions** by just-in-time compiling CUDA Kernels with Cling

[benchRooFitBackends](#) and [RooFitUnBinnedBenchmarks](#) in [rootbench](#) repo, plotting script is in same directory. Try it yourself! Remember to use a ROOT build with `-Dcuda=ON`

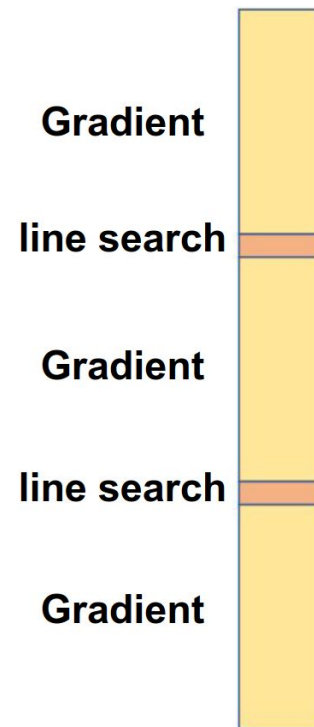
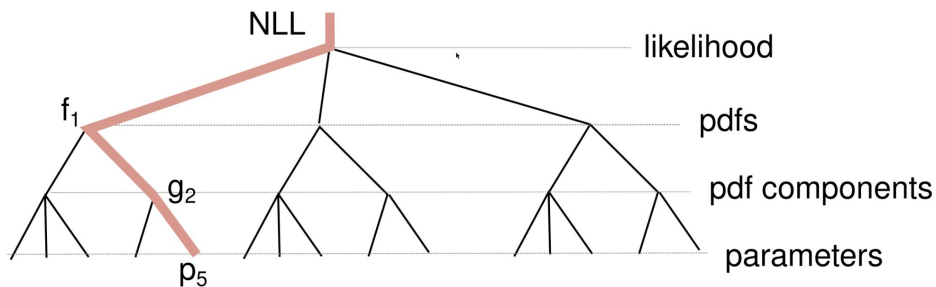


Automatic differentiation (AD) in RooFit



Numeric minimization of RooFit Likelihoods

- By default, RooFit uses **numerical differentiation**: Minuit 2 changes parameters **on-at-the-time** to get the full gradient
- One key concept of RooFit: **caching of intermediate results** to minimize redundant computations in gradient evaluation
- Still, gradient dominates minimization time (*see also the [ICHEP 2022 RooFit presentation](#)*)



Our goal: make evaluating gradients cheap with **Automatic differentiation (AD)**



Automatic differentiation engine for RooFit

- ▶ RooFit is a framework to build **computation graphs for function minimization**, similar to the ML frameworks **TensorFlow** or **PyTorch**
- ▶ Different from other frameworks, RooFit didn't have an **automatic differentiation engine**
- ▶ However, the other frameworks are generally not optimized for HEP use cases and workflows



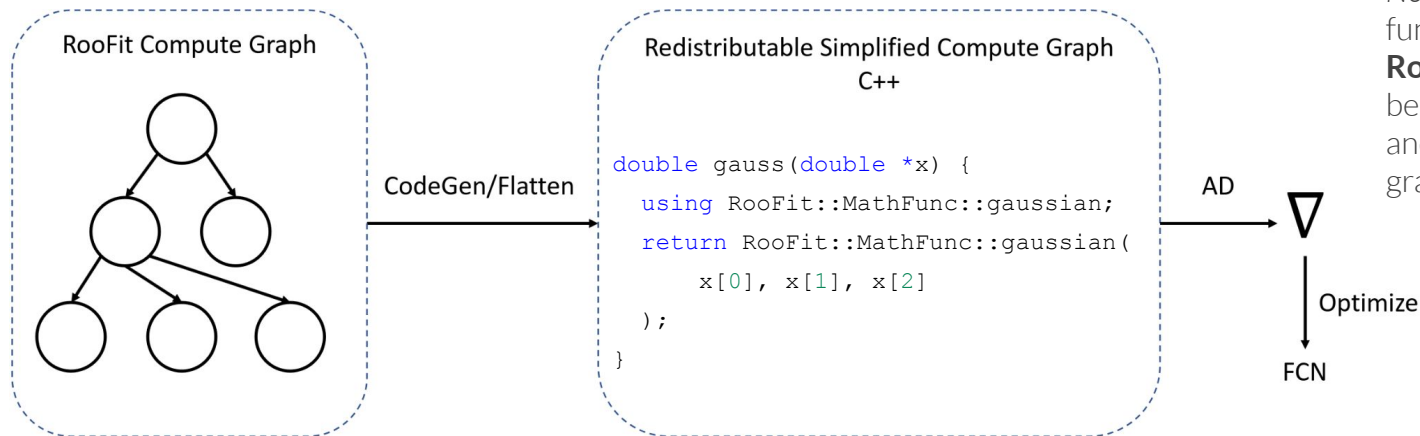
Therefore, we have added a differentiation engine based on **Clad** and **C++ code generation** to RooFit.



