Accelerating simulation-based inference in RooFit at LHCb with Clad

15th May 2025

Compiler Research Project Meeting

Johannes Albrecht, Marco Colonna, Jamie Gooding,
Abhijit Mathad, Biljana Mitreska, Jonas Rembser, Danilo Piparo

CERN ROOT Data Analysis Framework technische universität dortmund SMARTHEP REAL-TIME ANALYSIS FOR SCIENCE AND INDUSTRY MARIE CURIE ACTIONS

A little about me

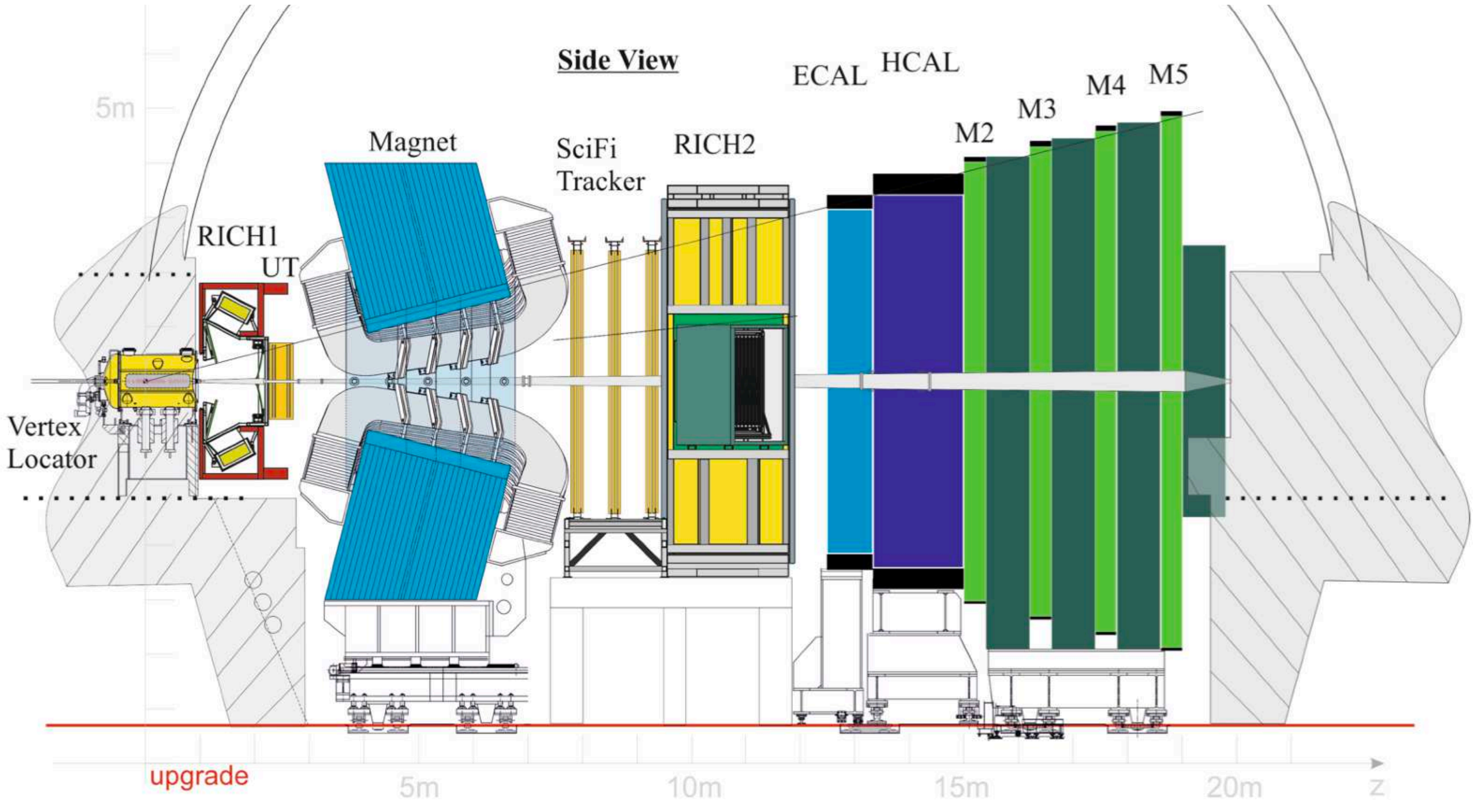
- 3rd Year Doctoral Student and SMARTHEP Early Stage Researcher at TU Dortmund in Dortmund, Germany
  - Member of LHCb since 2021 (Manchester + Dortmund)
  - Research focuses include:
    - Real-time analysis for trigger systems
    - Studies of CP-symmetry and lepton flavour universality violation in neutral b-meson decays
    - Novel tools and techniques for analysis (convener of [HEP Software Foundation](#) [Data Analysis Working Group](#))
- Currently working with Jonas Rembser and the ROOT team on a demonstrator for simulation-based inference (SBI) in an LHCb physics use case
  - Aim to produce example workflow for an LHCb physics application, whilst furthering ROOT SBI support

First of all, some physics*

This isn’t a prerequisite for understanding the rest of the talk, but adds some useful context!

The LHCb experiment at the LHC

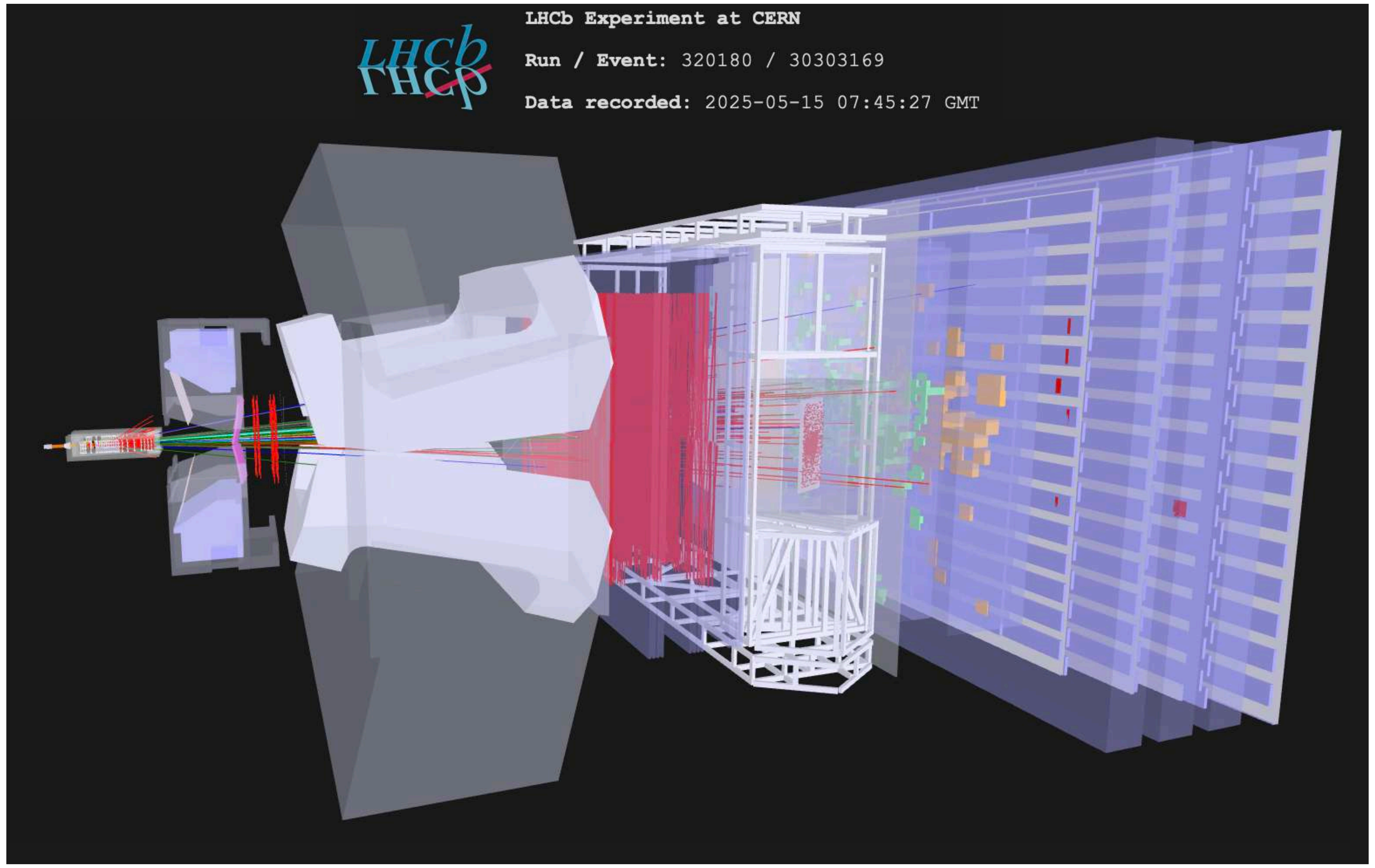
- The LHCb (Large Hadron Collider beauty) experiment is a dedicated flavour physics experiment at the LHC
  - Built to study the properties and decays of particles containing the heavy b and c quarks
  - By comparing against predictions from the Standard Model (SM, our best-understanding of particle physics) we look for signs of something out of place
  - We know something must be out there, as the SM doesn't describe gravity, dark matter, etc.
- As enormous a computing challenge as a physics/engineering challenge:
  - Processing $5\,\mathrm{TB/s}$ of data, store around $10\,\mathrm{GB/s}$
  - Statistical analyses regularly take place across datasets with millions of entries

