LArTPC differentiable simulator

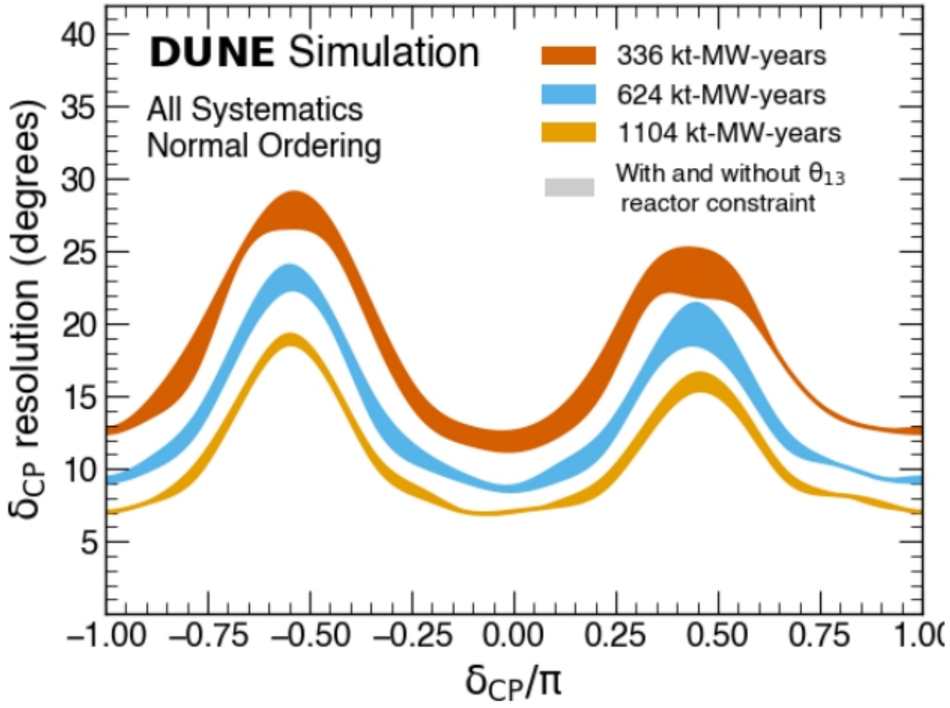
General update and looking at uncertainty propagation

Pierre Granger, 31 October 2025

Moving towards precision measurements

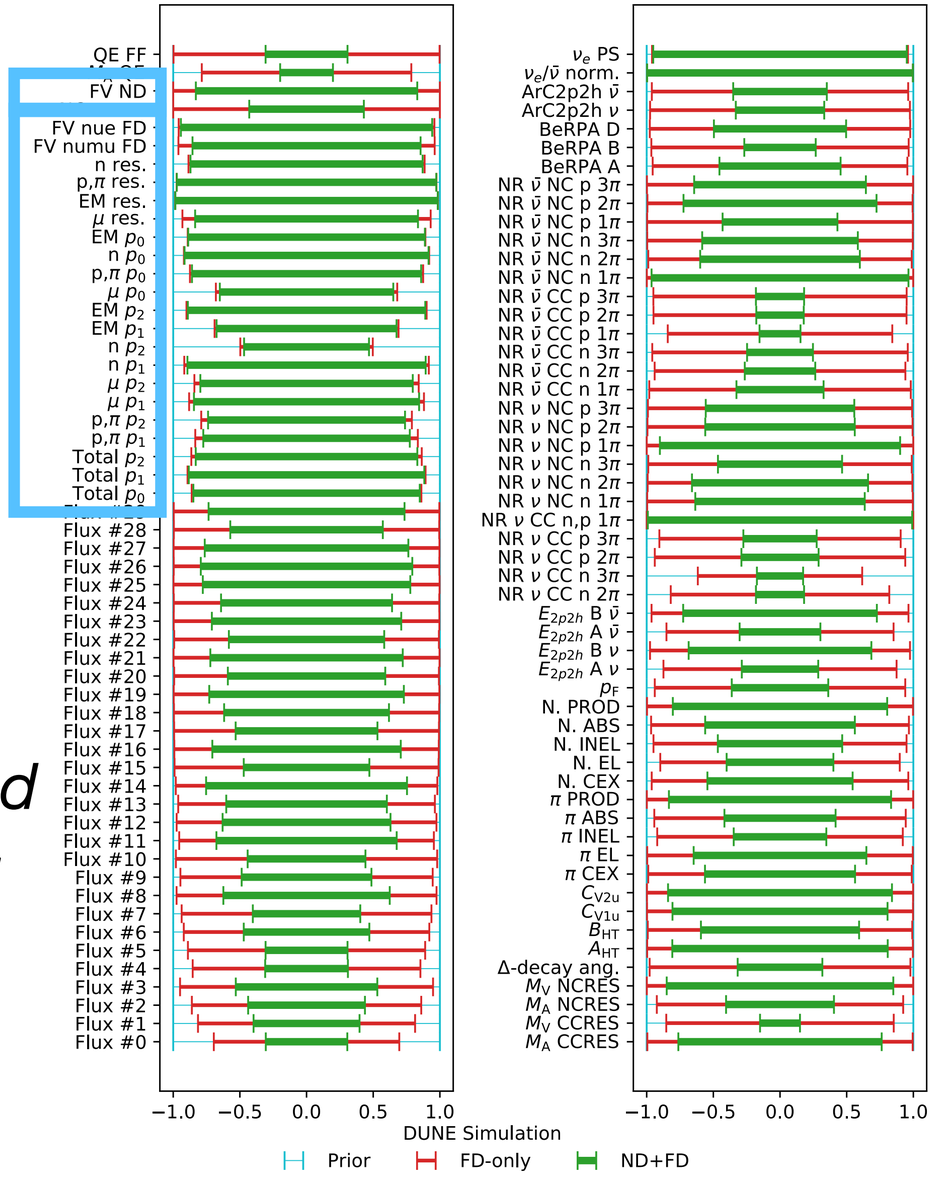
The example of DUNE

- Future high-precision measurements will require unprecedented constraining of the **systematic uncertainties**
- Detector-related systematics will be very important



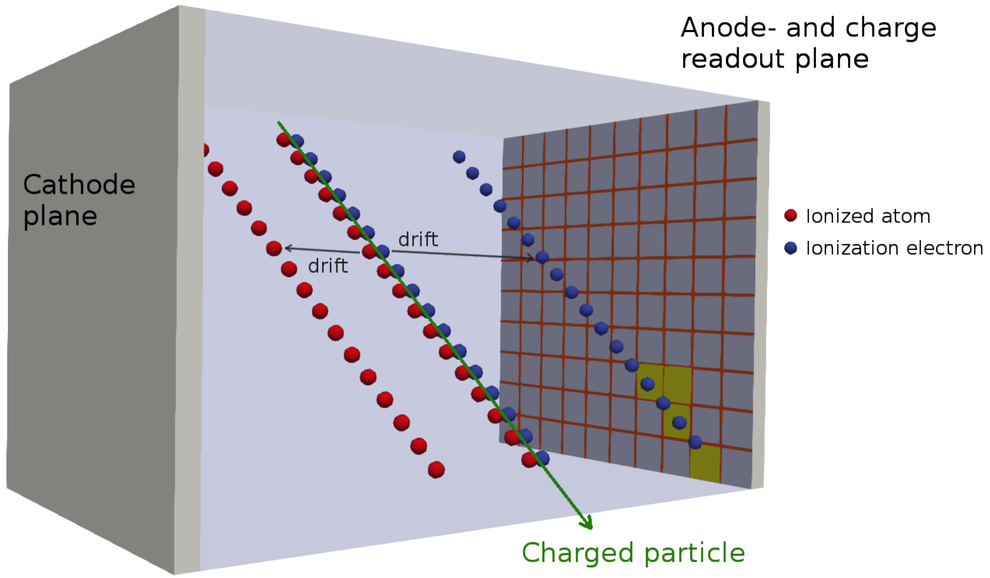
« Well-understood detector modeling and calibration are vital »