Automatic Differentiation with Clad

RooFit uses Clad to get analytic gradients: **Code generation** (aka. “codegen”)
More detail in [last month's ROOT blog post](#)

1. **Mathematical** concept
2. **RooFit** user code
3. **Automatic translation** of RooFit model to simple C++ code
4. **Gradient** of C++ code **automatically generated** with **Clad**
5. Gradient code **wrapped** back into RooFit object



Note: for the **nominal NLL** function, we **still use RooFits CPU backend** to benefit from vectorization and caching outside the gradients.

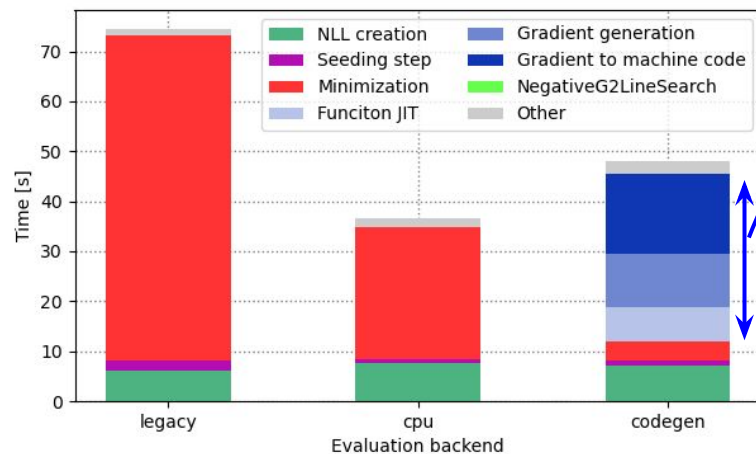


Performance with RooFit AD

- ▶ Using analytic gradients significantly **reduces minimization time** for many-parameter fits with
 - ATLAS HistFactory benchmark on the right
- ▶ Also **numerically more stable**: no tricks required to get better precision on numeric gradients (e.g. *likelihood offsetting*)
- ▶ *Caveat*: potentially long time for gradient generation
 - To benefit, workflow needs to reuse likelihood (e.g. **toy studies** or **profile likelihood scans**)

More detail in [ICHEP 2024 presentation](#).

Jit time can be amortized by re-using likelihood!



Detailed breakdown of minimization time for ATLAS Higgs combination benchmark with different RooFit backends (49 HistFactory channels, 739 parameters in total, in [rootbench](#))



Side-benefit of RooFit “codegen”: any tool that can generate C++ code naturally works with RooFit and you get AD for free.

- ▶ Example: using [SymPy](#) expressions in RooFit ([Presentation at PyHEP 2023](#))
- ▶ We will see another example later: **TMVA-SOFIE**

```
import sympy as sp

x, mu = sp.symbols('x mu')

gauss = sp.exp(-0.5*(x - mu)**2)

c_code = sp.ccode(gauss)

ws = ROOT.RooWorkspace()
ws.factory(f"CEXPR::gauss('{c_code}', x[0., 10.], mu[5., 0., 10.])")
```

Code example on how to use C code exported from sympy in a RooFit workspace.



