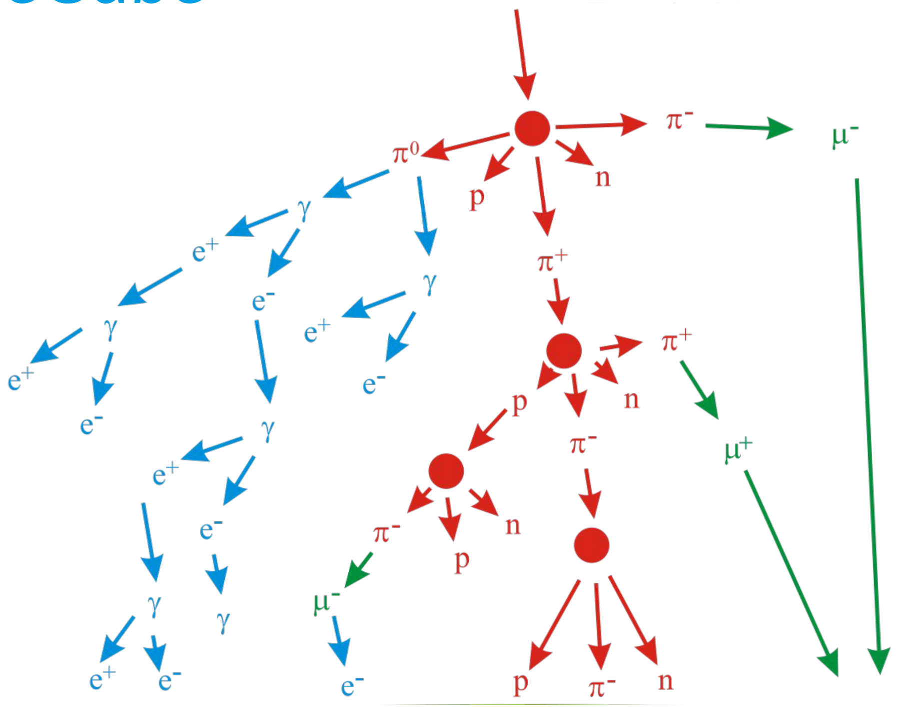
Towards Reducing the Computation Time for Air-Shower Simulations at IceCube

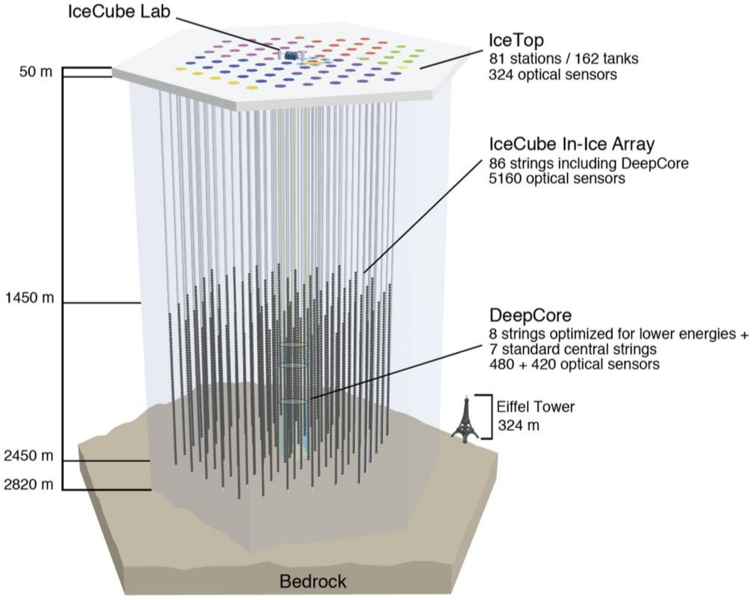
Navid K. Rad, Jakob van Santen

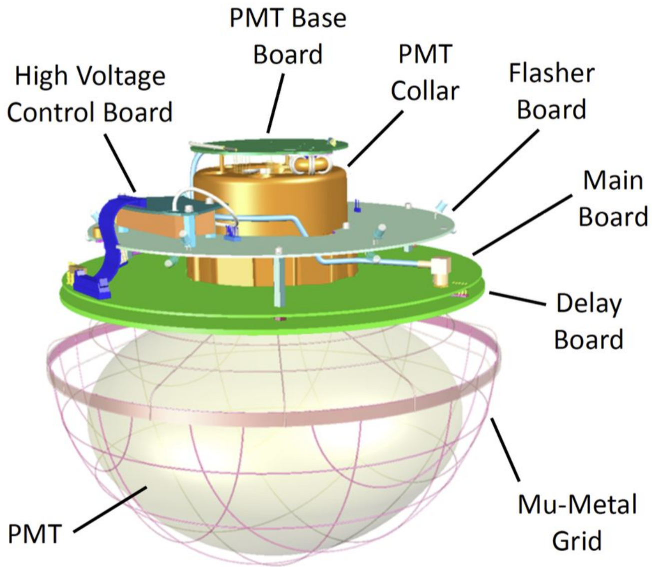
Adaptive Sampling Hackathon
May 06, 2024



IceCube Neutrino Observatory

- Cubic-kilometer Neutrino detector
- **5160** Digital Optical Modules (DOM) in the glacial Antarctic ice **(depths of 1450-2450m)**
- Each DOM: a 25 cm PMT, high voltage power supply and digi & com. electronics

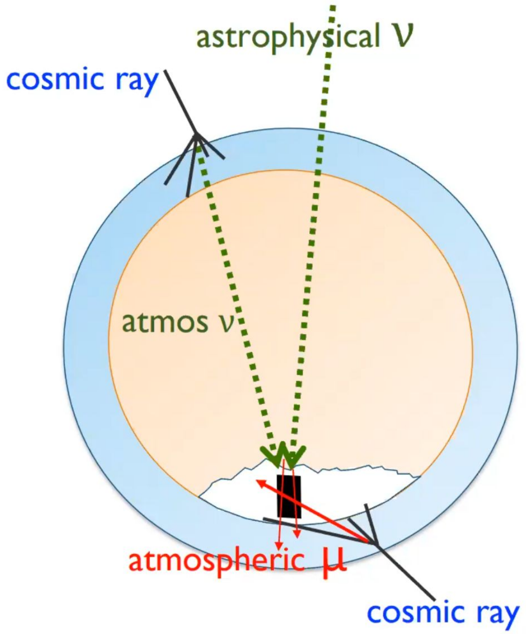




Background for high energy astrophysics neutrinos

- Atmospheric Muons and neutrinos:
  - Produced from the interaction of cosmic rays with the atmosphere
  - Even @ 1.5 km below ice, detected at high rates!
    - atmospheric muons: ~1000/s ~ $O(10^{3})$ Hz
    - atmospheric neutrinos: ~ 1/5min ~ $O(10^{-3})$ Hz
    - **astrophysical neutrinos ~ 1/month ~$O(10^{-6})$ Hz**
  - Need to reduce background by factors 10^{3}-10^{9}