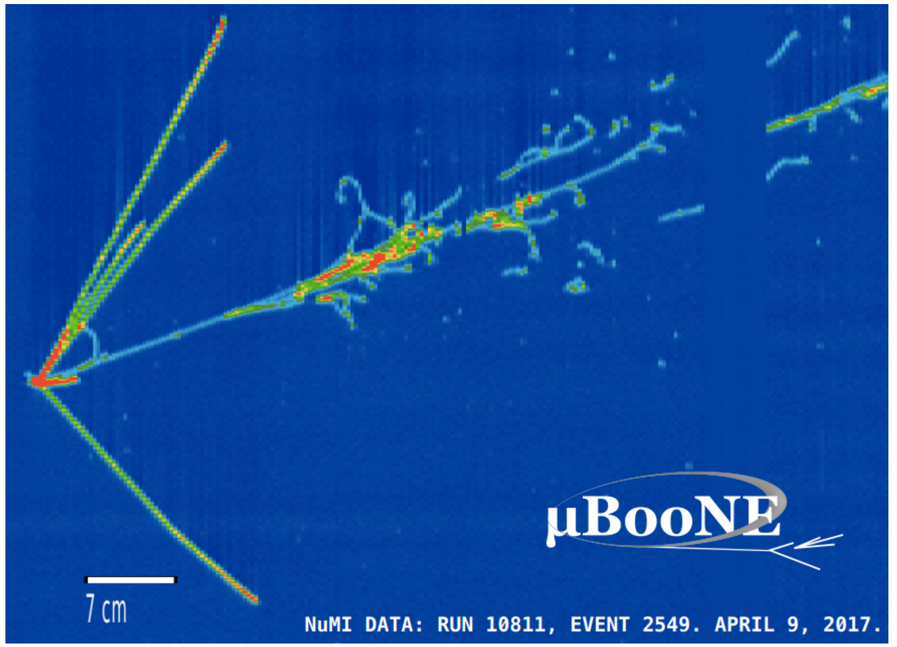
Eur.Phys.J.C 80 (2020) 10, 978



Liquid Argon Time Projection Chambers (LArTPCs)

Principle





Signal production steps:

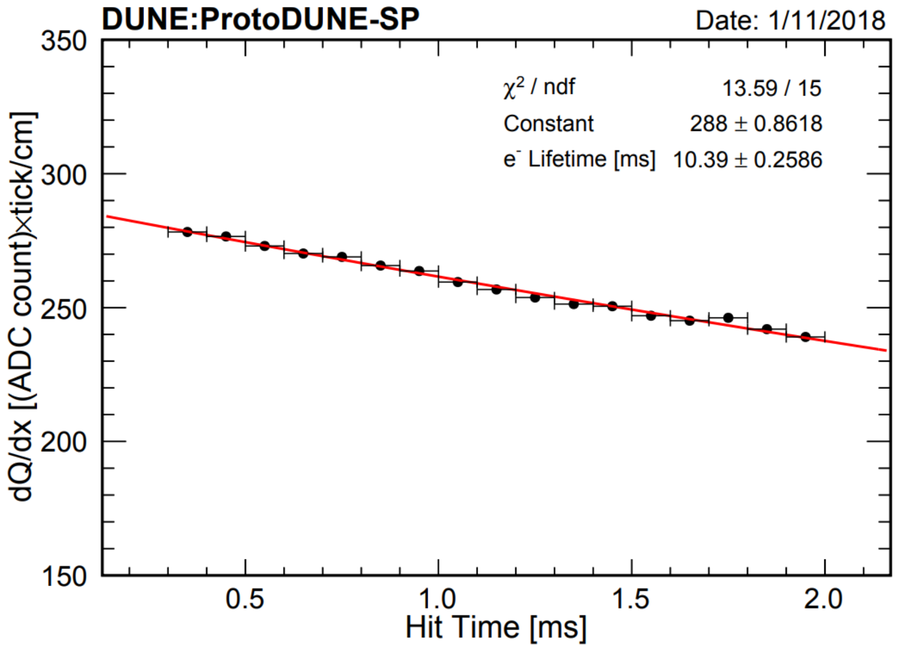
- Argon ionisation
- Ionisation electrons drifted by **E** field
- Signal induction on anode plane
- Electronics self-triggering

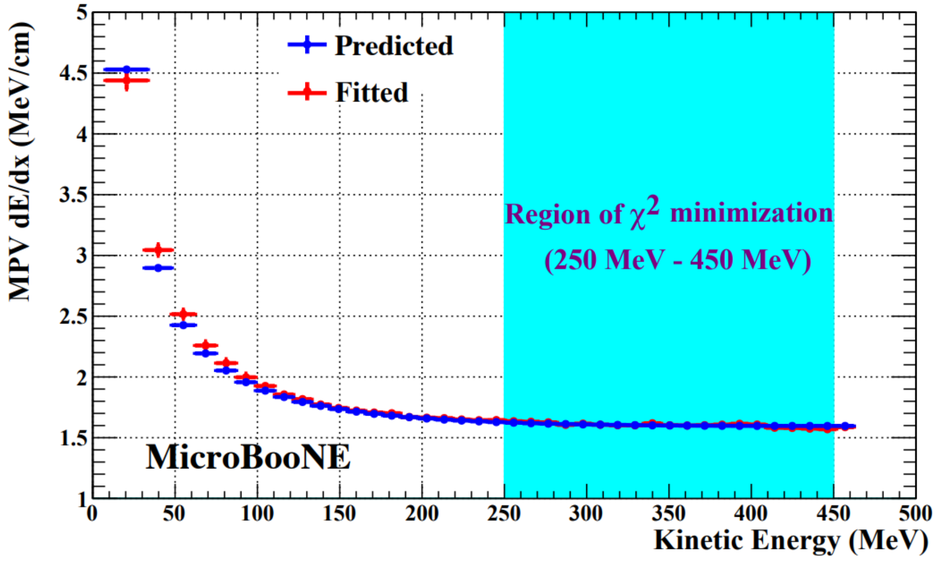
- Allows to get precise 3D picture of the interaction
- Relies on multiple physical processes → **importance of calibration**

Handling detector calibration

The usual way

- Find yourself some **calibration sample** (e.g., through-going muons)
- Look at some **summary quantity** (e.g., dQ/dx as a function of time)
- Fit parameter values
- Input new values into MC, **check Data/MC mismatch**