Minuit 2 improvements for RooFit



Profiling RooFit - not a black box!

- ▶ RooFit serves many use cases and users hit **different bottlenecks**
- ▶ Since written in C++, RooFit code is convenient to profile
- ▶ Flamegraphs often inspire **significant performance improvements** in RooFit
- ▶ Guarantees that RooFit continues to scale well for cutting edge fits

Example workflow to profile ROOT macro with `perf` and `flamegraph.pl`:

- *Make sure ROOT is built with debug info but not in debug mode*
(`-DCMAKE_BUILD_TYPE=RelWithDebInfo`)
- *Macro needs a `main()` function so it can be compiled*

```
g++ $(root-config --cflags --libs) -g \  
    -lRooFitCore -lRooFit -o fit_macro \  
    fit_macro.C  
perf record -F 99 -g -- "./fit_macro"  
perf script | stackcollapse-perf.pl >out.perf-folded  
flamegraph.pl out.perf-folded > flamegraph.svg
```

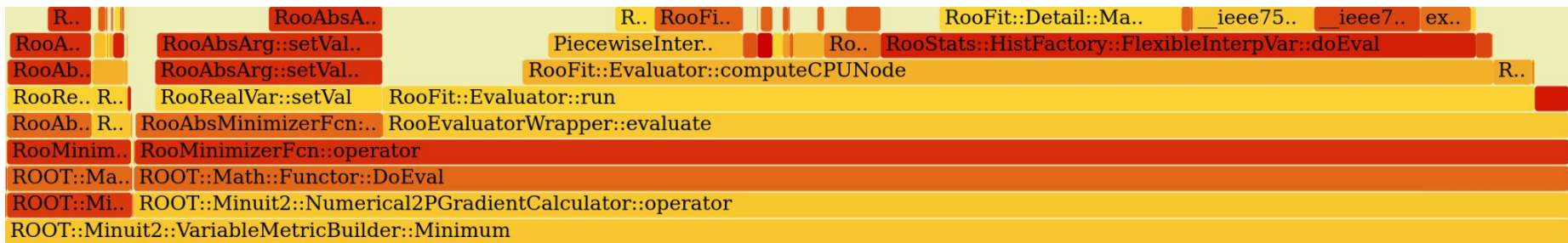


Optimizing Minuit 2 for RooFit

Example of an optimization after profiling:

- ▶ **Transforming** parameters into **Minuit 2 internal space** was **bottleneck** of uncertainty estimation with **HESSE**
- ▶ This is **fixed in [ROOT 6.36](#)** by caching transformed parameters
 - Up to 50 % faster HESSE for highly-optimized likelihoods (ATLAS, CMS Higgs combinations)

Holistic optimization of RooFit + Minuit 2:
exemplifies *benefits of an integrated software stack*



Flamegraph for minimization with Minuit 2 of the ATLAS Higgs combination benchmark with ROOT 6.36