- The challenge is the rare background events:
  - Muon + few or no other low energy muons
  - a lone atmospheric neutrino

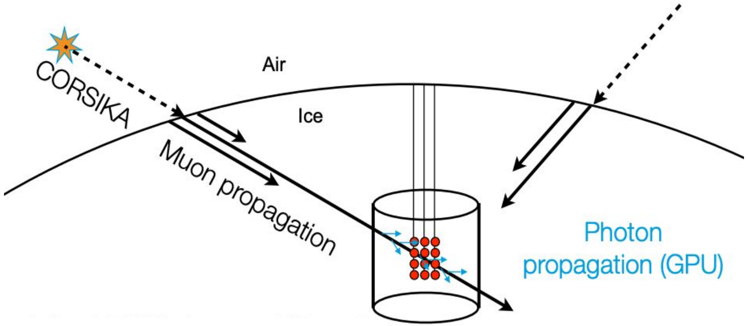


⇒ **Very large simulations are needed!**

IceCube simulation chain:

Air Shower Generation (Corsika) → Muon propagation → Photon propagation (GPU) → Detector response → Trigger & Filter (L2, L3,...)

- CORSIKA simulates Air Shower:
  - Interaction of the primary cosmic rays with atmospheric nuclei
  - EM and Hadronic showers, $\pi^{\pm}$, π^{0} , K, **μ, *ν*** are produced and propagated to the ice surface
- Muons are propagated through the ice:
  - account for the stochastic energy losses
- Photon propagation:
  - ice properties depend on depth ⇒ photons need to be tracked individually...
  - **The most computationally intensive part of simulation (but parallelizable)**



IceCube simulation chain:

Air Shower Generation (Corsika) → Muon propagation → Photon propagation (GPU) → Detector response → Trigger & Filter (L2, L3,…)

The Problem:

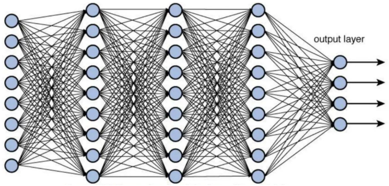
a few % of generated air-showers pass the L2 filtering

a few % L2 filtered pass the L3 selection

⇒ **lots of wasted CPU & GPU power**

Taking a shortcut?

Air Shower Generation (Corsika) → Muon propagation → Photon propagation (GPU) → Detector response → Trigger & Filter (L2, L3,...)

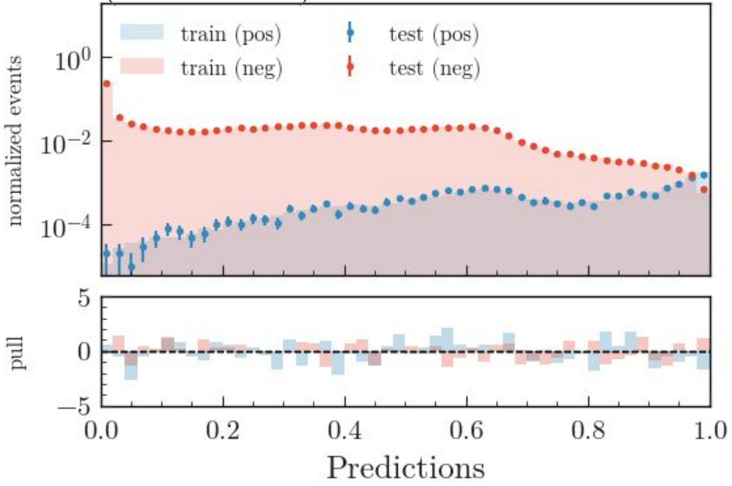


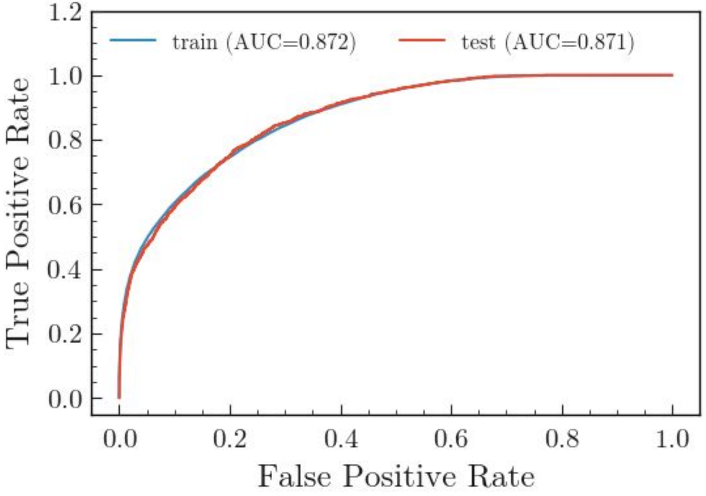
Use a Neural Net to try to predict probability that a certain air shower will pass the Filtering

Model Predictions (based only on **primaries**)

- Train and hyper tune a NN on the available simulations
  - Balance training set by uniform undersampling the majority class
  - Positive: shower passed the filtering
  - Negative: shower failed

normalized to the "test population" (unbalanced)

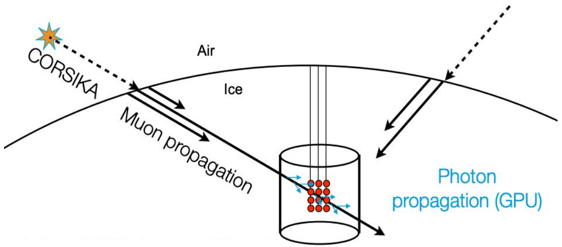


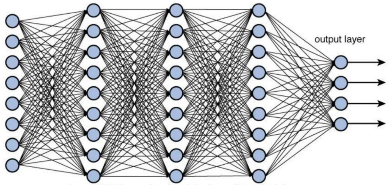


⇒ Peak at 0! turn out to be events with no muons produced in the shower

Can we do better using the **muon** information?

Air Shower Generation (Corsika) → Muon propagation → Photon propagation (GPU) → Detector response → Trigger & Filter (L2, L3,...)