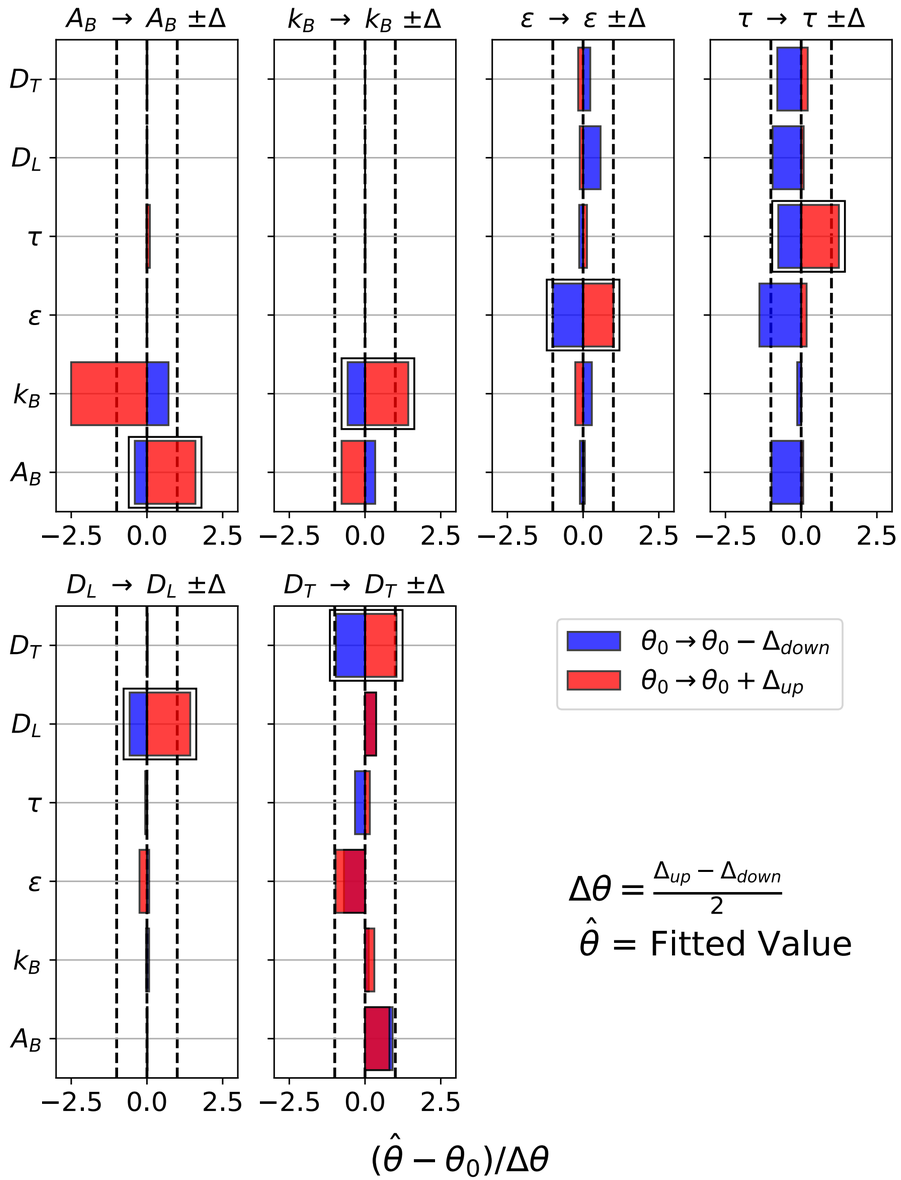




Handling detector calibration

The issues

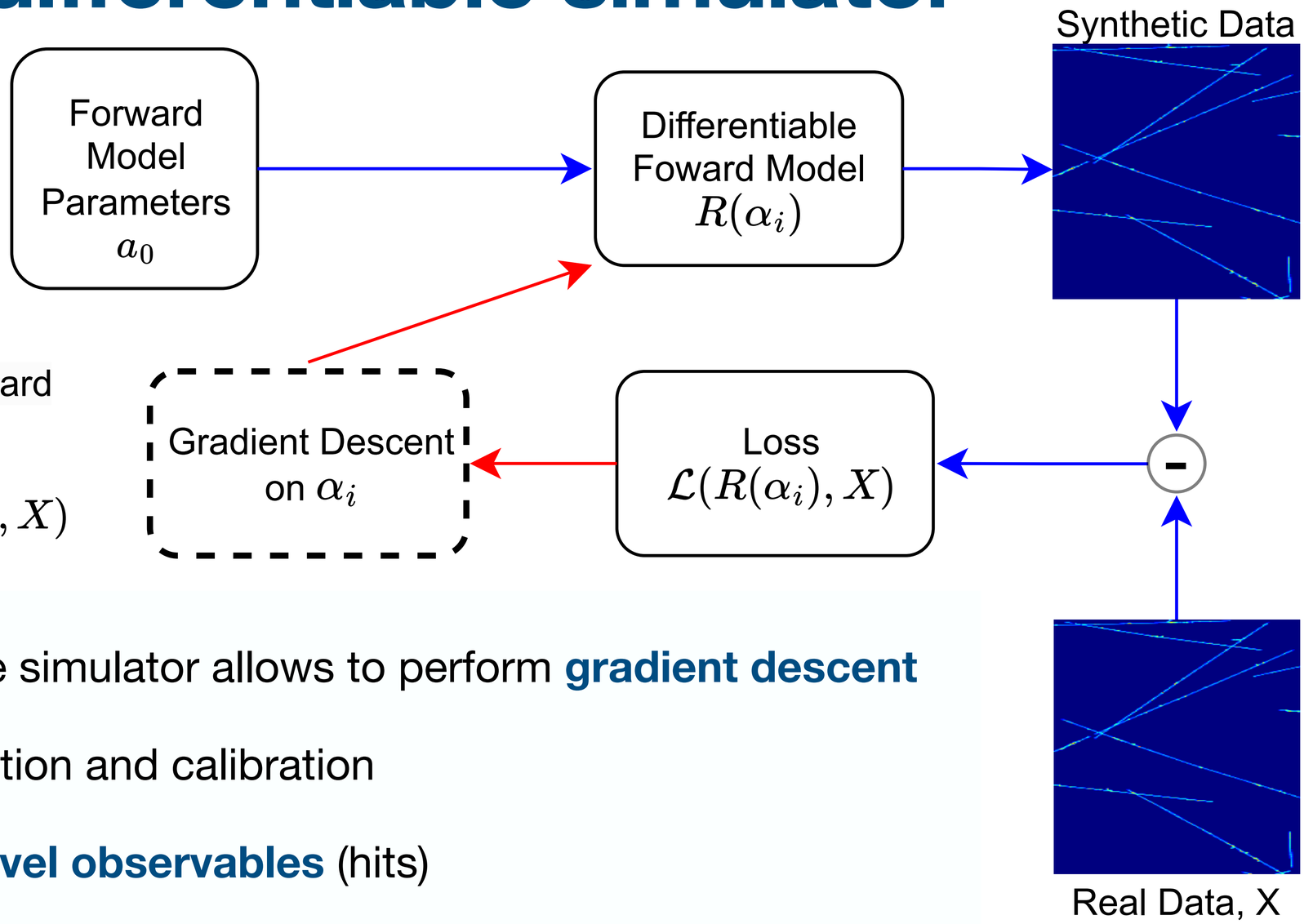
- Tedious iterative calibration procedure
- Hard to handle well **interplays** between the different parameters
- Relies on summary quantities rather than low-level information → **lossy compression of information** + depends on some reconstruction



Introducing a differentiable simulator

Working principle

$$\alpha^* = \arg\min_{\alpha} \mathcal{L}(R(\alpha), X)$$



- The differentiable nature of the simulator allows to perform **gradient descent**
- **Unified framework** for simulation and calibration
- Working directly on the **low-level observables** (hits)

Calibration in practice

A 2D example

1. Choose the initial parameter values θ_0

2. Run the forward simulation $f(\chi, \theta_0)$

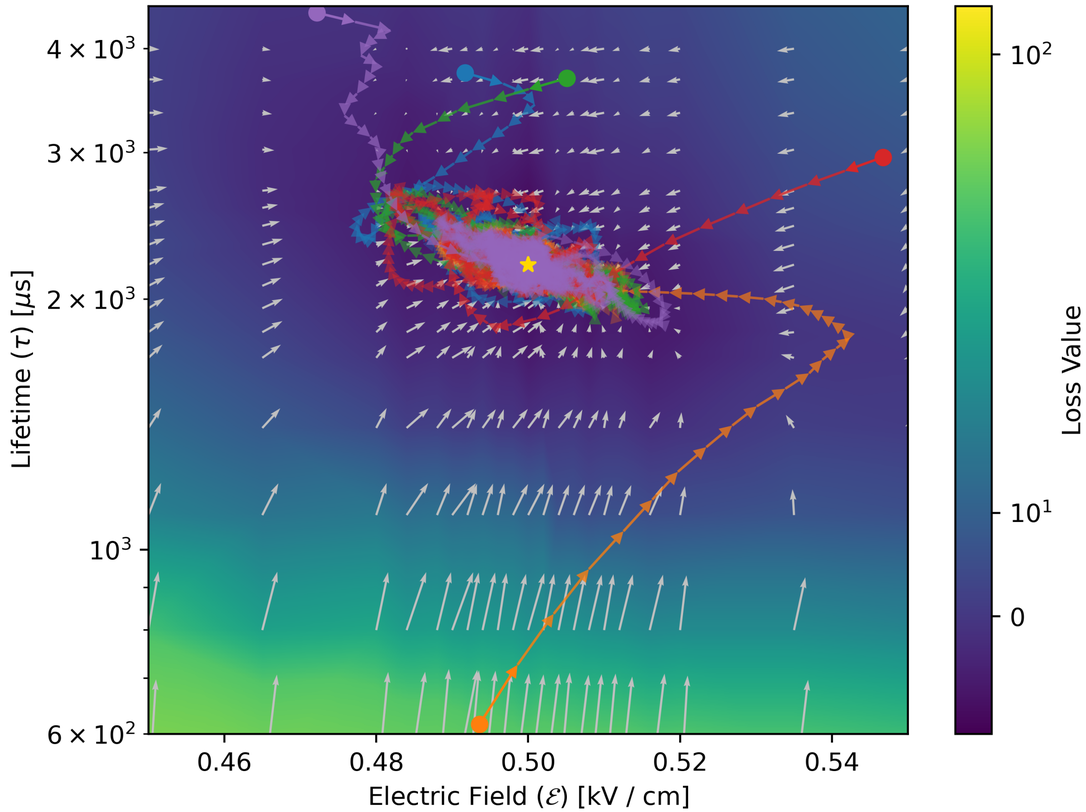
3. Compare the simulation output and the target data with a loss function $\mathscr{L}(f(\chi, \theta_0), F_{\text{target}})$

4. Calculate gradients for the parameters $\nabla_\theta \mathscr{L}(f(\chi, \theta_0), F_{\text{target}})$

5. Update parameter values $\theta_0 \rightarrow \theta_i$ to minimize the loss

Iterate step 2. to 5.