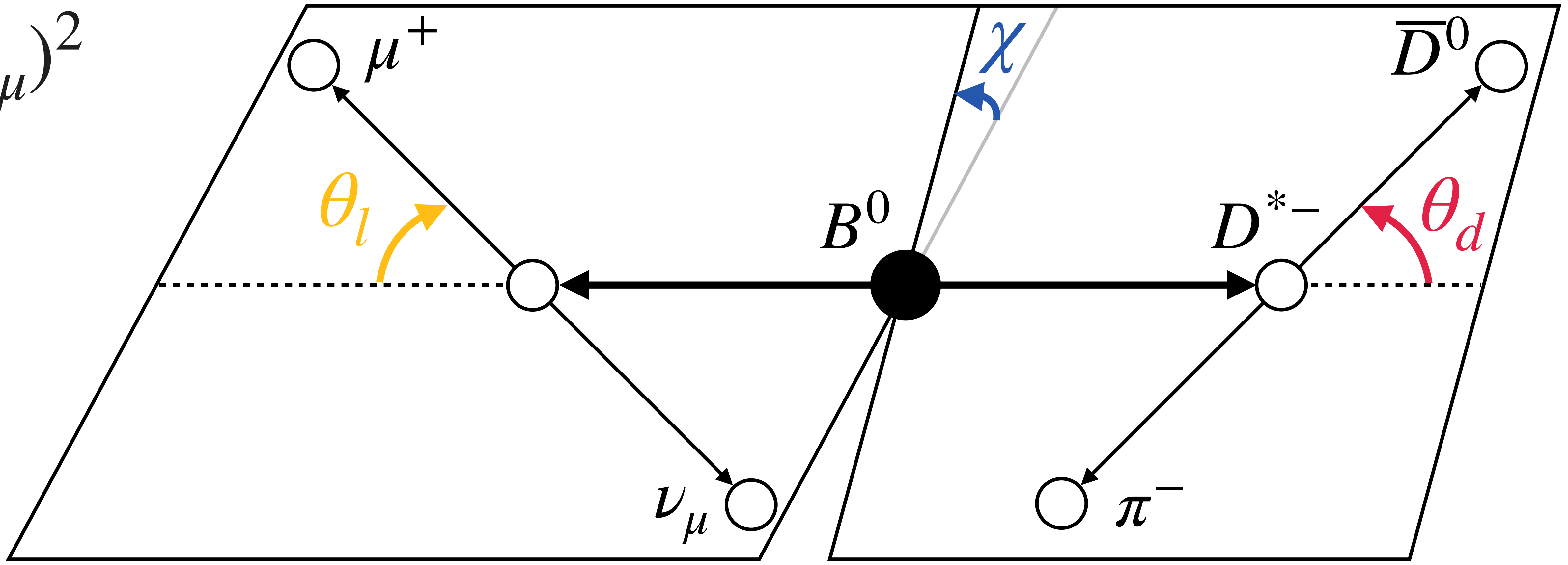


R. Aaij et al. JINST 19 (2024) P05065

Semi-leptonic $B^0 \to D^{*-}\mu^+\nu_\mu$ decays

- $B^0 \to D^{*-}\mu^+\nu_\mu$ decays readily observed at LHCb in pp collisions (e.g., *right from <u>this morning</u>*):
  - $D^{*-}(\to \overline{D}^0 \left(\to K^-\pi^+\right) \pi^-)$ and μ^+ fully reconstructed, but ν_μ <u>missing</u>
  - Angles of final state particles can be parameterised in three angles θ_d, θ_l, χ
  - We will also be interested in:
    - Momentum transfer, $q^2 = p_B^2 - p_{D*}^2$
    - Square missing mass, $M_{\text{miss}}^2 = (p_B - p_{D*} - p_\mu)^2$
- Can use the RapidSim fast MC package to generate pseudodata samples of these decays
- Angular observables may be affected by new physics (NP) contributions…





Parameterising new physics

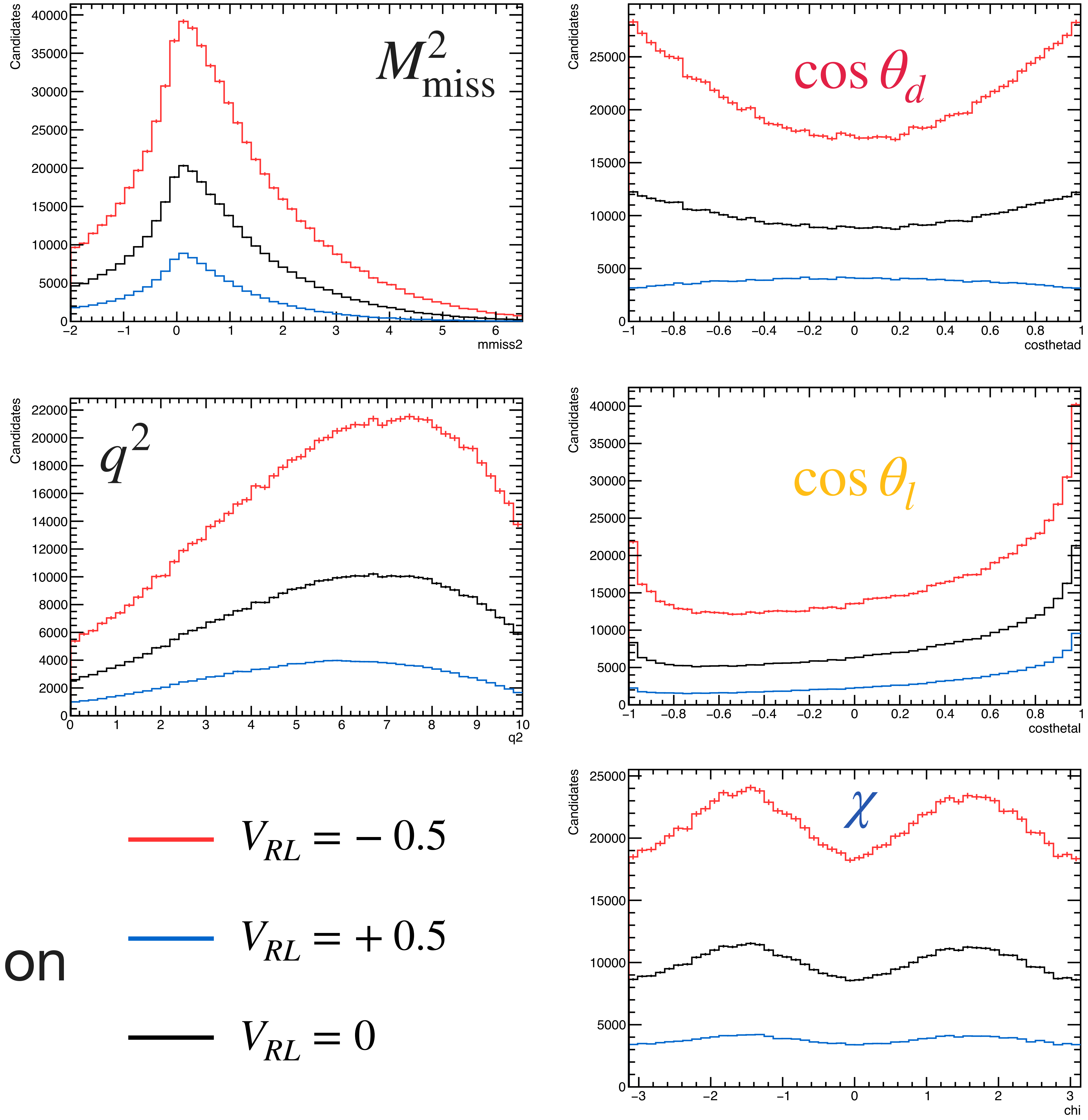
- Take an effective field theory (EFT) approach to considering decay processes
- Contributions to effective Hamiltonian expressed as Wilson operators, O_i, with coefficients, C_i:

$$H_{\text{eff}} \propto \Sigma_i \, C_i O_i \quad \text{with} \quad C_i = C_i^{\text{SM}} + C_i^{\text{NP}}$$

- These modify the decay rate with dependence on momentum scale and angular observables
- The NP (Wilson) coefficients (WCs), are complex and constructed to be 0 in the SM
- We consider contributions (with WCs) which are:
  - V_{RL}: vector coupling of right-handed Fermion to left-handed ν,
  - S_{LL}: scalar coupling of left-handed Fermion to left-handed ν
  - T_{LL}: tensor coupling of left-handed Fermion to left-handed ν
- The key takeaway from this: <u>we look at our kinematic and angular observables and fit for WCs</u>
- Note: for simplicity, we will only consider the real part of V_{RL} as our NP parameter