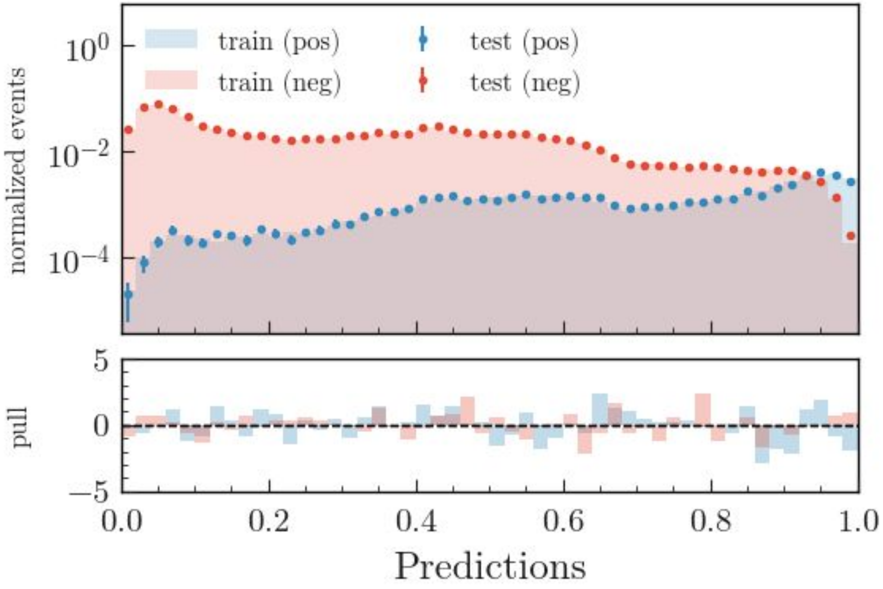


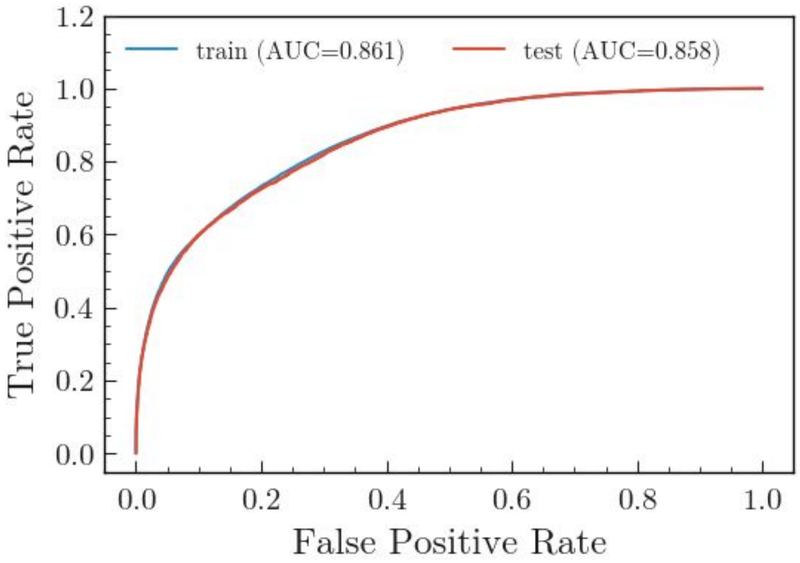
Use a Neural Net to try to predict probability that a certain air shower will pass the Filtering

Require at least 1 muon in the shower

- Use muon/muon bundle information as well:
- Remove events without a muon from training

normalized to the “test population”





⇒ no more “low hanging fruits” so the model can focus on the difficult cases

Quantifying the gain in computation time

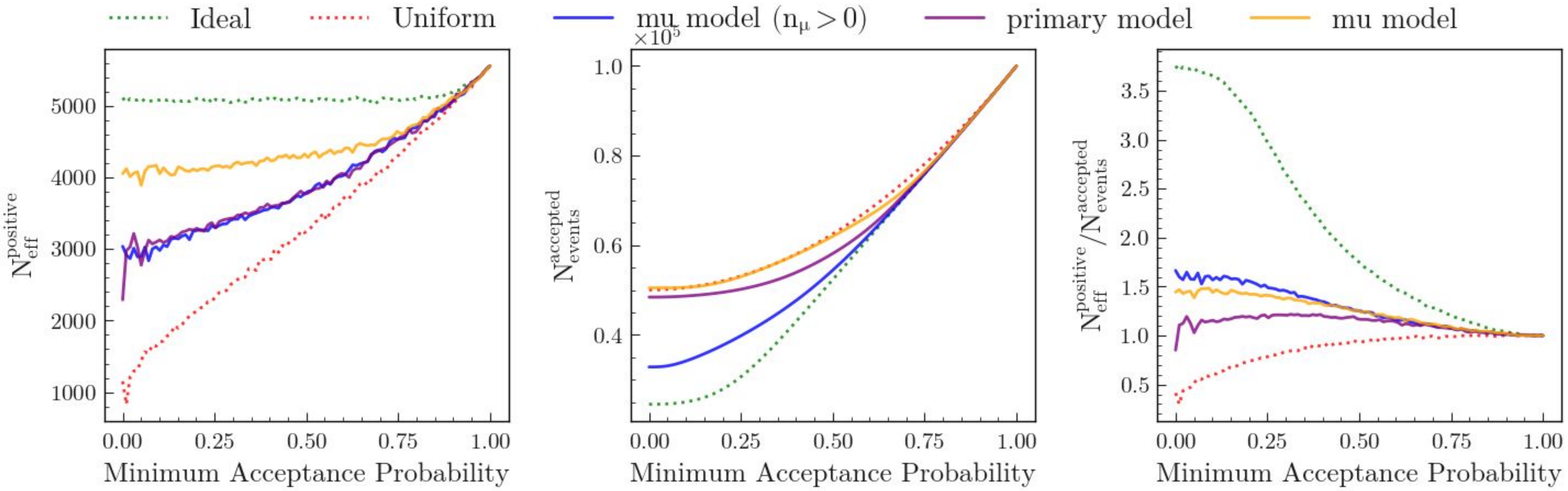
- **Assumptions:**
  - air shower generation time << simulation time
  - evaluation time of the model << simulation time
  - computation time goes as $N_{accepted}$
- **Method:**
  1. Use predicted score (**p_i**) as "**acceptance probability**" of the event
  2. Assign a corresponding weight to each event as **$w_i=1/p_i$**
  3. Scan the minimum acceptance probability threshold (avoid very large weights)
- **Simple "speed up" Metric:** $\mathrm{speedup} = \mathrm{N}_{\mathrm{eff}}^{\mathrm{positive}} / \mathrm{N}_{\mathrm{accepted}}$
  - N_{eff}: effective sample size of the **accepted positive events** (size of an unweighted sample that would have same relative uncertainty)

$$N_{eff}(w) = \frac{\left(\sum_i w_i\right)^2}{\sum_i w_i^2}$$

  - $N_{accepted}$: number of **accepted events** (sampled based on their p_i)