- Input particle segments (position and energy deposition): χ
- Model parameters: θ
- Differentiable simulation: $f(\chi, \theta)$
- Target data: F_{target}



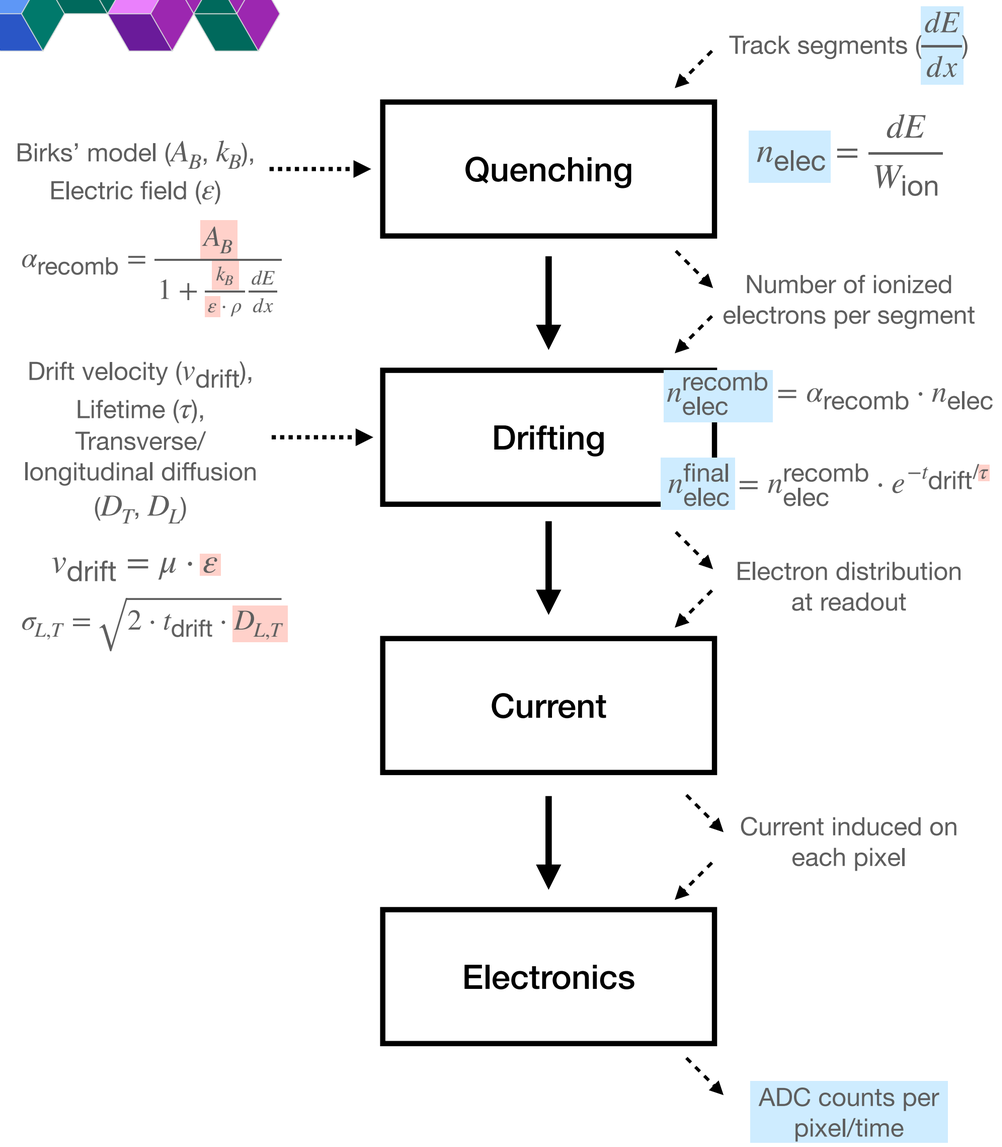
LArNDSim-jax

https://github.com/pgranger23/larnd-sim-jax

How does it work in principle

Take existing DUNE near-detector simulation (JINST 18 P04034) and make it differentiable:

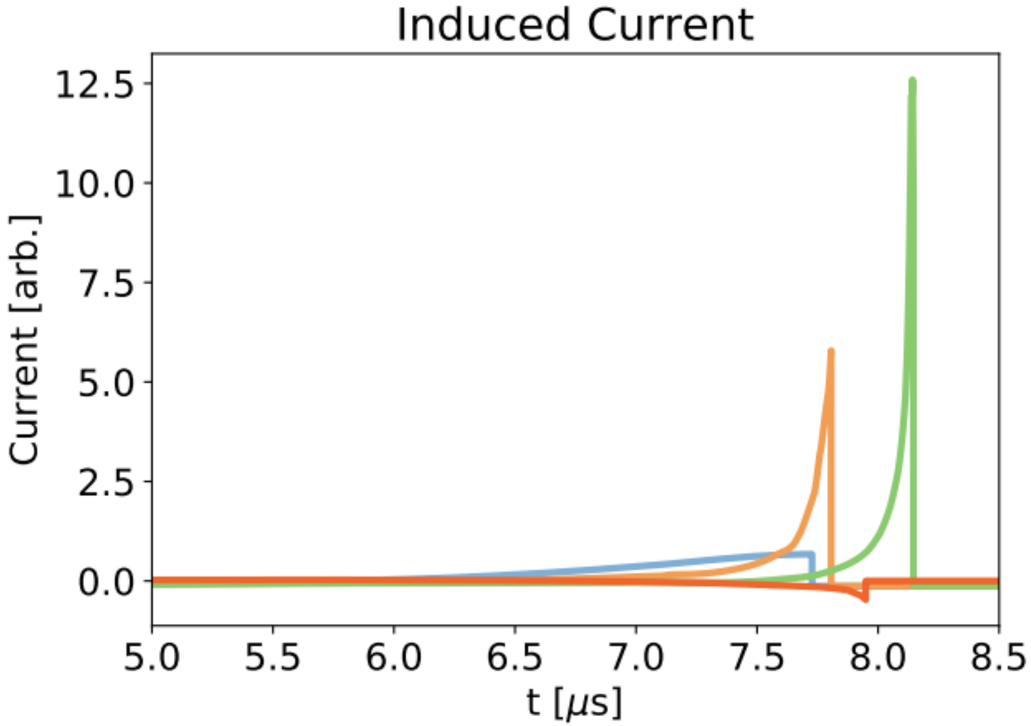
- Retain **physics quality** of a tool used collaboration-wide while adding ability to propagate derivatives
- Demonstrate the use of this differentiable simulation for **gradient-based calibration and uncertainty propagation**
- Rewritten in JAX to benefit from autodial, JIT, XLA, …

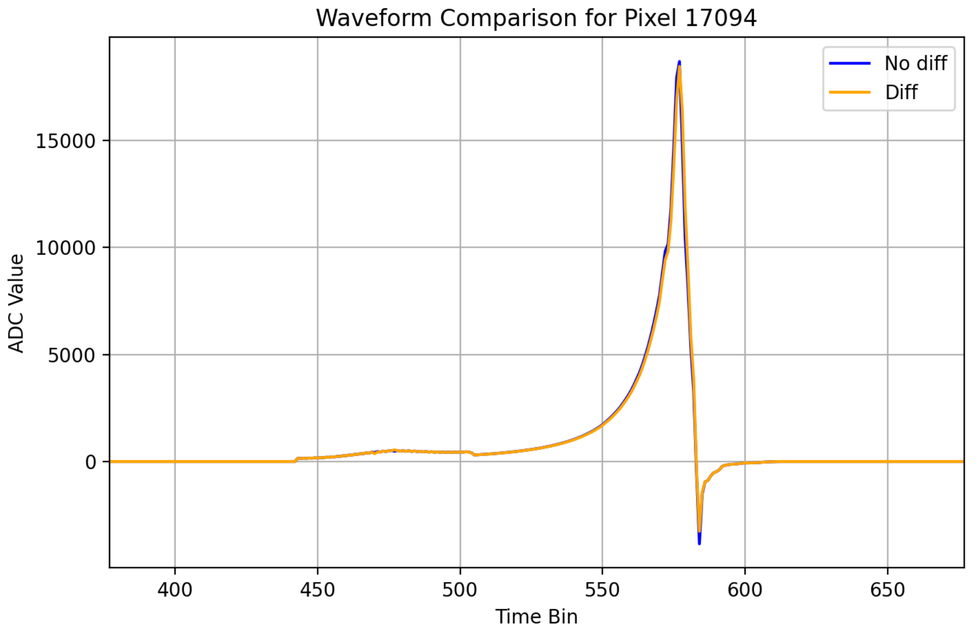


LArNDSim-jax

Changes since last year

- Computational performance improvement
- Implementation of **current LUT** with interpolation
- Now considering signals induced on neighbouring pixels (each electron induces signal on 81 pixels)
- Running more tests to ensure **no major deviation from the reference simulator**