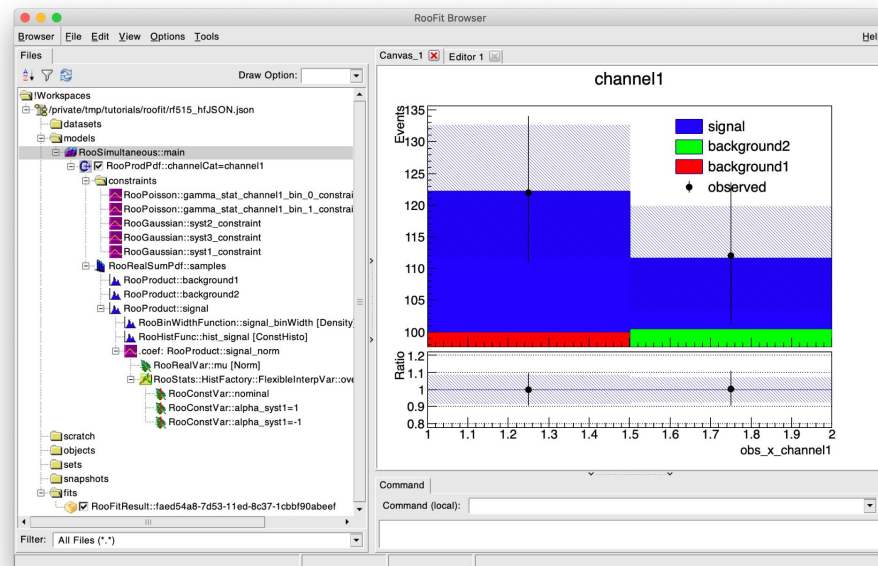


Community Contributions to RooFit



xRooFit and RooBrowser

- ▶ [xRooFit](#) is a high-level abstraction layer over RooFit (also shipped with ROOT)
- ▶ Developed by ATLAS member W. Buttinger
- ▶ It sits between RooFit and other end-user frameworks in ATLAS
- ▶ Can be seen as an **incubator** for new RooFit interfaces before they become part of the stable RooFit

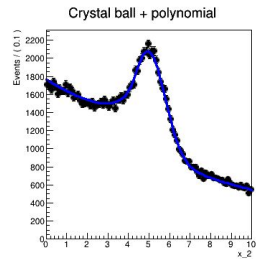
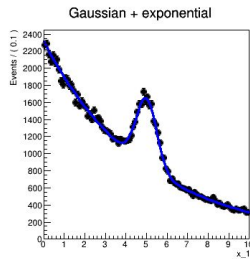


Screenshot of the [RooBrowser](#) to interactively explore RooFit workspaces

- ▶ The [HEP Statistics Serialization Standard](#) aims to define a statistical workspace format based on JSON
- ▶ Its [reference implementation](#) is integrated in RooFit
- ▶ Especially useful for likelihood publishing when binary ROOT files are a concern
- ▶ Several [ATLAS analyses](#) already have HEPData entries that include HS3 JSONs! (*and xRooFit code examples*)

```

"distributions": [
  {
    "name": "sig_1",
    "type": "gaussian_dist",
    "x": "x_1",
    "mean": "mean",
    "sigma": "sigma_1"
  },
  {
    "name": "sig_2",
    "type": "crystalball_dist",
    "m": "x_2",
    "m0": "mean",
    "sigma": "sigma_2",
    "alpha": "alpha",
    "n": "ncb"
  },
  {
    "name": "model_1",
    "type": "mixture_dist",
    "coefficients": ["n_sig_1", "n_bkg_2"],
    "summands": ["sig_1", "bkg_1"]
  },
  {
    "name": "bkg_1",
    "type": "exponential_dist",
    "x": "x_1",
    "c": "c_1"
  },
  {
    "name": "bkg_2",
    "type": "polynomial_dist",
    "x": "x_x",
    "coefficients": ["3", "a_1", "a_2"]
  },
  {
    "name": "model_2",
    "type": "mixture_dist",
    "coefficients": ["n_sig_2", "n_bkg_2"],
    "summands": ["sig_2", "bkg_2"]
  }
],
"functions": [],
"likelihoods": [
  {
    "data": ["data_channel_1", "data_channel_2"],
    "distributions": ["model_channel_1", "model_channel_2"],
    "name": "my_likelihood"
  }
],
"analyses": [
  {
    "name": "my_analysis",
    "likelihood": "my_likelihood",
    ... etc. ...
  }
],
"data": [
  {
    "name": "data_channel_1",
    ... content ...
  },
  {
    "name": "data_channel_2",
    ... content ...
  }
],
"domains": [...],
"parameter_points": [...]
```