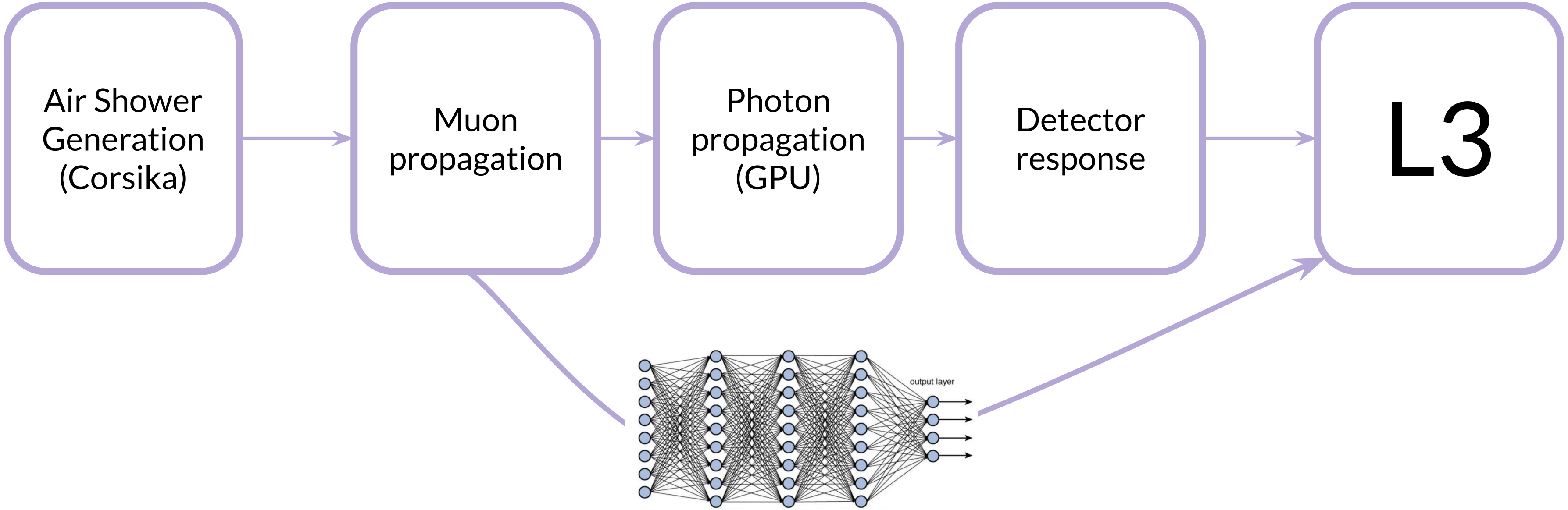
Potential gain in computation time

- **Nearly Ideal:** “prediction” based on truth information → Best case
- **Uniform:** “prediction” based on uniform distribution → Worst case



⇒ Improvement of about 1.5x compare to default scenario

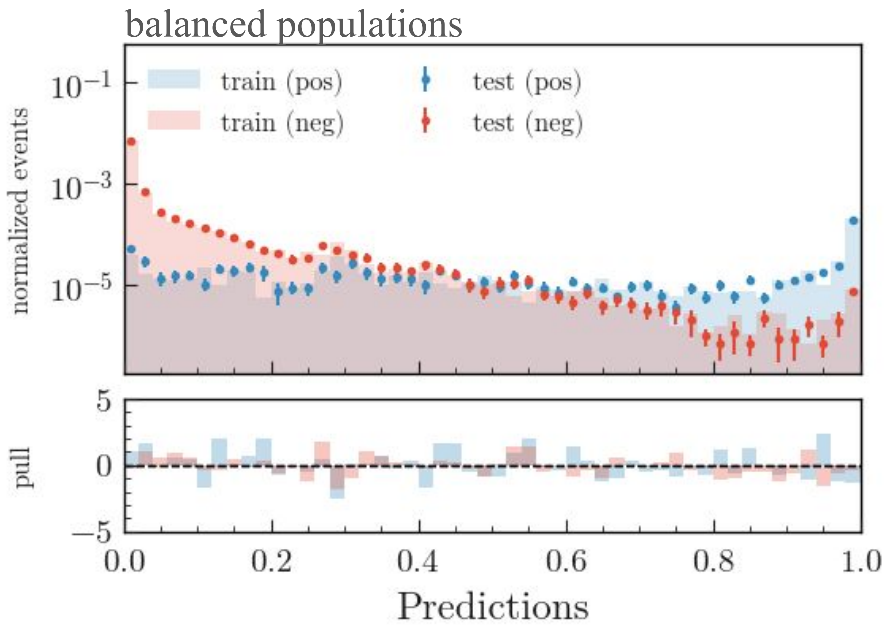
Let’s take it to the next level (Level3)

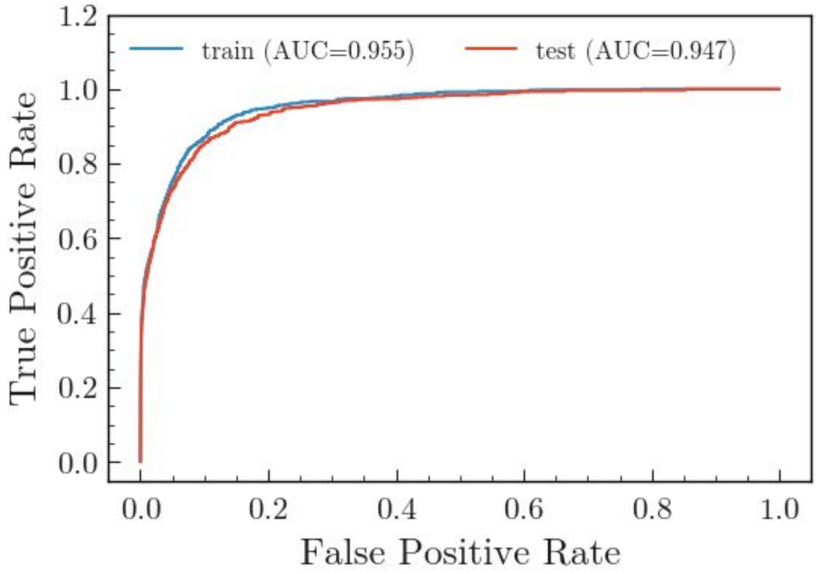


- L3 Filter:
  - Reconstruct cascades and tracks (here only cascades are used)
  - Closer to the analysis level.
  - Only 0.5% of L2 events are reconstructed cascades