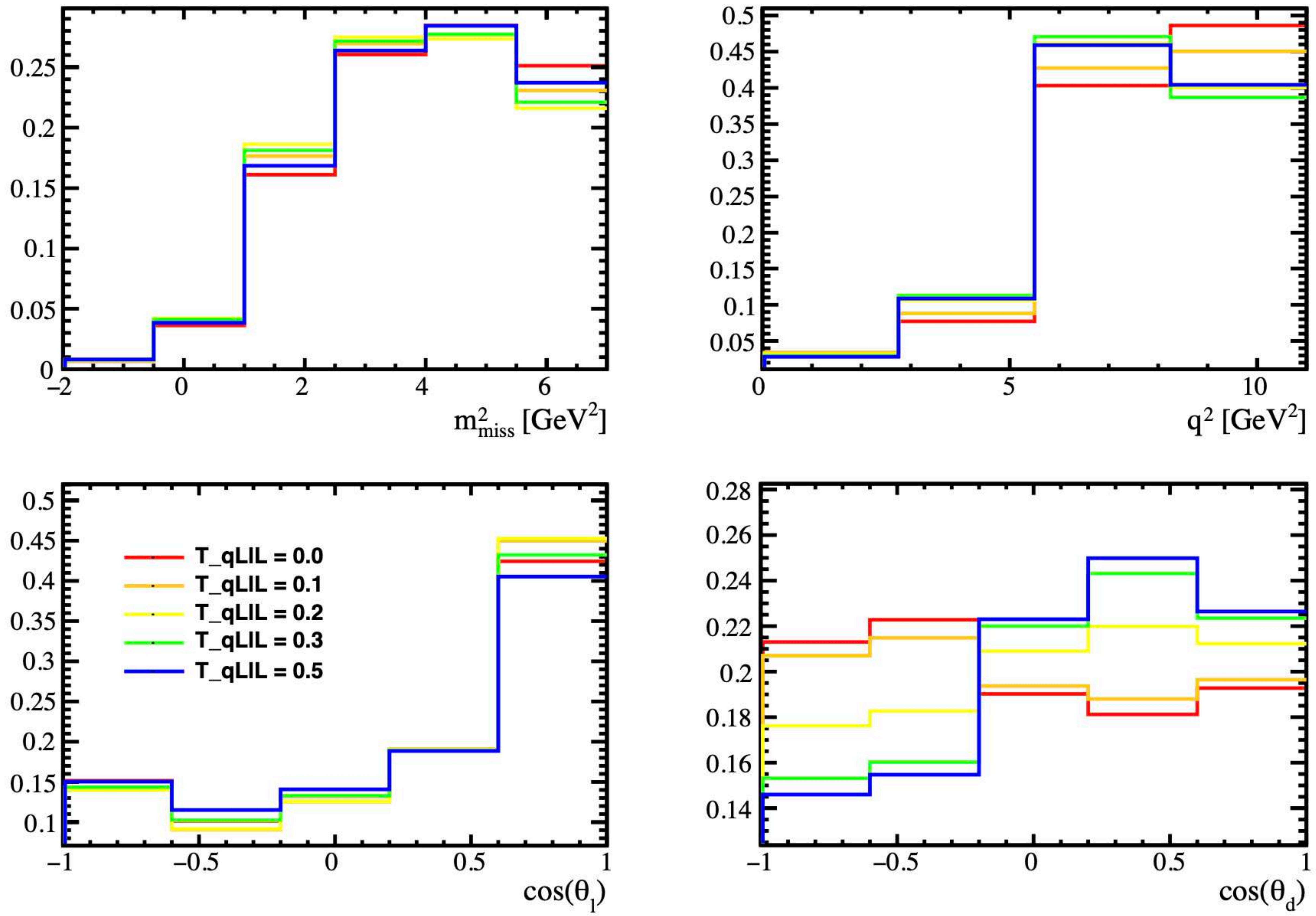
Injecting new physics

- Many tools exist to reweight samples for given NP scenarios, e.g., [HAMMER](), [EOS](), etc.
  - These compute relative rates based on specified WCs (and form factors, describing the strong interaction, excluded here for simplicity)
  - For our workflow, we use HAMMER
- We pass in our unweighted pseudodata sample and attach weights for our WC values of interest, *e.g.*, *right,* showing effect of $\mathrm{Re}\{V_{\mathrm{RL}}\}$
- In our workflow, compute weights in 4 scenarios:
  - $V_{\mathrm{RL}} = 0$, i.e., SM case
  - $V_{\mathrm{RL}} = \theta_{\mathrm{gen}}$, for θ_{gen} drawn from a uniform distribution
  - $V_{\mathrm{RL}} = \pm 0.5$, used for normalisation (*see later*)



Classical fitting of $B^0 \to D^{*-}\mu^+\nu_\mu$

- Classical approach is to bin each of the 5 dimensions and fit template histograms
  - Generate template histograms from MC simulation
  - Modify histograms with HAMMER rates
  - For each modification, compute likelihood (and gradient)
  - Repeat modification to maximise likelihood
- Often fit with [HistFactory](#)/[pyhf](#), integrated with tools like HAMMER (*e.g.*, [RooHammerModel](#) for HistFactory/[redist](#) for pyhf)
- This approach is well-established and optimised, though requires large MC sample and significant computing power



[B. Mitreska, talk in Challenges in SL B decays, September 2024](#)

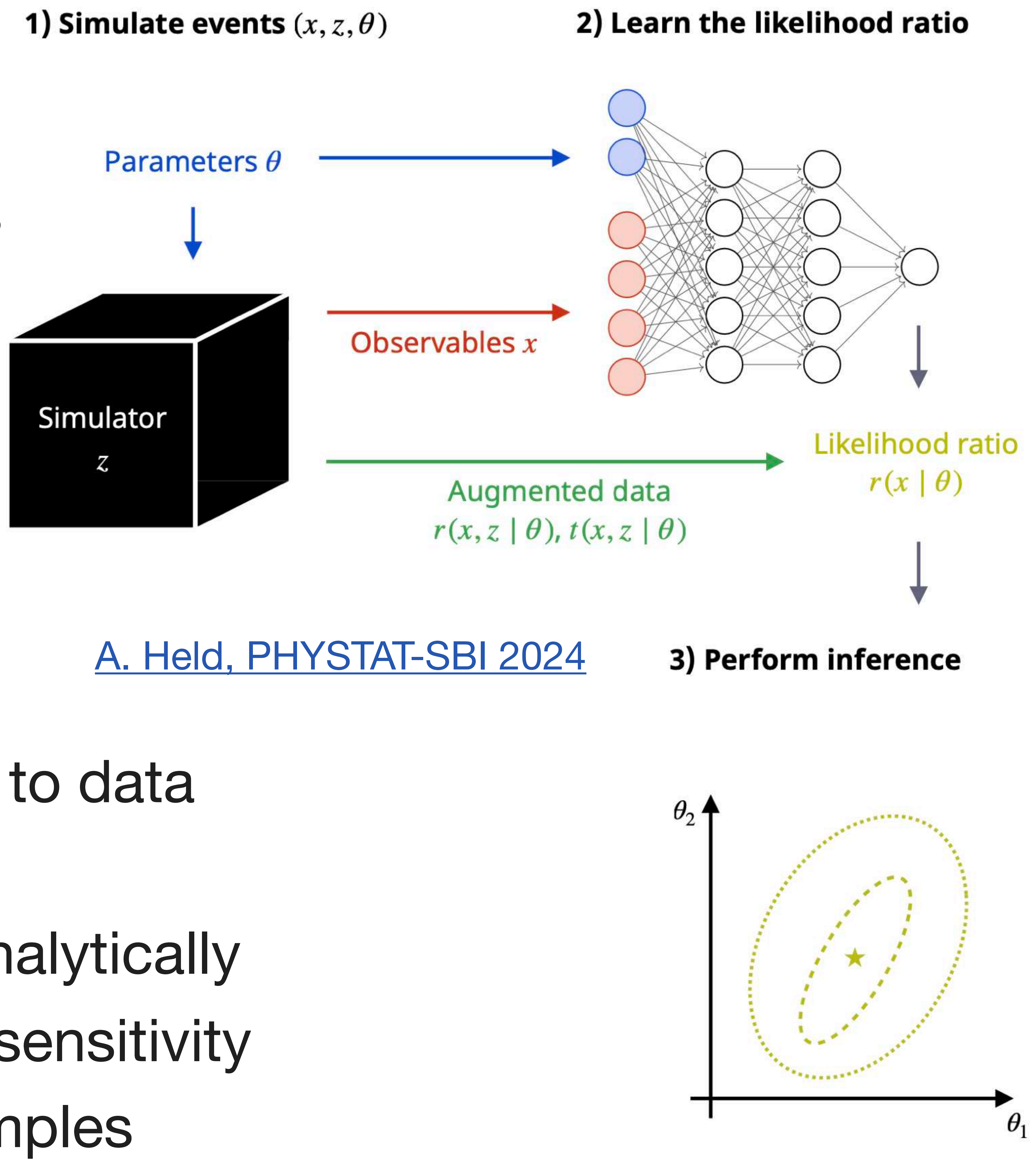
Simulation based inference

Simulation-based inference

- In lieu of analytic models for probability distributions, can learn the behaviour from simulation (*see right*)
- Train classifier to distinguish between different hypotheses
- Can apply the likelihood ratio trick: construct a negative log-likelihood ratio of hypothesis H and a reference hypothesis from the classifier score:

$$r(\theta) = -\sum_i \log\left(\frac{p_H\left(\vec{x}_i | \theta\right)}{p_{\text{ref}}\left(\vec{x}_i | \theta\right)}\right)$$