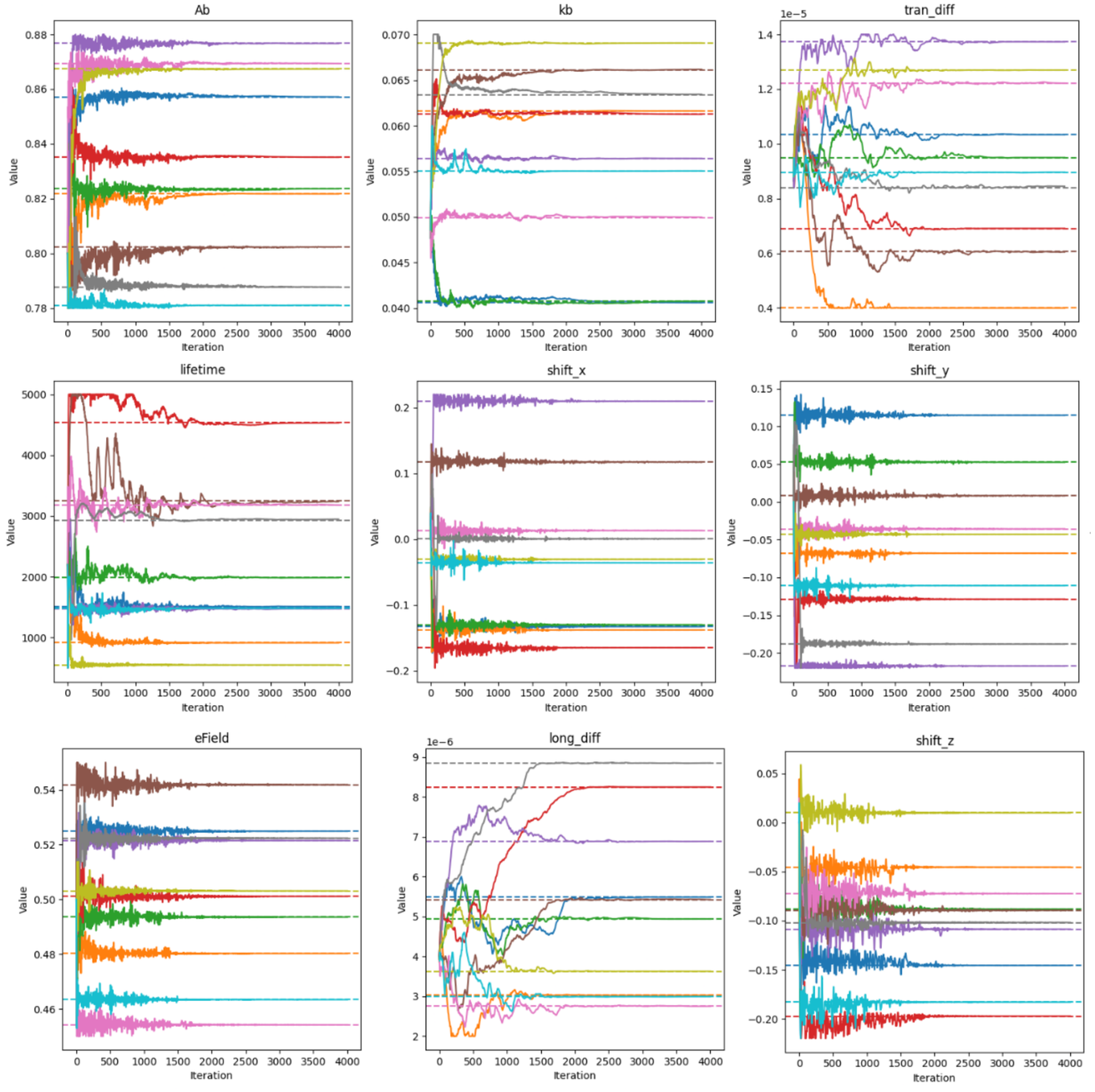




Calibration

Improvements

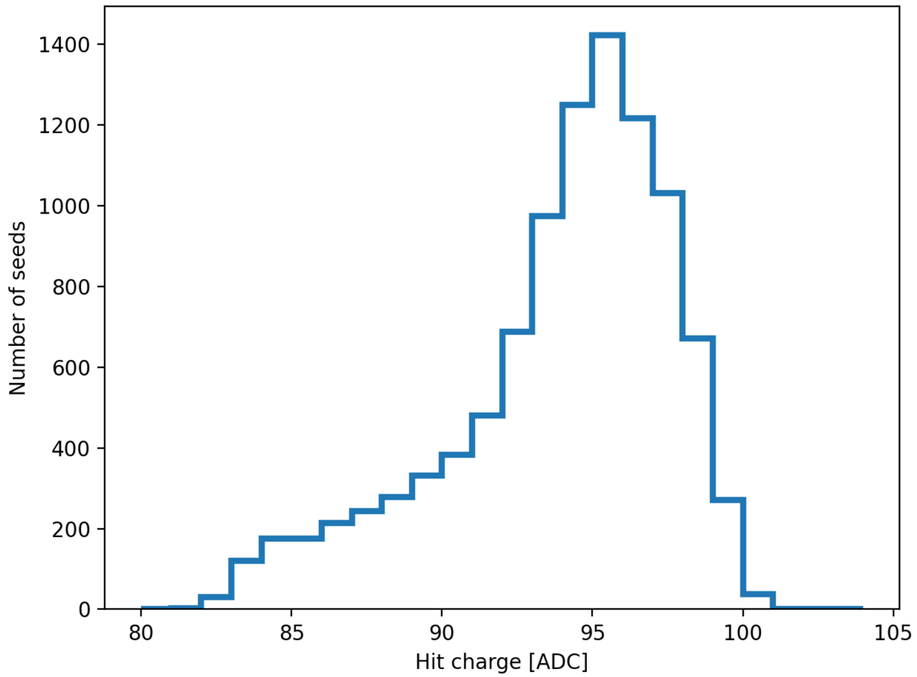
- Adding misalignment parameters (shift_{xyz})
- New loss function that can handle translations
- **See more in Yifan's talk with application on real data**



Sources of uncertainties

And where to find them

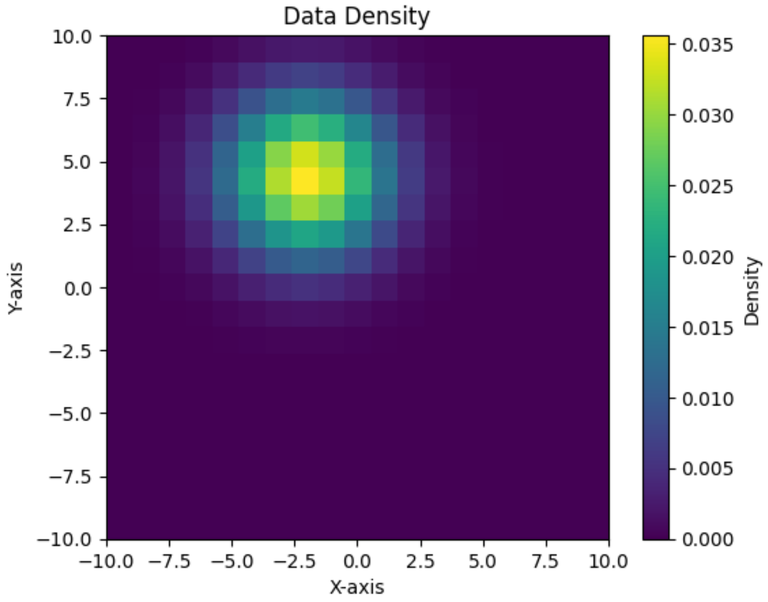
- **Statistical uncertainty** from stochastic processes:
  - Electrons diffusion
  - Electronics noise
- **Systematic uncertainties** from our model:
  - Uncertainty of the input data model (track dE/dx, position, …)
  - Uncertainty of the detector parameters (the ones we want to calibrate)