Next filtering Level (L3)

- Much larger imbalance (1:10,000)
- challenging to get large enough sample for training

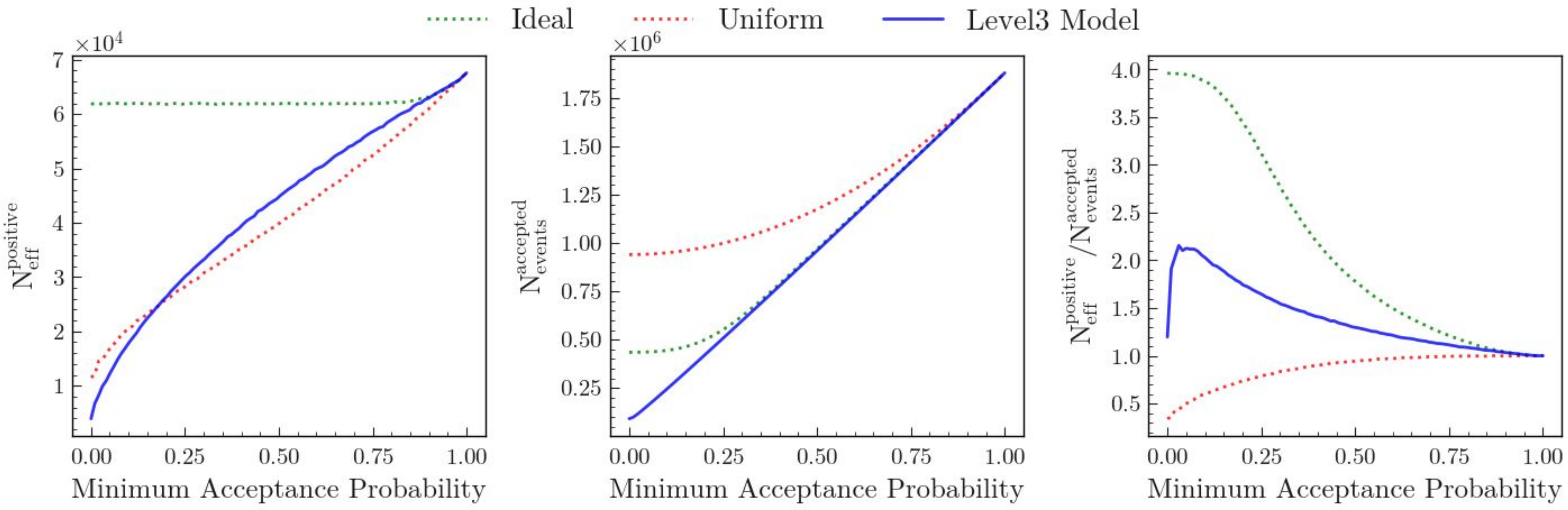




⇒ fresh results! still needs to be hypertuned

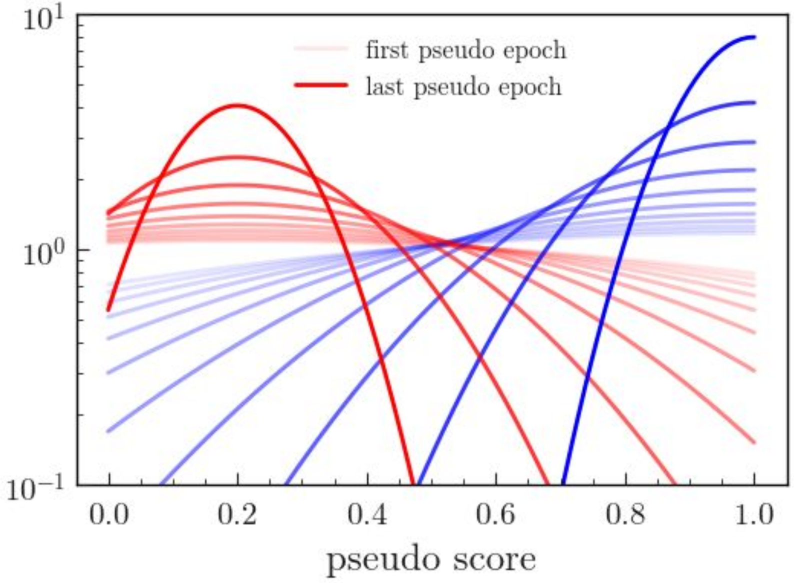
Potential gain in computation time (Level3 Model)

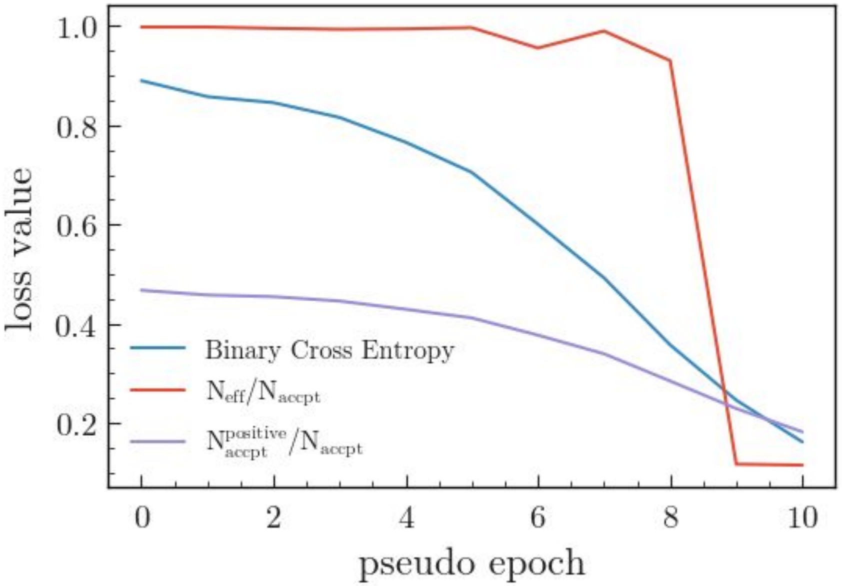
- **Nearly Ideal:** "prediction" based on truth information → Best case
- **Uniform:** "prediction" based on uniform distribution → Worst case



Playing with custom loss function?