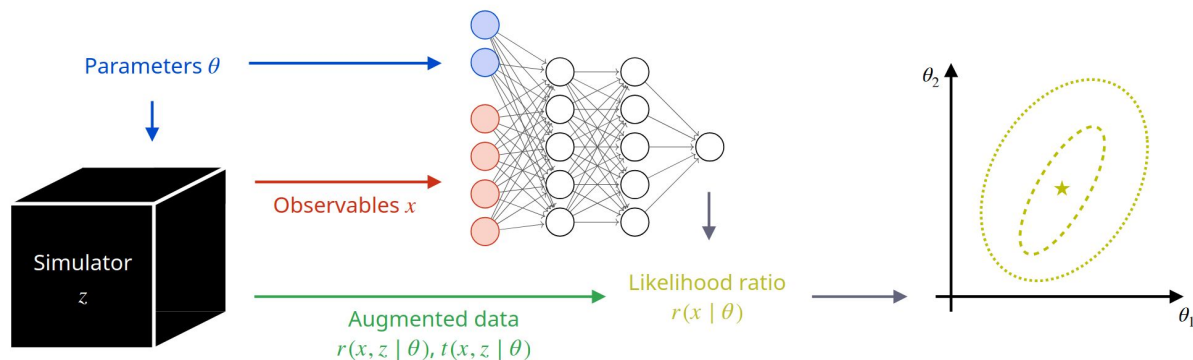
Simulation Based Inference



New Analysis Paradigms - Simulation Based Inference (SBI)

- ▶ In case where you don't have analytic models for probability, but you can **sample** with MC simulators
- ▶ Learn (**parametrized**) **likelihood ratio** to do parameter estimation **without any histograms**

Figure borrowed from [Alexander Held's talk](#) at the [PHYSTAT-SBI 2024 workshop](#)



$$\text{NLL}(\theta) = - \sum_i \log p(x_i | \theta)$$

likelihood ratio trick

$$- \sum_i \log \frac{p(x_i | \theta)}{p_{\text{ref}}(x_i | \theta)}$$

$$- \sum_i \log \frac{s(x_i | \theta)}{1 - s(x_i | \theta)}$$

learn likelihood ratio from MC samples



Wrapping Python functions in RooFit to enable SBI

- ▶ RooFit can now wrap Python functions that take and return **NumPy arrays**
- ▶ Crucial feature for **using ML models in your likelihood**, which are usually trained and evaluated with Python

```
# Set up RooRealVars before: m4l, mu, n_sig, n_bkg

def llr_zz_vs_higgs_f(m4l: np.ndarray) -> np.ndarray:
    prob = model_xgb.predict_proba(m4l.T)[:, 0]
    return (1 - prob) / prob

llr_zz_vs_higgs = RooFit.bindFunction("llr_zz_vs_higgs",
                                       llr_zz_vs_higgs_f,
                                       m4l)
```

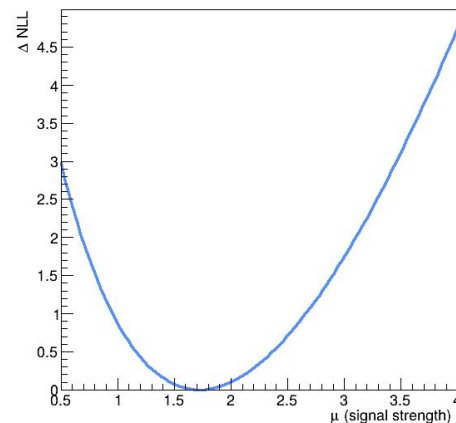
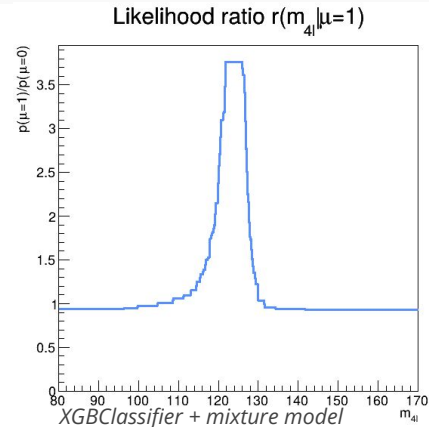
Example on how to create a RooFit object from a Python function with the pythonized RooFit::bindFunction()



Several new RooFit tutorials were added with SBI examples using RooFit:

- ▶ The [rf615 tutorial](#) shows a “Hello World” **Gaussian SBI fit**
- ▶ The [rf617 tutorial](#) repeats the exercise for a **higher-dimensional observable space**, highlighting where template histograms are limited
- ▶ The [rf618 tutorial](#) shows this analysis, which **follows up on the dataframe tutorial**
 - First RooFit tutorial that **uses open data!**

We already heard back from happy users that implement their own likelihoods based on the concepts shown in these tutorials.