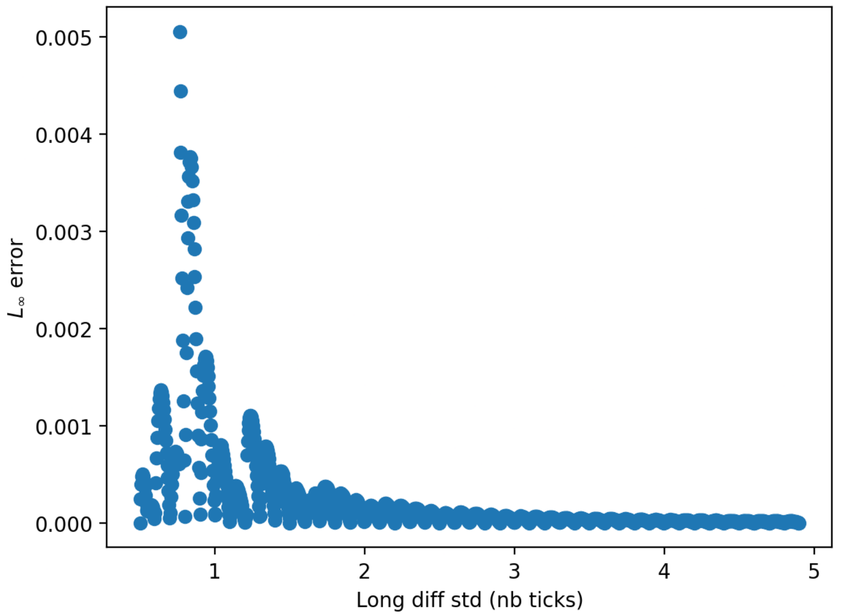


Dealing with stochasticity

Electrons diffusion

- Large number of electrons → only need to simulate the integrated collective behaviour
- Transverse diffusion: compute the integrated amount of charge on each LUT bin using error function.
- Longitudinal diffusion:
  - Using templated response functions with different values of the transverse diffusion value
  - Making a **quadratic interpolation** for any value
  - **Max of the difference on the whole waveform below 0.5%**

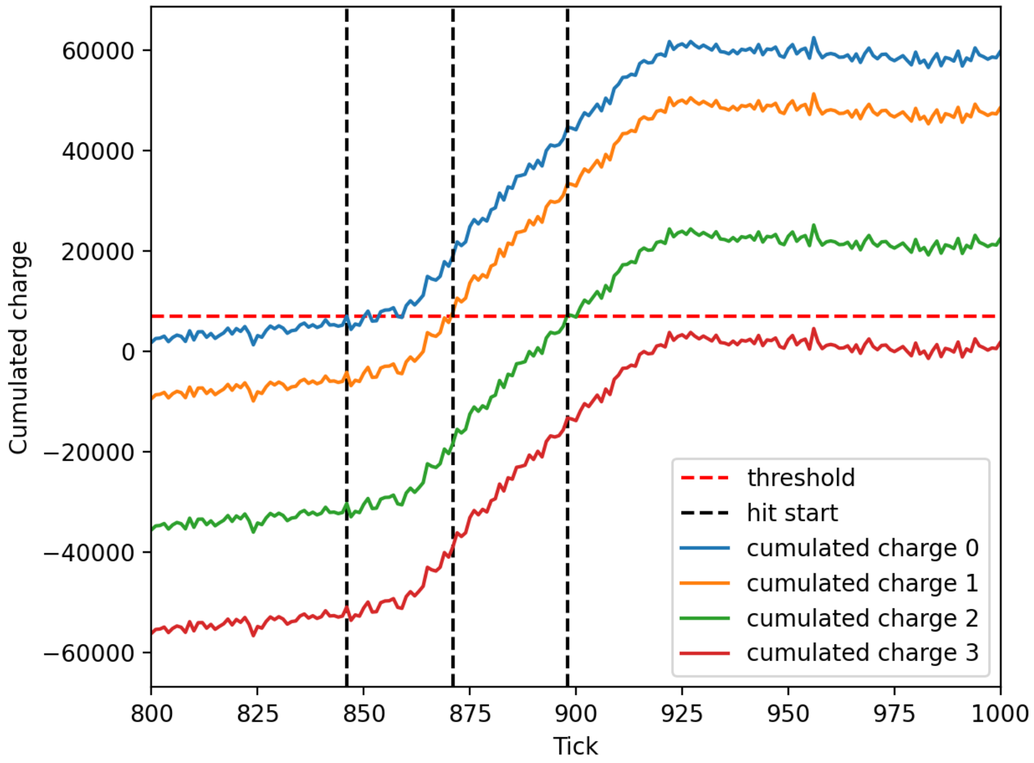




Towards a probabilistic electronics simulation

Self triggering in DUNE ND

- DUNE ND technology is relying on **self triggering** logic to automatically build hits
- Simulation of **correlated noise** across the ticks (reset charge noise) and **uncorrelated noise**
- When the signal (+ relevant noises) crosses the threshold, a new hit is triggered. The hit duration is fixed and the number of ADC corresponds to the digitised cumulated charge at (hit_time + hit_duration).
- Then **re-triggerings can happen** to form new hits and so on.



Towards a probabilistic electronics simulation

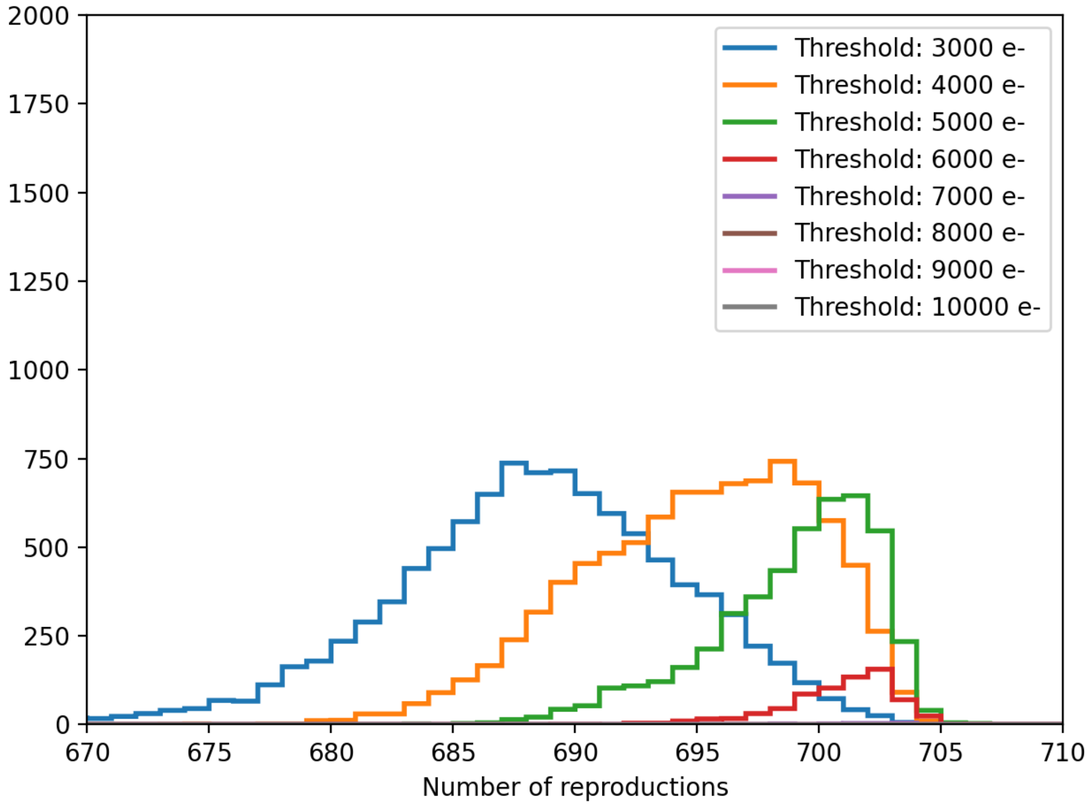
Impact of electronics stochasticity

This self-triggering logic implies several things:

- 2 different noise seeds can lead to very different (>20 ticks) hit outputs
- Output depends highly on the pixel discrimination threshold (non-uniform across the readout plane)

→ **not ideal to get meaningful and accurate gradients/estimates for calibration**

Running the digitization simulation 10000 times with different seeds for identical waveform