- Toy test:
  - assign arbitrary “pseudo scores” to positive and negative events
  - different “degrees of separation” represent evolution of “pseudo epochs”
  - test the behavior of different loss functions





Summary and Outlook

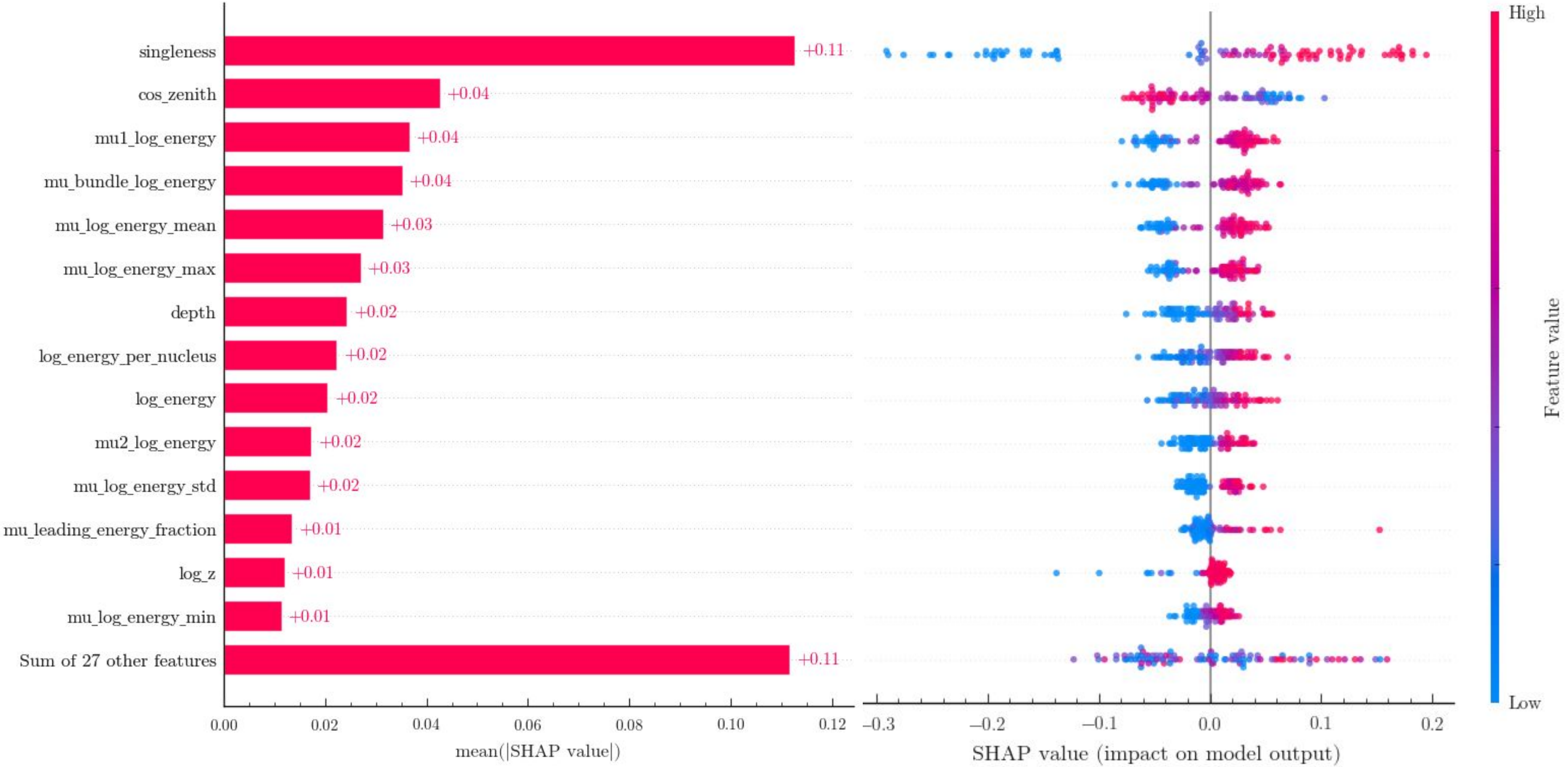
- Proof of concept that acceleration is possible!

- Current challenges:
  - very large imbalance in sample (~ 1:10,000 at L3)
    - means having to process and store lots of unused events.
    - possibly try different undersampling techniques
- Dedicated loss:
  - Q: How to deal with a loss function which depends on sample size (batch size dependent?)
  - Q: Current speed up metric requires sampling based on p_i values… how to incorporate “sampling” in the loss function?
    - Better to use a more realistic time estimates
    - Simulation time ~ number of photons