- ▶ If the ML model could be **translated to simple C++ code**, it would naturally work with **RooFit AD** thanks to **Clad**
- ▶ ROOT already includes a package to do that: **TMVA-SOFIE!**

SOFIE : System for Optimised Fast Inference code Emit

Input: trained ML model file

Output: generated C++ code

Outputs

1. Weight File

Input: Trained ML Model
(.onnx, .pt, .h5)



ONNX



PyTorch



Keras

Parser: From ONNX (or Pytorch or Keras) to `SOFIE::RModel`

SOFIE
Parser

RModel



DAT

2. C++ header file



HXX



L. Moneta EP-SFT



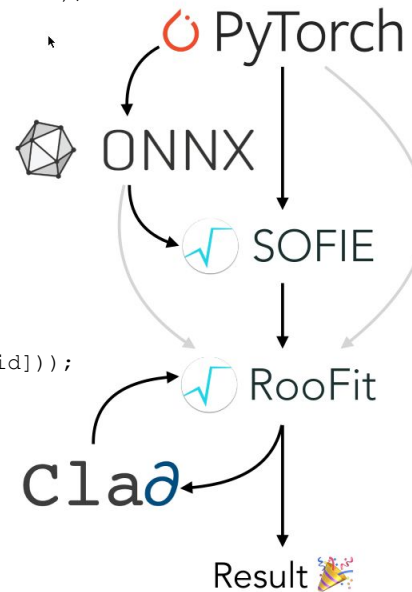
Fully differentiable SBI with RooFit, SOFIE and Clad

- ▶ Code emitted by **SOFIE** can generally be **differentiated** with **Clad**
- ▶ Still requires some tweaks that are yet to be merged in ROOT
- ▶ We plan to present example based on LHCb analyses in a future conference

Side effect: *shows the applicability of **Clad** to **deep learning**!*

```
void doInfer(float *tensor_x,
            float *tensor_theory_params,
            float *tensor_linear_3)
{
    using namespace TMVA::Experimental::SOFIE; // for Gemm_Call
    //----- Matrix multiplication
    Gemm_Call(tensor_linear,5, 1, 1, 1, tensor_val_0,
              tensor_theory_params,1, tensor_theory_projectorbias);
    //----- Add
    for (size_t id = 0; id < 5; id++) {
        tensor_add[id] = tensor_x[id] + tensor_linear[id];
    }
    //----- Matrix multiplication
    Gemm_Call(tensor_linear_1,128, 1, 5, 1, tensor_val_2,
              tensor_add,1, tensor_base_model0bias);
    //----- Sigmoid
    for (int id = 0; id < 128; id++) {
        tensor_val_4[id] = 1 / (1 + std::exp(-tensor_linear_1[id]));
    }
    ...
}
```

Extract of SOFIE-emitted code for fully-connected neural net





Outlook and Conclusions



- ▶ Modern RooFit has all **building blocks** for performant statistical analysis at scale
- ▶ In the near future:
 - **Deploy** modern RooFit features in experiment frameworks
 - **Upstream** some experiment code to RooFit (e.g. from CMS Combine)
 - Improve/add support for emerging **analysis approaches**:
 - Simulation Based Inference
 - Large-scale fits with many 1D projections (*like W mass*)
 - Continue improving RooFit **codegen** and **CUDA** backends



- ▶ RooFit has served the HEP community well, and continues to do so because **scalability is constantly improved** according to user needs
- ▶ With **Clad**, RooFit can make use of a powerful engine for **Automatic Differentiation** (AD)
- ▶ New **Python interfaces** enable new approaches to statistical analysis (e.g. Simulation Based Inference)
- ▶ The RooFit ecosystem is a HEP **community effort**
 - RooFit developers work closely with users from different experiments