- Can compute ratio above from trained classifier and apply to data
- In our case this brings a few benefits:
  - Partially reconstructed backgrounds difficult to model analytically
  - No longer necessary to bin the data → potential gain in sensitivity
  - In principle, could reduce required size of simulation samples



A. Held, PHYSTAT-SBI 2024

Constructing a classifier to learn NP

- We construct a binary classifier NN to distinguish between samples with SM weights ($V_{\mathrm{RL}} = 0$) and NP weighted-samples ($V_{\mathrm{RL}} = \theta_{\mathrm{gen}}$ for unique θ_{gen} per event)
  - Neural network constructed as a theory-aware NN for physical features, $\vec{x}$, and theory parameters, $\vec{\theta}$, *i.e.*,
    - Base model, a dense NN (2 hidden layers with 100 nodes each) in $\vec{x}$: $s_{\mathrm{base}}\left(\vec{x}\right)$
    - Additional layer $\vec{v}\left(\vec{\theta}\right)$ of dimension $\left(n_{\theta}, n_{\vec{x}}\right)$ introduces NP modification of features:

$$\vec{x} \rightarrow \vec{x} + \vec{v}\left(\vec{\theta}\right)$$

    - This modification gives a NN with

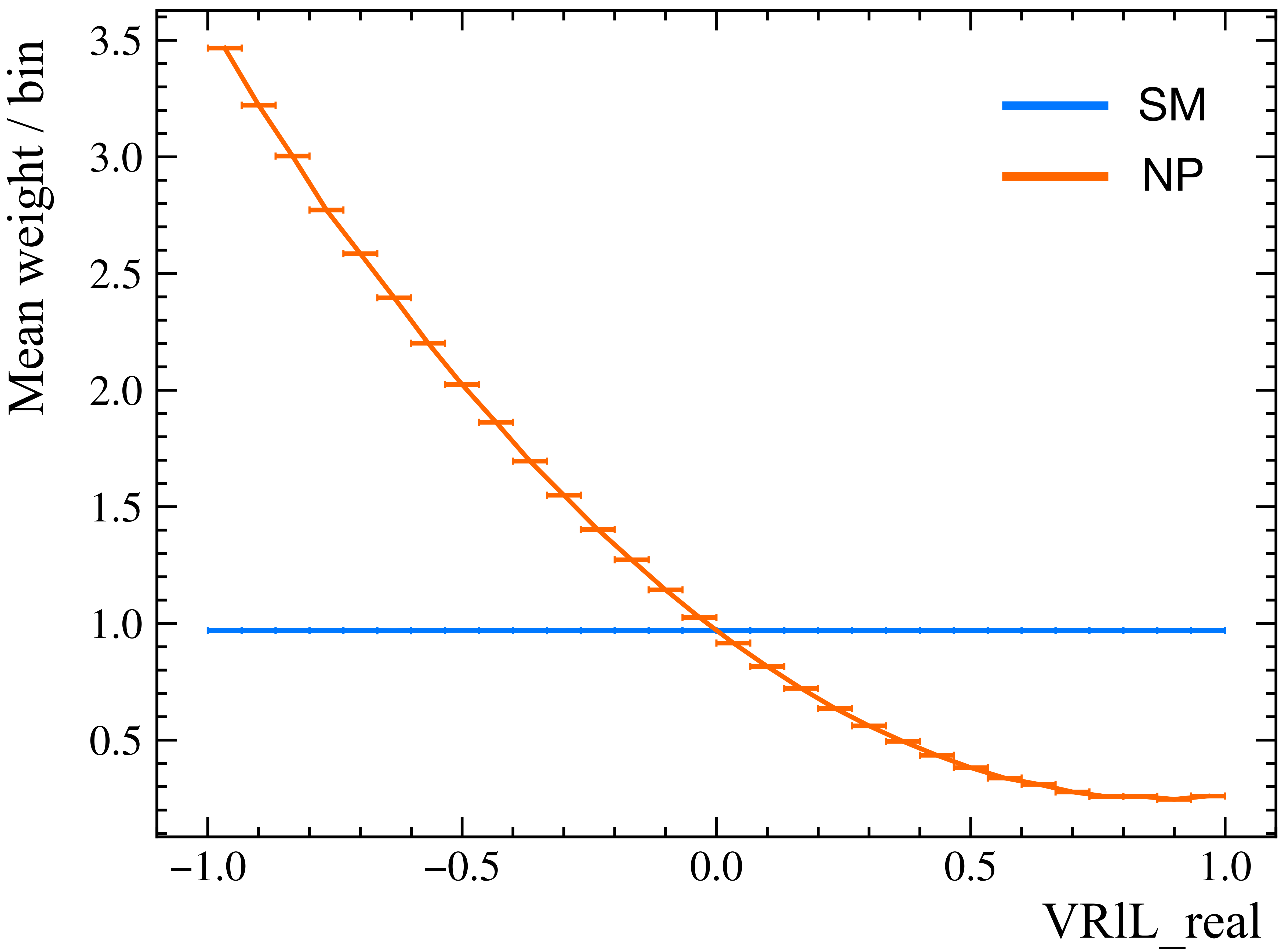
$$s_{\mathrm{base}}\left(\vec{x}\right) \rightarrow s\left(\vec{x}, \vec{\theta}\right) = s_{\mathrm{base}}\left(\vec{x} + \vec{v}(\vec{\theta})\right)$$

- At the moment, consider $\vec{\theta} = \{\theta\} = \{\mathrm{Re}\{V_{\mathrm{RL}}\}\}$ for demonstration
  - Plan to include multiple WCs and their terms in a single network

Generation of per-event weights

- We wish to generate a SM and NP weight (w^i_{SM} and w^i_{NP}) for every event
- Notice that the sum of weights has parabolic dependence on θ (*see right*)
  - Can define this normalisation as $I(\theta)$
  - Obtain this analytically by generating $\theta \in \{-0.5,\ 0,\ +0.5\}$; sum of weights for each gives 3 fixed points
- Then normalise the per-event weights as

$$w'^i_{\mathrm{SM}} = \frac{w^i_{\mathrm{SM}}}{I(0)} \qquad w'^i_{\mathrm{NP}} = \frac{w^i_{\mathrm{NP}}}{I(\theta)}$$



Training of the theory-aware NN

- Every event has weights w'^{i}_{SM} and w'^{i}_{NP}, with uniformly distributed random value of $\theta \in [-1,1]$ used for w'^{i}_{NP} per event
- Dataset constructed by concatenatenating data with w'^{i}_{SM} weights (label 0) and with w'^{i}_{NP} weights (label 1), train/test split in data of 80/20
- Using `BCEWithLogitsLoss` from PyTorch, loss is