- Run through the full generation with and without adaptive sampling

Backup

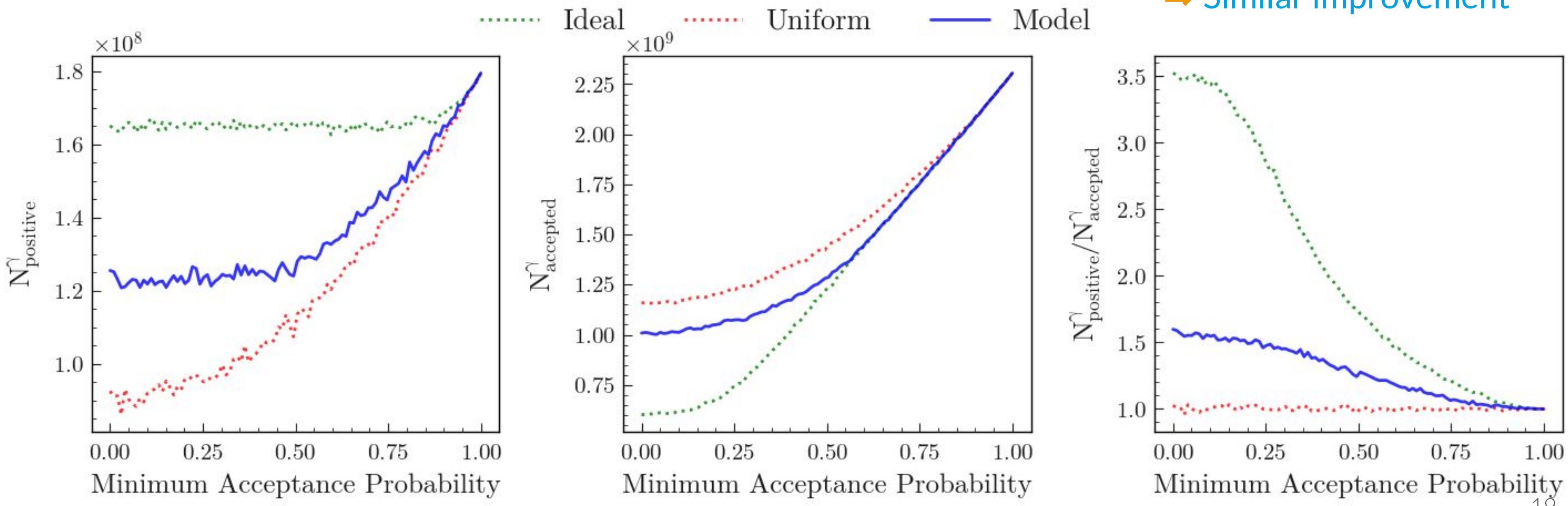
Feature importance: Shaply Values



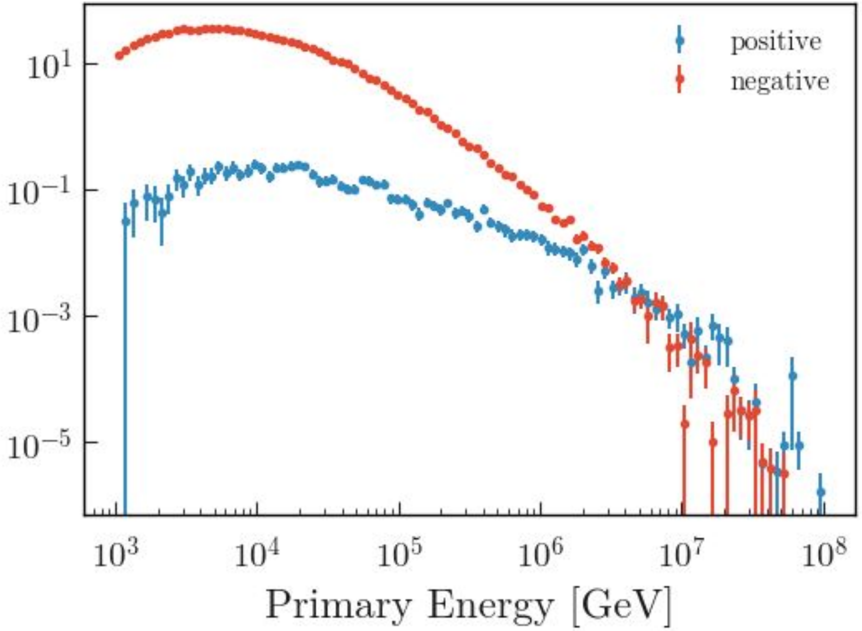
Potential gain in CPU time (slightly more realistic)

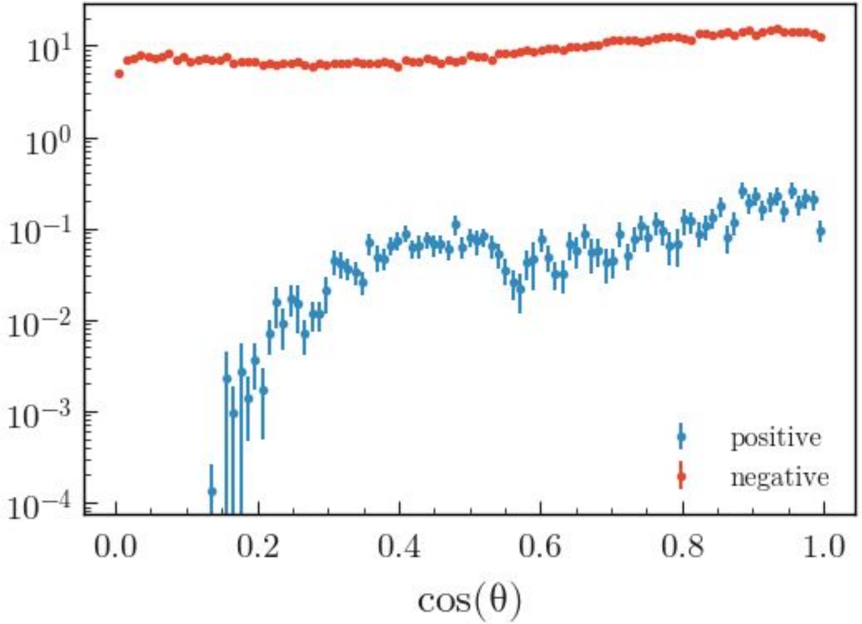
- Modify Assumptions:
  - CPU time ~~goes as $N_{accepted}$~~ goes as $N(\gamma)$
- Slight more realistic "speed up" Metric: $N_{\text{positive}}(\gamma)/N_{\text{accepted}}(\gamma)$
  - $N_{accepted}(\gamma)$: total number of photons in the accepted events
  - $N_{positive}(\gamma)$: total number of photons in the accepted positive events