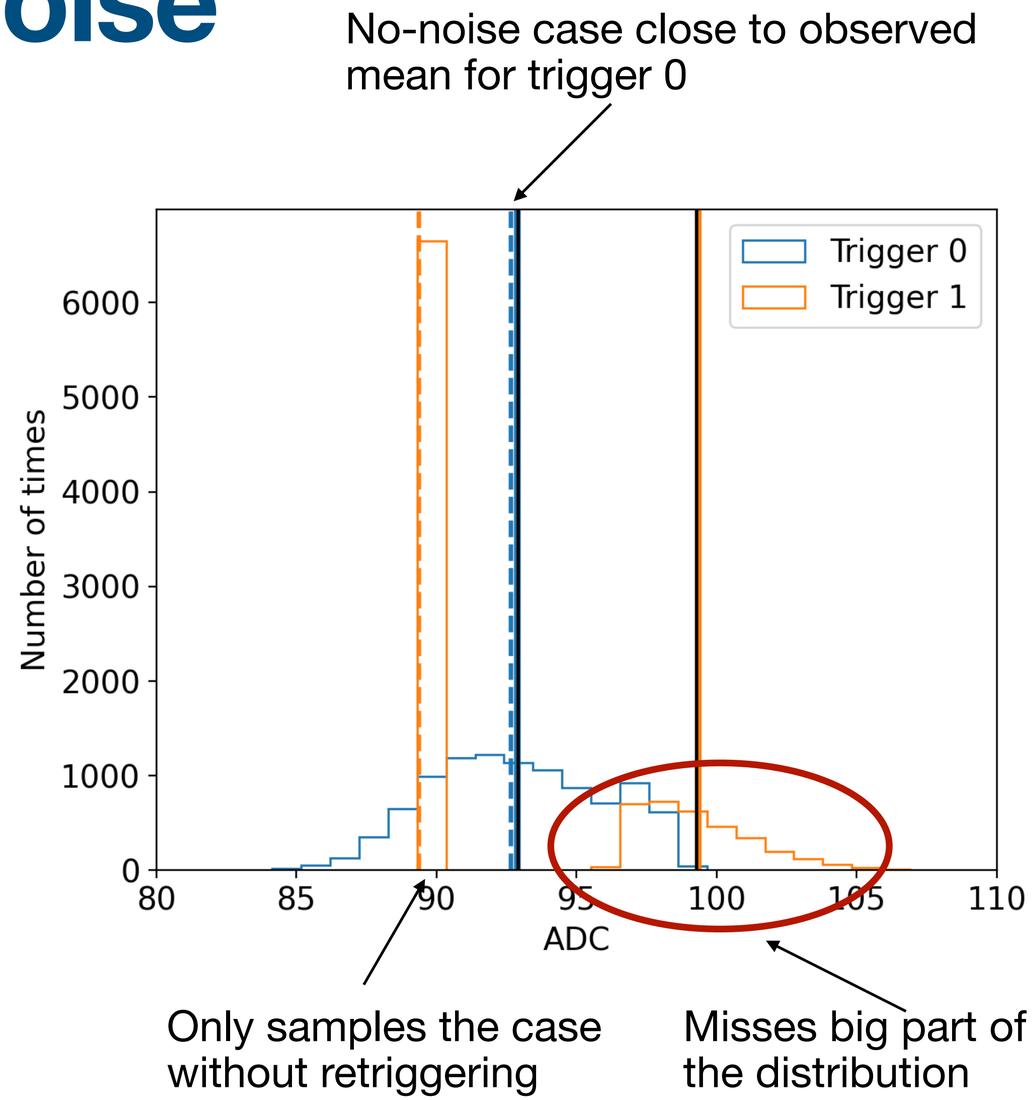


Dealing with electronics noise

Towards a probabilistic simulator

Difficulties associated to the stochastic processes.
Different options/problems:

- Simulate reference without noise, should be representative, no? Well no!
- Simulate a single noise seed: not very accurate (2σ away from another random seed)
- Simulate many noise seeds and get some average: slow!
- **Directly outputting probability distribution**: enables likelihood fit, smoother evolutions/ gradients, fast



dashed line: avg w.o. noise

plain line: actual avg

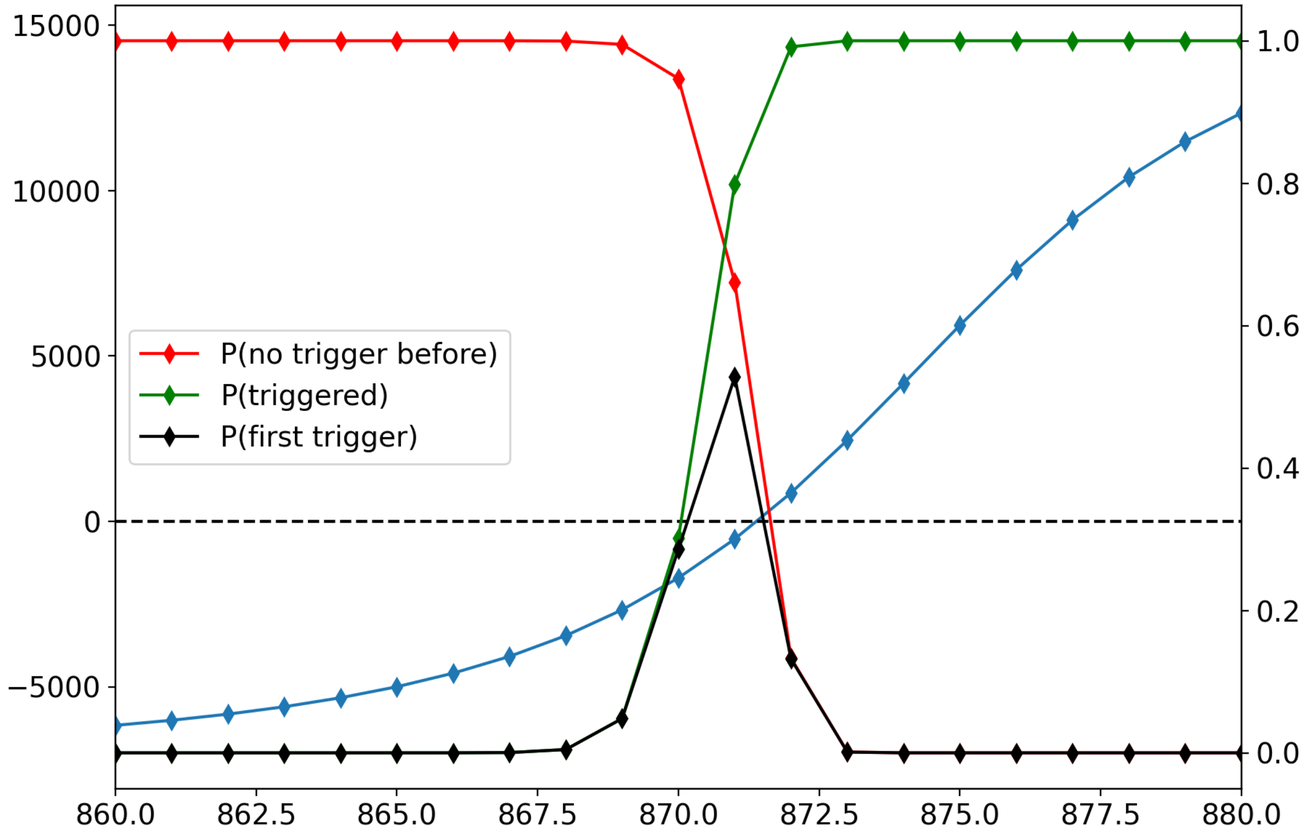
Towards a probabilistic simulator

Doing some basic maths

The noise being gaussian, it is possible to **derive analytically** the hit probability distribution for a given waveform (under some approximations and assumptions)

Things boil down to (modulo many subtleties):

$$\mathscr{P}(\text{trigger at } t) = \frac{1 + \operatorname{erf}\left(S(t) - T\right)}{2\sqrt{2}\sigma}$$