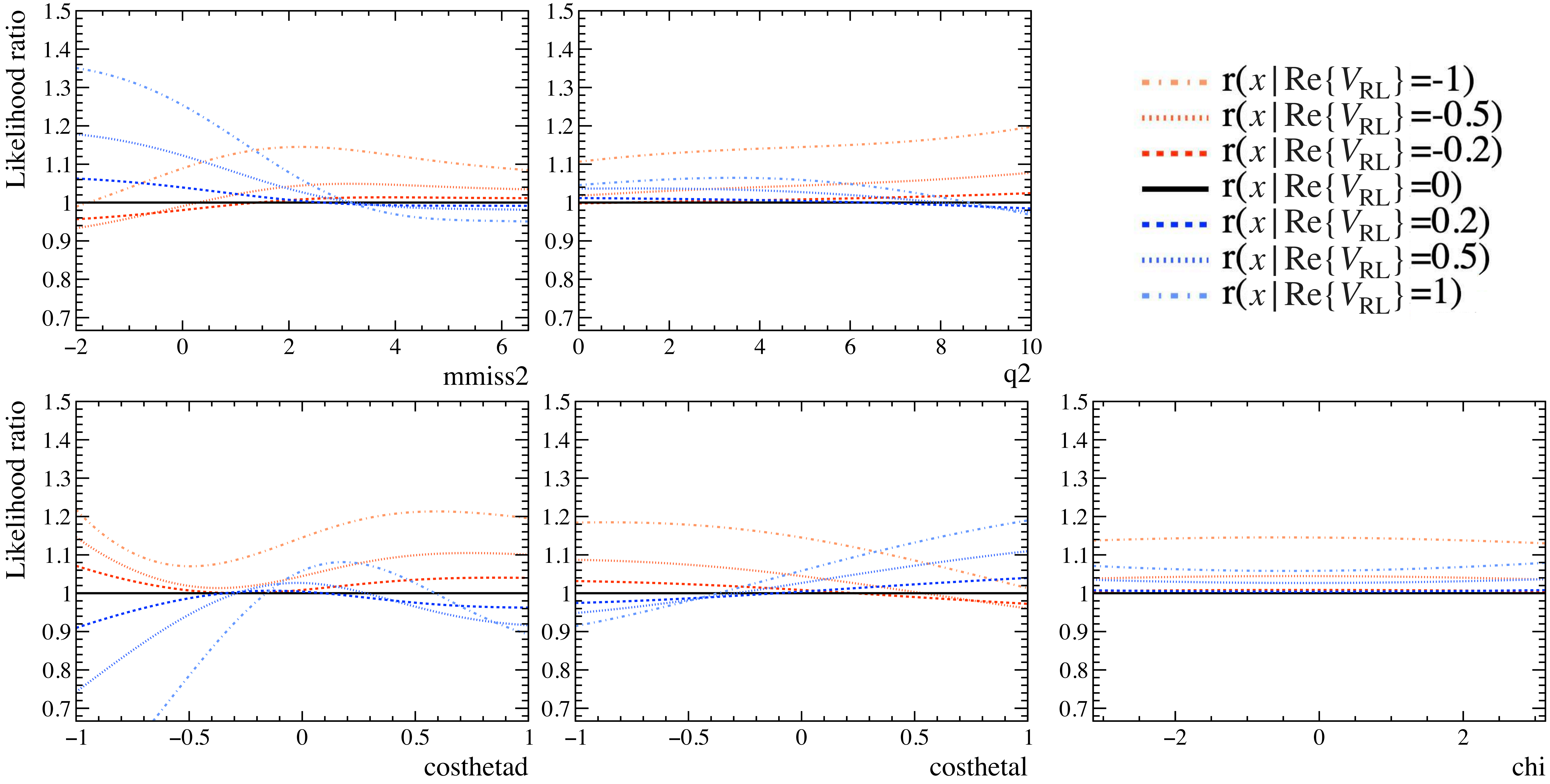
$$\mathscr{L}\left(\vec{x}\,|\,\theta\right) = \frac{-1}{S_w}\sum_i w'^{i}_{\mathrm{NP}} \log s\left(\vec{x}_i\,|\,\theta\right) + w'^{i}_{\mathrm{SM}} \log\left(1 - s\left(\vec{x}_i\,|\,\theta\right)\right) \quad \text{where} \quad S_w = \sum_i w'^{i}_{\mathrm{NP}} + w'^{i}_{\mathrm{SM}}$$

- Optimise with Adam, hyperparameters not formally tuned but are in a good ballpark:
  - Run for $1\mathrm{k}$ epochs with a learning rate of 0.005, include small L2 regularisation ($\lambda = 0.0001$)
- The likelihood ratio can be constructed from the NN score

$$r(\theta) = -\sum_i \log\left(\frac{p_{\mathrm{NP}}\left(\vec{x_i}\,|\,\theta\right)}{p_{\mathrm{SM}}\left(\vec{x_i}\right)}\right) = -\sum_i \log\left(\frac{s\left(\vec{x_i}\,|\,\theta\right)}{s\left(\vec{x_i}\,|\,0\right)}\right)$$

Probing the learned likelihood ratio

- Computing likelihood ratio in each feature for given values of θ gives insight into learned ratio:



Applying the learned likelihood ratio

- From trained models to likelihood (*see right*):
  1. Obtain likelihood ratio from NN output, *e.g.*, binding function calling the NN
  2. Convert likelihood ratio to PDF with `RooWrapperPDF`
     - Since we know sum of weights for given θ, $I(\theta)$, we can extend PDF with $I(\theta)$ as yield
  3. Create negative log-likelihood from PDF and dataset with `createNLL` method

```python
# Compute the learned likelihood ratio
llhr_learned = ROOT.RooFit.bindFunction(
    "MyBinFunc", learned_likelihood_ratio, x_var, mu_var)

# Create the learned pdf and NLL sum based on the learned likelihood ratio
pdf_learned = ROOT.RooWrapperPdf(
    "learned_pdf", "learned_pdf", llhr_learned, True)

nllr_learned = pdf_learned.createNLL(obs_data)
```

RooFit tutorial [rf615_simulation_based_inference.py]()

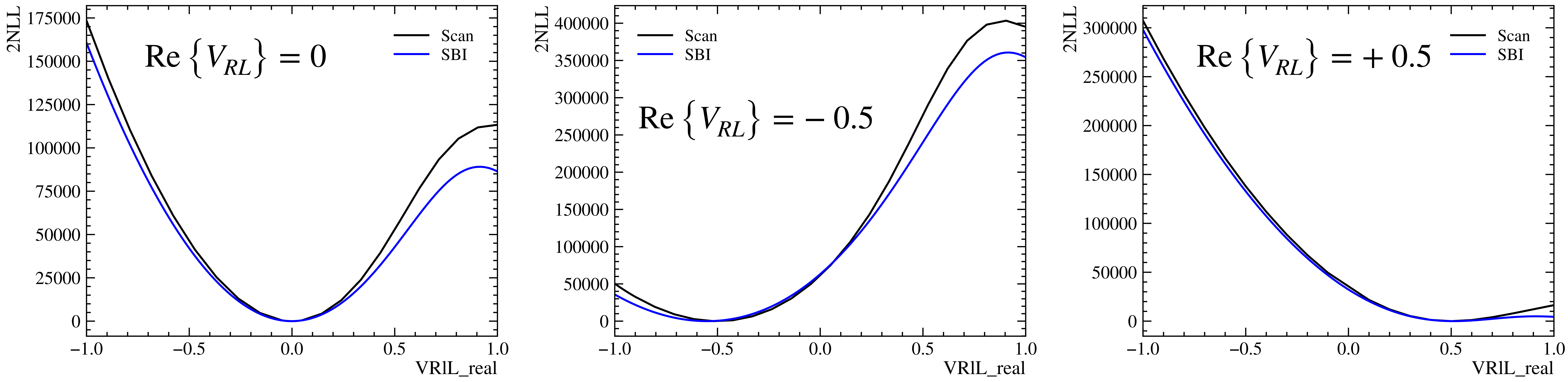
- This can then be manipulated as any RooFit likelihood object, *e.g.*, applying minimiser
- Perform Asimov test to check that NLL is sensible:
  1. Generate per-event weights for all events at a given fixed value of $\theta = \theta_{\text{gen}}$
  2. Create NLL for dataset with these weights applied
  3. Minimise and check that retrieved value θ_{min} consistent with θ_{gen}

Incorporating a “classical” comparison

- We can go one further than checking the minimised value of θ likelihood and perform a similar test for a “classical” fit setup
- Take a fitter built on pyhf, HAMMER and redist-Hammer, in use for $b \to c\tau^{-}\overline{\nu}_{\tau}$ WC global fits
  - Fit takes HAMMER histogram of 5 bins in each of M^2_{miss}, $\cos\theta_d$, $\cos\theta_l$ and χ^2, and 4 bins in q^2
  - Uses HAMMER to provide modifications of histogram passed to pyhf
  - Many thanks also to Marco for providing the setup for this project
- As in Asimov scan for SBI likelihood, apply fitter to sample with weights for a given θ_{gen}
- Aim to check that likelihood curves for SBI and classical fit are consistent

Asimov studies for SBI and classical fits

- Performing the Asimov scans for both fits:
  - Training dataset used to construct template for fit
  - Both likelihood curves computed using fits to test dataset