⇒ Similar improvement



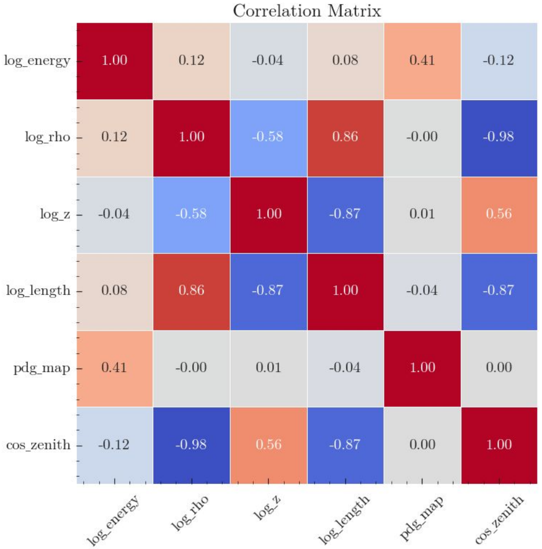
Primary Distributions



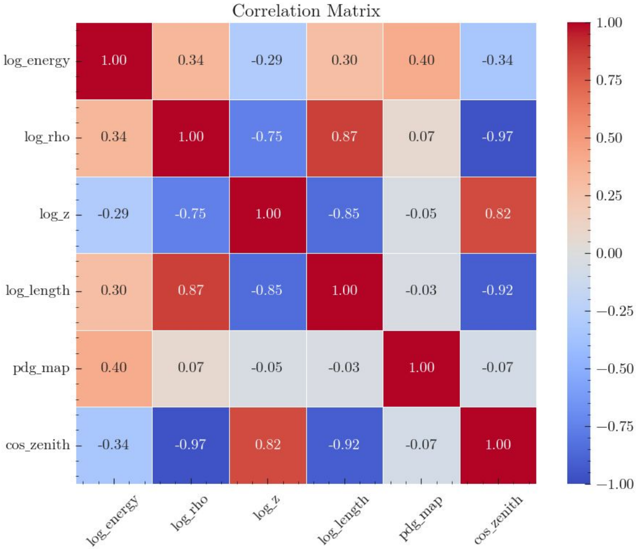


Primary Correlations

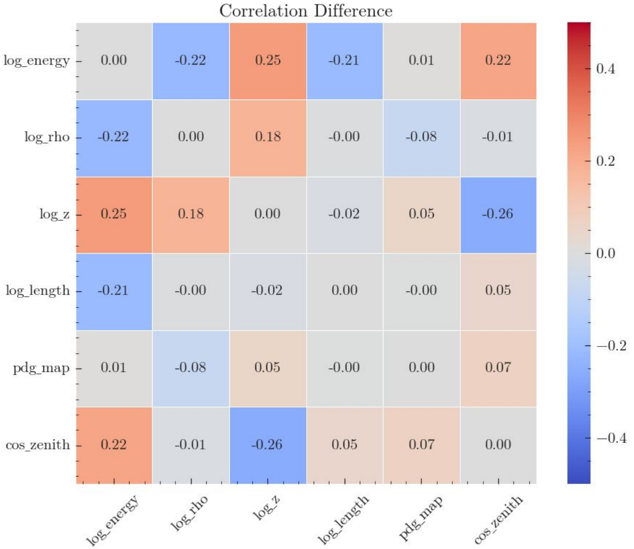
Negatives



Positives



Difference



IceCube simulation challenges:

- The challenge:
  - Majority of the simulated showers are not triggered and do not pass the initial filtering (~2%)
  - Lots of CPU+GPU time is wasted on showers that are thrown away, way before getting to the analysis level.

- Many attempted solutions:
  - Bias the generated distributions:
    - e.g. on average proton primaries end up with lower muon multiplicity
  - Parametrize the muon bundle properties
  - Hard energy cut: kill the shower if no particles about the energy threshold remain
  - Soft energy cut: rejection probability based on the expected number of muons above certain energy

⇒ Only work in specific cases, and only to some degree… need a more general solution!

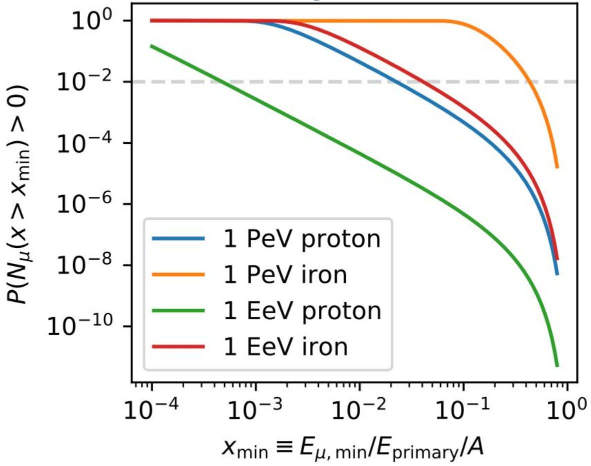
Current solutions

- Parametrize the problem:
    - MUGUN: generates muons based a parametrization of muon bundle properties under the ice

- Importance Sampling
    - Bias the generated distributions so more likely to pass the filtering
        - Primary compositions:
            - Proton primaries more likely to produce single high-energy muons
    - Apply minimum energy requirement for the muons in the shower:
        - hard requirement: kill the shower if no muons above a certain energy threshold (ICECUBE1)
        - soft requirement (muon biasing): reject the shower based on the probability of the primary of a given energy to produce a muon above a certain energy

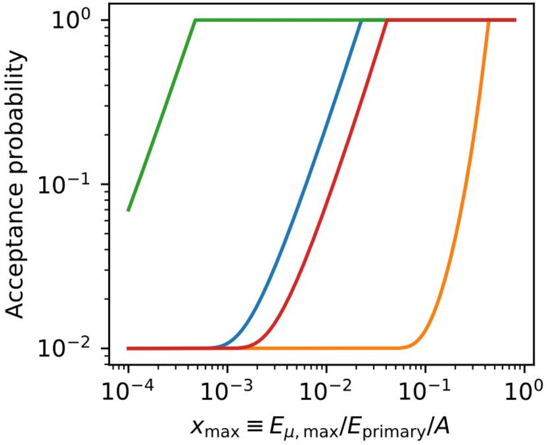
Muon Biasing (JVS)

JVS, Meager CHEP2023



- Elbert's formula gives the probability of having N muons above a certain energy threshold:

$$\left\langle N_\mu \left(\frac{E_\mu}{E_{\text{primary}}/A} > x_{\min} \right) \right\rangle = 14.5 \frac{A^2}{E_{\text{primary}} \cos\theta} x_{\min}^{-1.757} (1 - x_{\min})^{5.25}$$



- User specifies a "bias factor" (acceptance fraction)
- Muon energy threshold is chosen to match the probability of having at least 1 muon

$$1 - p_{\text{kill}}(x_\mu) = \begin{cases} 1, & x_\mu \geq x_{\min} \\ \frac{1 - \exp(-\langle N_\mu(x > x_{\min})\rangle)}{1 - \exp(-\langle N_\mu(x > x_\mu)\rangle)}, & x_\mu < x_{\min} \end{cases}$$

⇒ Could this be generalized?

```
Model: "Primary Hypertuned"
_______________________________________________________________________________
Layer                         | Size | Activation | Param #
===============================================================================
batch_normalization           | 8    |            | 32
Dense_0                       | 224  | leaky_relu | 2016
dropout                       | 224  |            | 0
Dense_1                       | 224  | leaky_relu | 50400
Dense_2                       | 224  | leaky_relu | 50400
Dense_3                       | 112  | leaky_relu | 25200
Dense_4                       | 112  | leaky_relu | 12656
Dense_5                       | 112  | leaky_relu | 12656
Dense_6                       | 56   | leaky_relu | 6328
Dense_7                       | 56   | leaky_relu | 3192
Dense_8                       | 56   | leaky_relu | 3192
Dense_9                       | 28   | leaky_relu | 1596
Dense_10                      | 28   | leaky_relu | 812
Dense_11                      | 28   | leaky_relu | 812
Dense_12                      | 14   | leaky_relu | 406
Dense_13                      | 14   | leaky_relu | 210
Dense_14                      | 14   | leaky_relu | 210
Dense_15                      | 8    | leaky_relu | 120
dense                         | 1    |            | 9
===============================================================================
Total params: 170,247
Trainable params: 170,231
Non-trainable params: 16
```