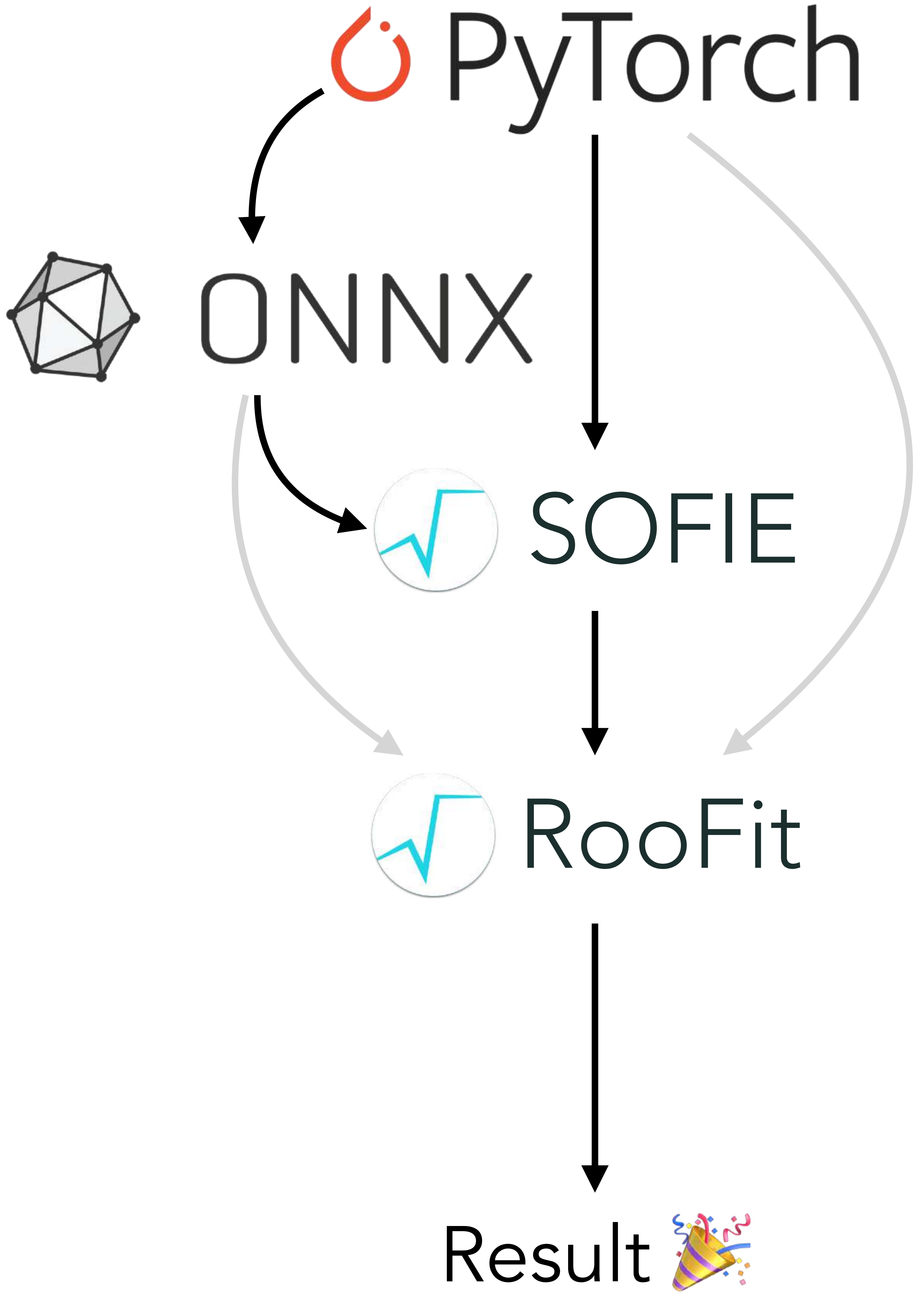


- Generally good agreement between the two fits
- Statistical power of SBI fit strongly dependent on quality of NN training

Automatic differentiation in this workflow

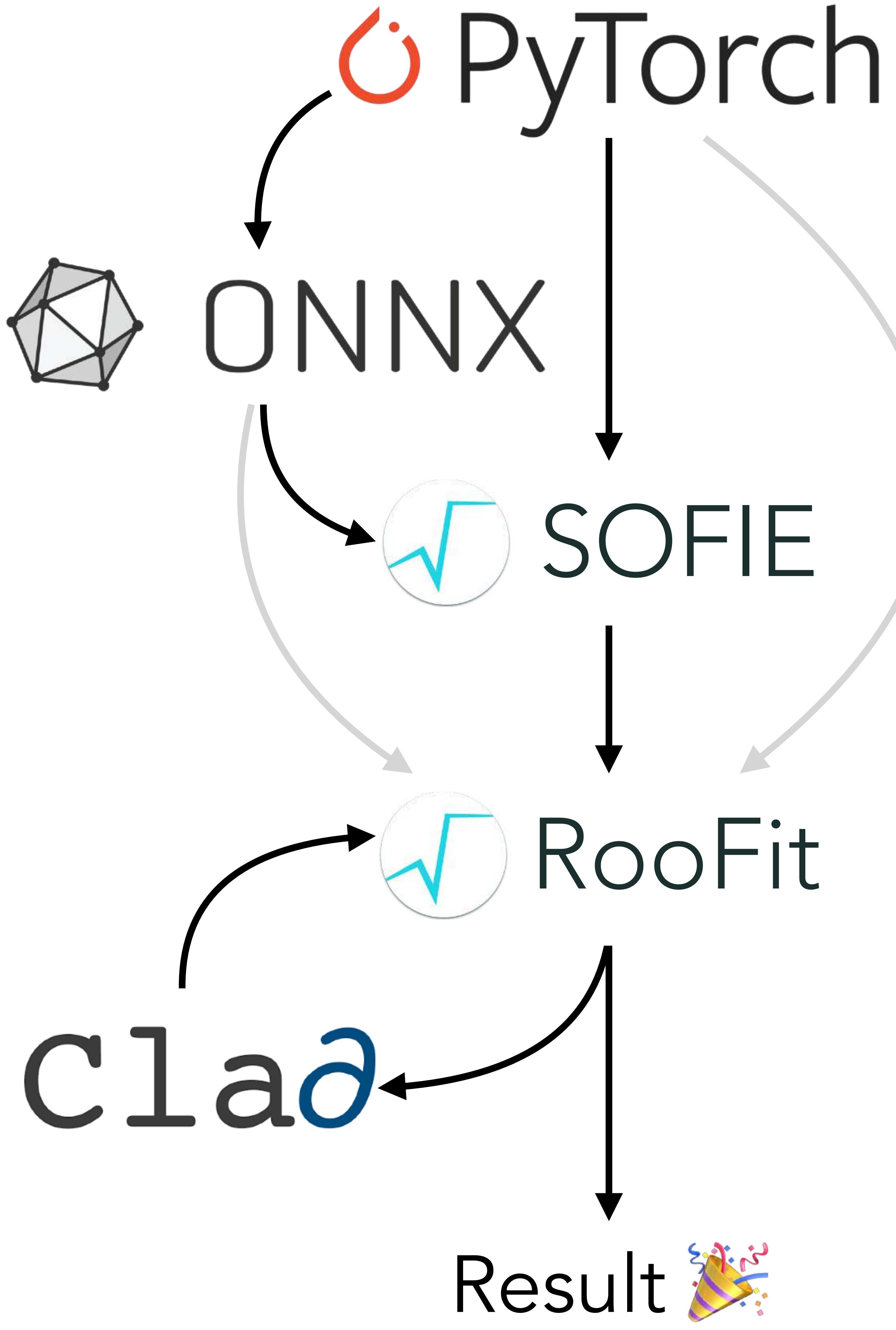
Converting models to code with SOFIE

- Previous RooFit examples involve directly wrapping the call of some Python function
  - Can use PyTorch or ONNXRuntime for this, depending on saved format of model
- Alternative is to use the TMVA SOFIE framework to convert a saved model (ONNX or PyTorch format) into a C++ header
- Can load these headers with ROOT and use `TFormula`/`RooFormulaVar` to manipulate NN in RooFit
- This implementation is at least as performant as wrapped calls to a PyTorch model



Converting models to code with SOFIE

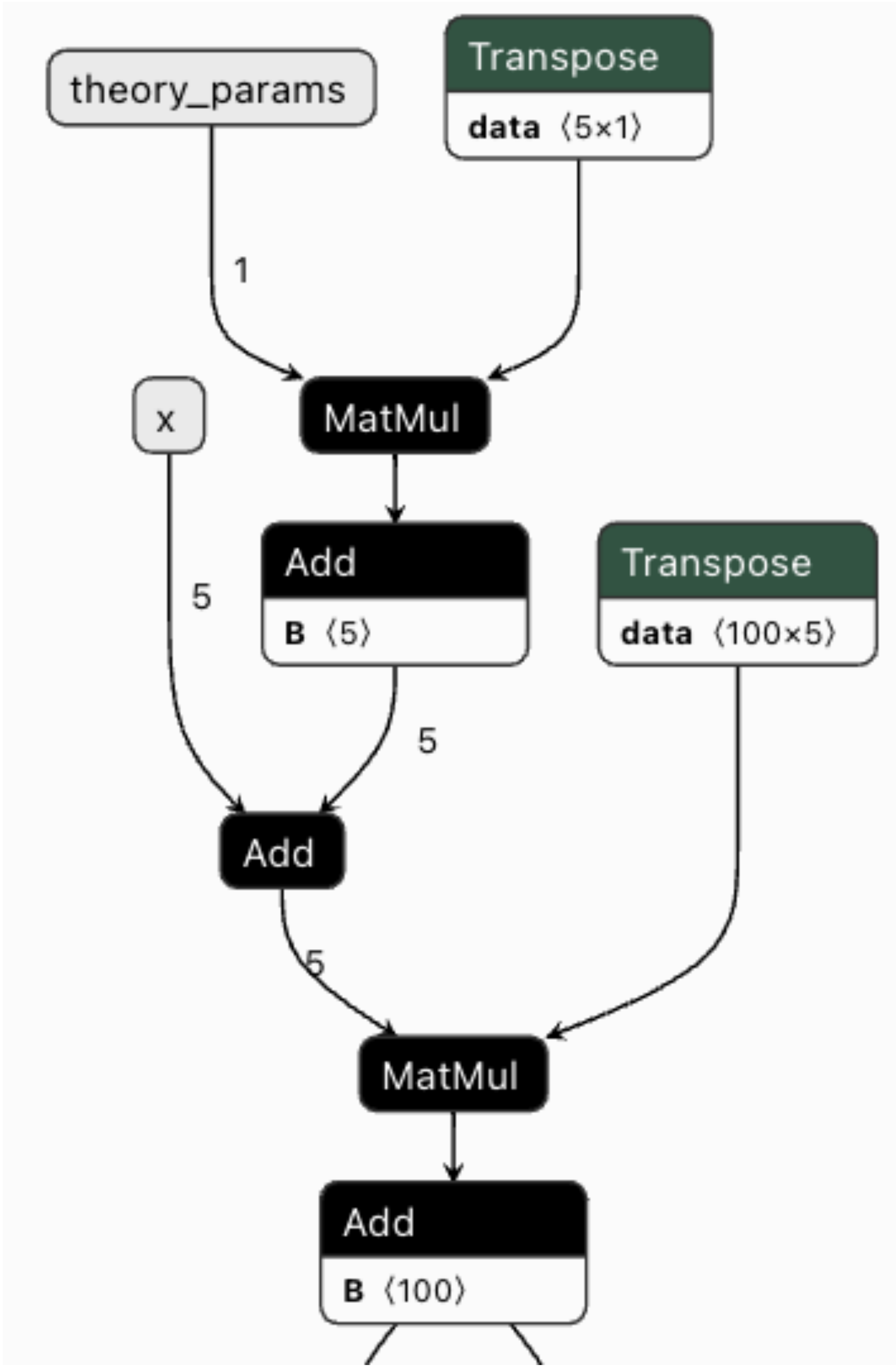
- Previous RooFit examples involve directly wrapping the call of some Python function
  - Can use PyTorch or ONNXRuntime for this, depending on saved format of model
- Alternative is to use the TMVA SOFIE framework to convert a saved model (ONNX or PyTorch format) into a C++ header
- Can load these headers with ROOT and use `TFormula`/`RooFormulaVar` to manipulate NN in RooFit
- This implementation is at least as performant as wrapped calls to a PyTorch model
- But we should be able to make this even faster…



Automatic differentiation for fast inference

- Would like to use Clad to perform automatic differentiation (AD) of code generated by SOFIE

model.pth/model.onxx



...

SOFIE

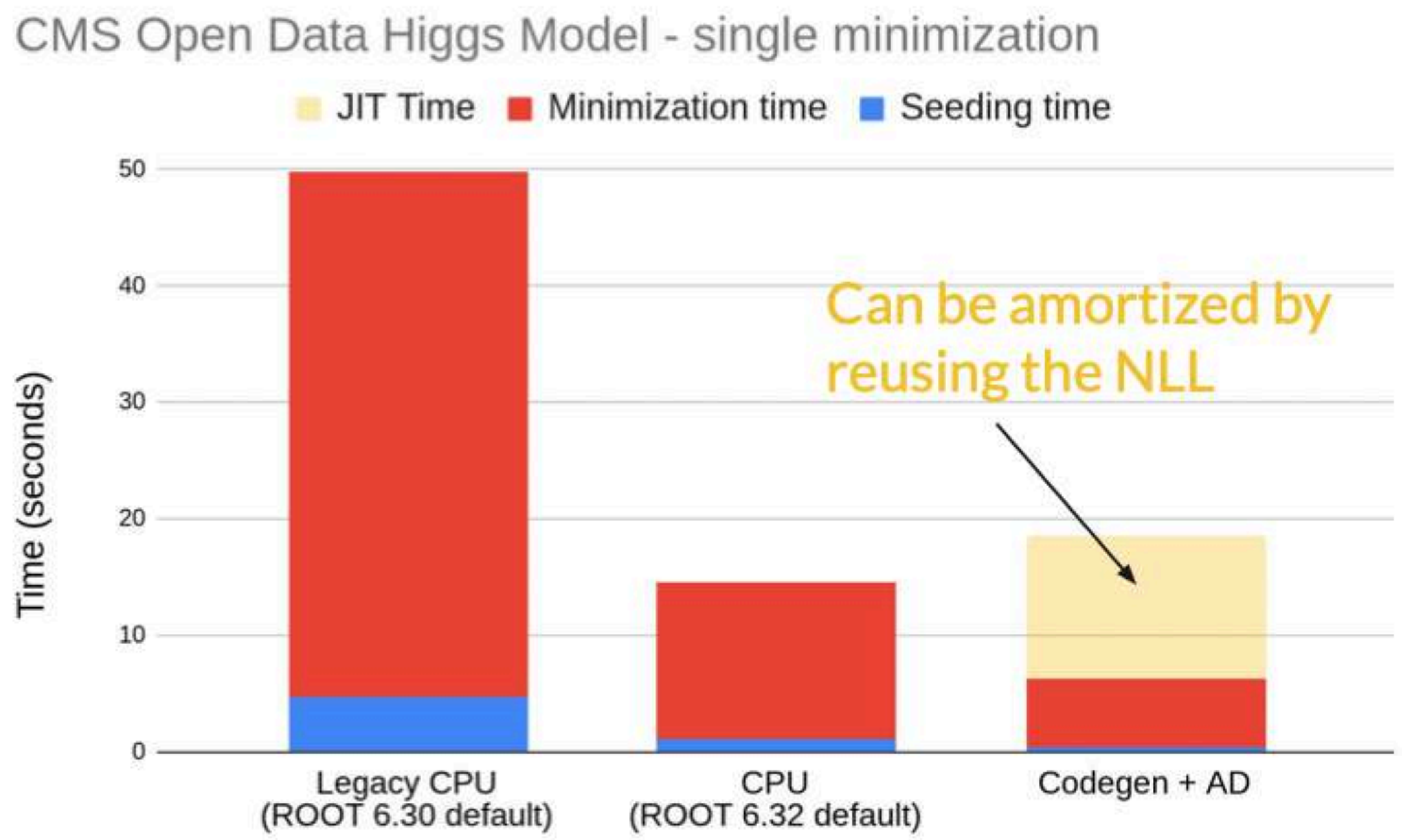
model.hxx

```
std::vector<float> infer(float* tensor_x,float* tensor_theory_params){

//--------- Gemm
   TMVA::Experimental::SOFIE::Gemm_Call(tensor_linear, false, false, 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] ;
   }

//--------- Gemm
   TMVA::Experimental::SOFIE::Gemm_Call(tensor_linear_1, false, false,
   100, 1, 5, 1,tensor_val_2, tensor_add, 1,tensor_base_model0bias);

//------- Sigmoid -- 5
   for (int id = 0; id < 100 ; id++){
      tensor_val_4[id] = 1 / (1 + std::exp( - tensor_linear_1[id]));
   }

//------- Mul
   for (size_t id = 0; id < 100 ; id++){
      tensor_silu[id] = tensor_linear_1[id] * tensor_val_4[id] ;
   }

//--------- Gemm
   TMVA::Experimental::SOFIE::Gemm_Call(tensor_linear_2, false, false,
   100, 1, 100, 1,tensor_val_5, tensor_silu, 1,tensor_base_model4bias);

//------- Sigmoid -- 9
   for (int id = 0; id < 100 ; id++){
      tensor_val_7[id] = 1 / (1 + std::exp( - tensor_linear_2[id]));
   }
```

...

Clad

Generated C++ code for gradients of model (included in likelihood gradients)



[J. Rembser, ICHEP 2024]

Potential analysis speedups?

Status of AD in the workflow

- The aim is to have support in RooFit/Clad so that AD can be handled by RooFit `codegen`
  - Should allow users to simply construct their likelihoods and specify the `codegen` backend
  - AD should take place all the way down from likelihood to ML inference
- Where do we stand on this? There has been quite some action on this within ROOT (thanks to Jonas for the significant work on this!):

Implement coverage of GeMM

[TMVA][SOFIE] Implement wrapper for `BLAS::sgemm_` call #18349
Merged

[TMVA][SOFIE] Implement custom pullback for Gemm operator and corresponding test #18364 Merged

[math] Forward declare `sgemm_` in custom derivatives header #18476
Merged

Refactoring of code emitted by SOFIE

[TMVA][SOFIE] Small improvements necessary for AD support #18341
Merged

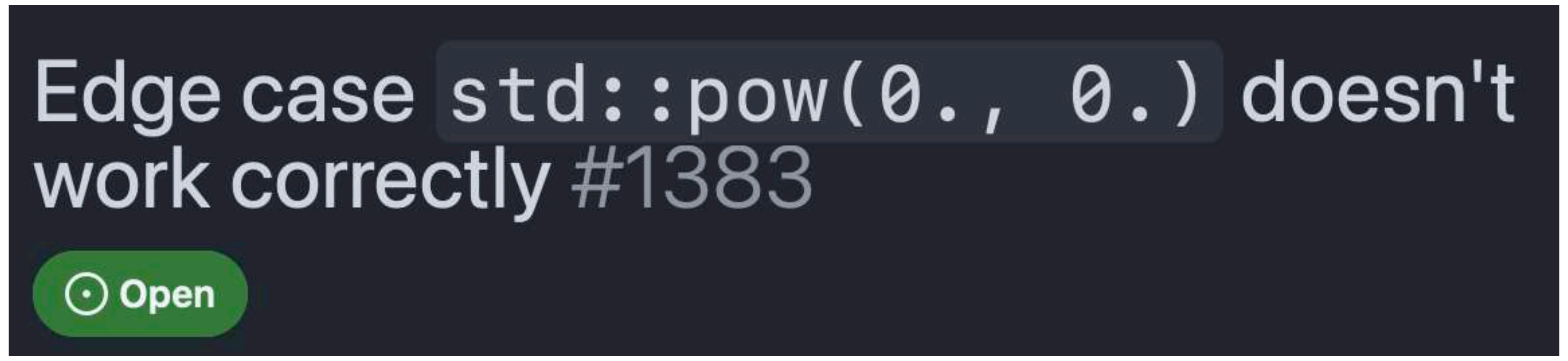
[tmva][sofie] Add overload for inference code that takes output params #18399
Merged

Implement coverage for RooWrapperPDF

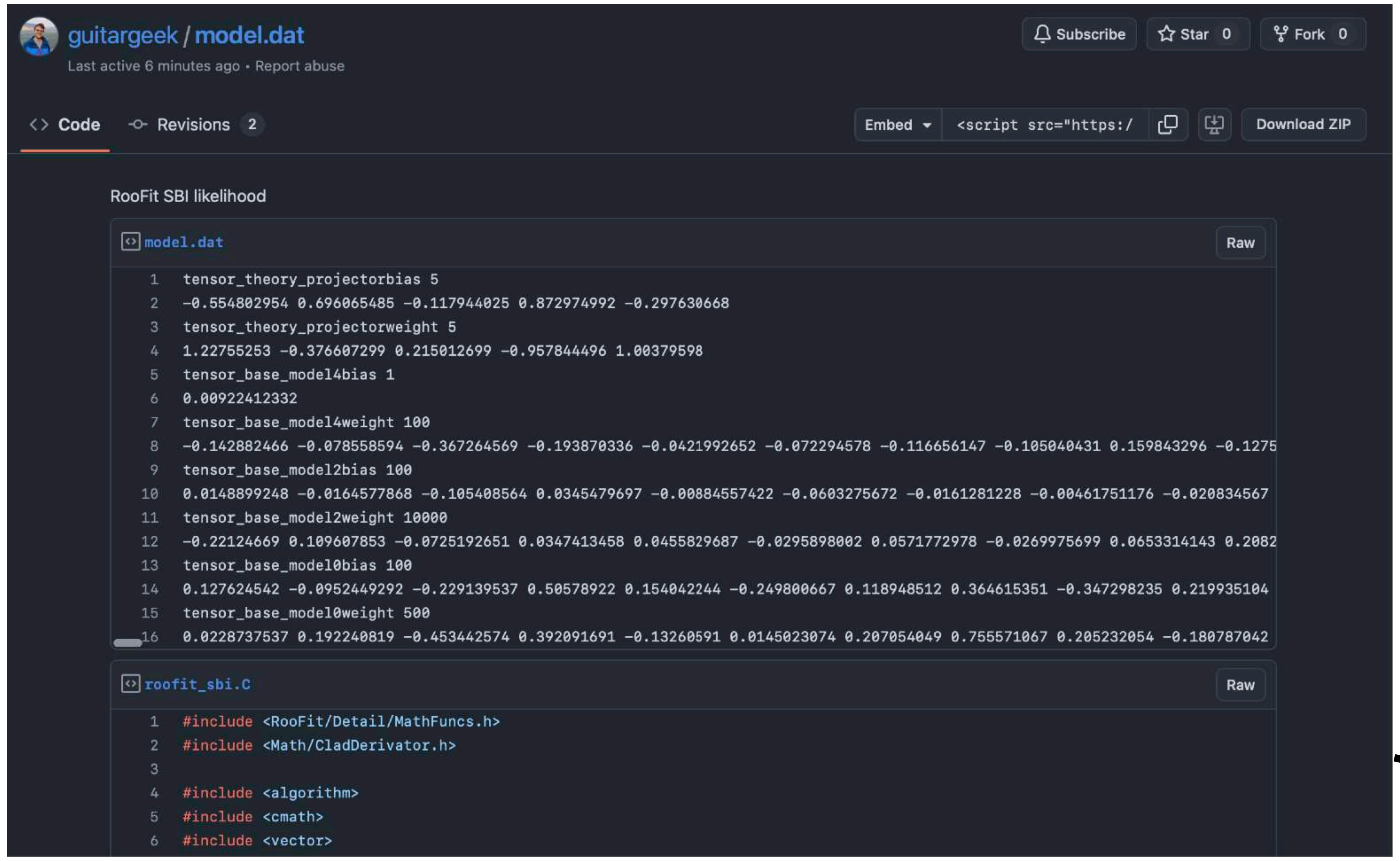
[RF] Implement codegen support for the RooWrapperPdf #18699
Merged

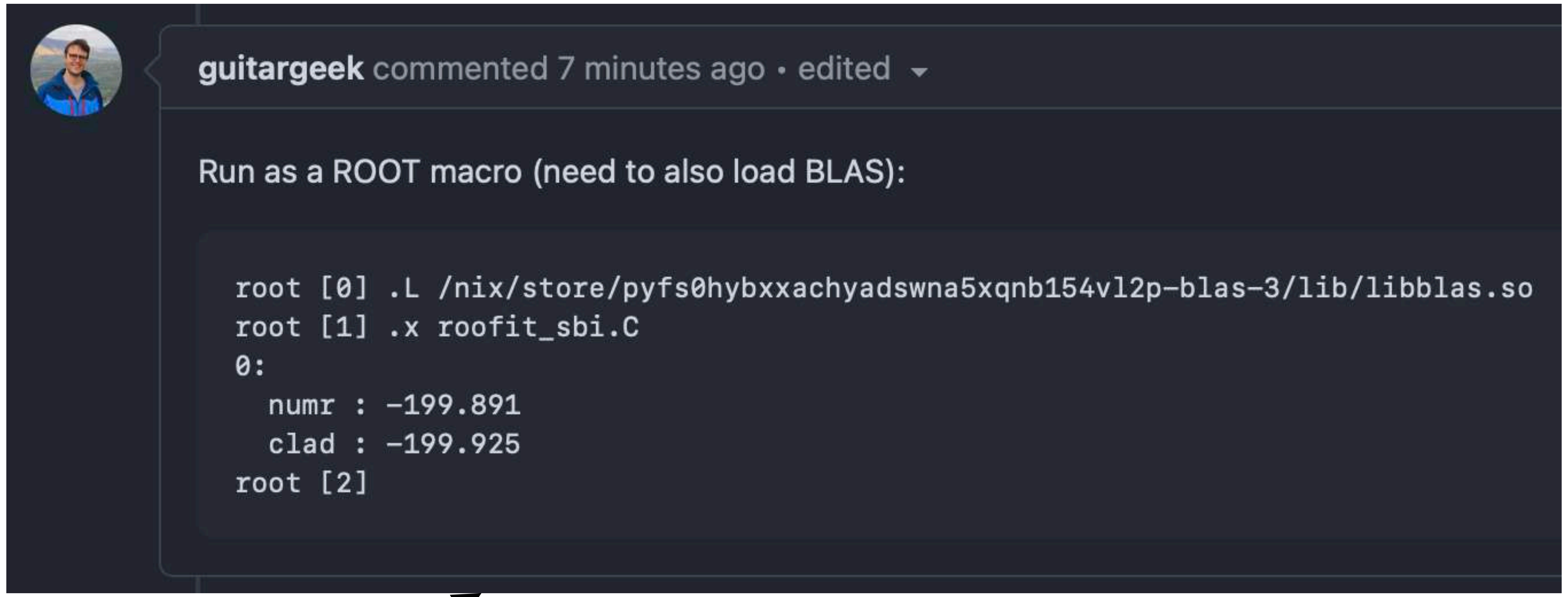
What still needs to happen?

- With some final tweaks, we can integrate fully with `codegen` functionality
- Issues opened in Clad covering the final items required for this to work



- For a full reproducer of the RooFit SBI likelihood, see the following gist:





What next?

- With this in place, should be able to simply enable AD with `codegen`
- Would like to do a comparison of performance (timing) in the statistical analysis:
    1. With a bound Python call to PyTorch
    2. With SOFIE code and AD disabled
    3. With SOFIE code and AD enabled
- Preliminary checks show that 1. and 2. are roughly consistent
- Expectation is that, for analysis size datasets, *i.e.*, upwards of $\mathcal{O}(10^6)$, JIT time becomes a small fraction of total timing in 3., and that this therefore quickly surpasses 1. and 2.
    - This is only a fair comparison if TBR optimisation is enabled

Conclusion

Conclusion

- Project well underway, with most of our goals now complete:
  - LHCb SBI fitting demonstrator workflow constructed
  - Comparisons to the current state-of-the-art established
  - AD of ML inference calls almost enabled
- Still work to be done in the final week of the project:
  - Incorporate efficiency effects in the train/test samples and start to train NNs with combinations of WC terms
  - Training and statistical inference with larger samples sizes (GPU training set up this week should enable this)
  - Consolidate the work of the last few months into a battery of statistical/performance comparisons against the current state-of-the-art

Thank you for your attention

Any questions?

Backup

The LHCb experiment

R. Aaij et al. JINST 19 (2024) P05065