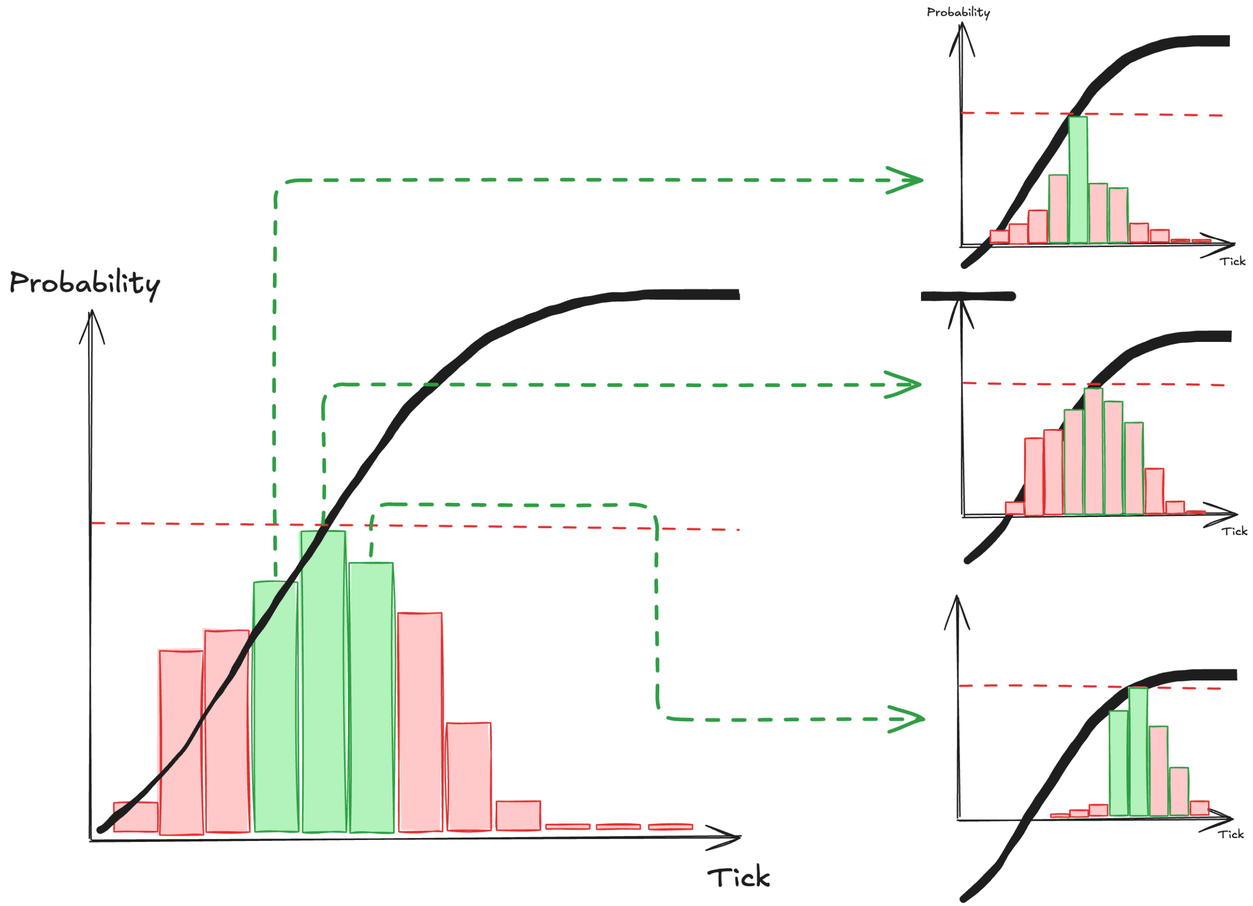


Towards a probabilistic simulator

Handling re-triggering

Things get a bit more complicated with re-triggering as the state of the new waveform depends on the triggering probabilities of the previous hit → **complexity grows exponentially**

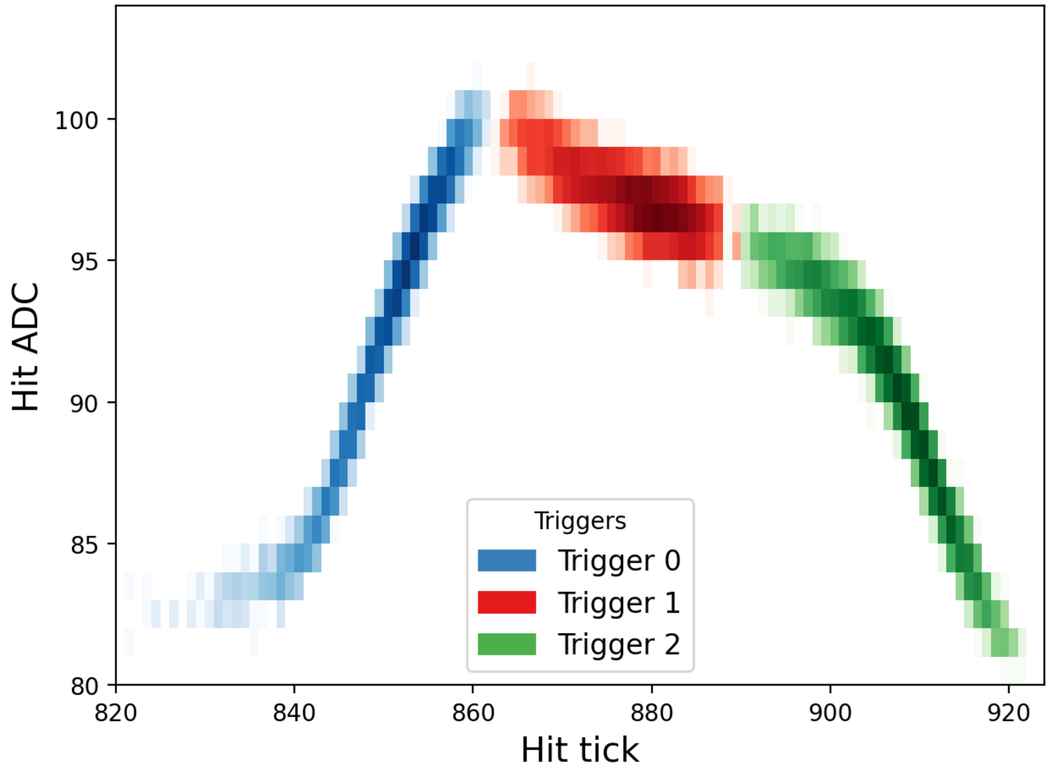
Implementing some **beam-search algorithm**: iteratively work on many possible paths in parallel, keeping the most probable ones



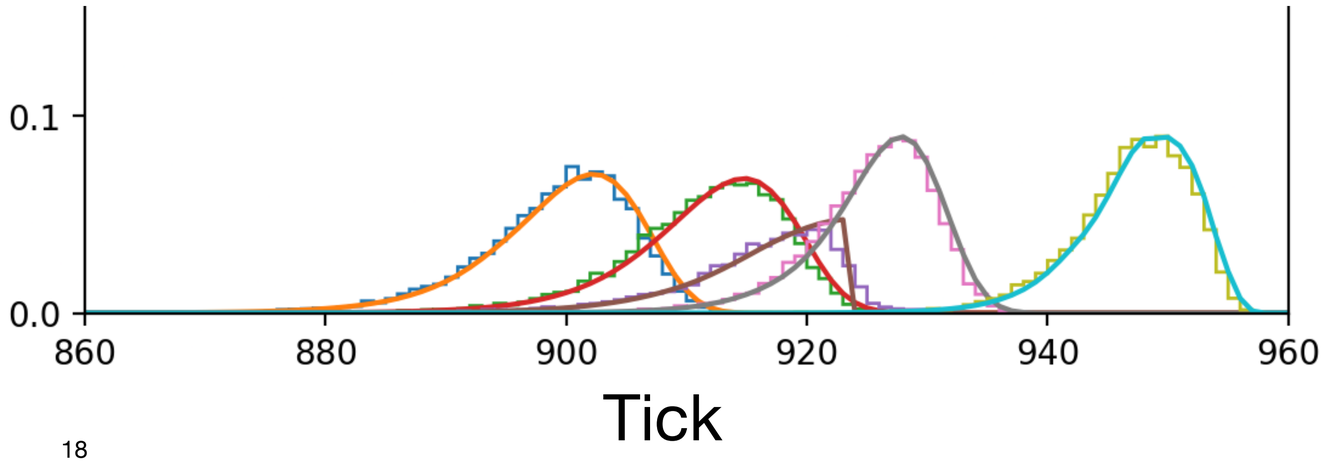
Towards a probabilistic simulator

Results

- Outputs the **joint 2D probability distribution (hit tick + hit ADC)**
- **Way more powerful** than returning one random draw (statistical uncertainty estimation, likelihood fit, …) for **<10x** the computational cost of a single seed



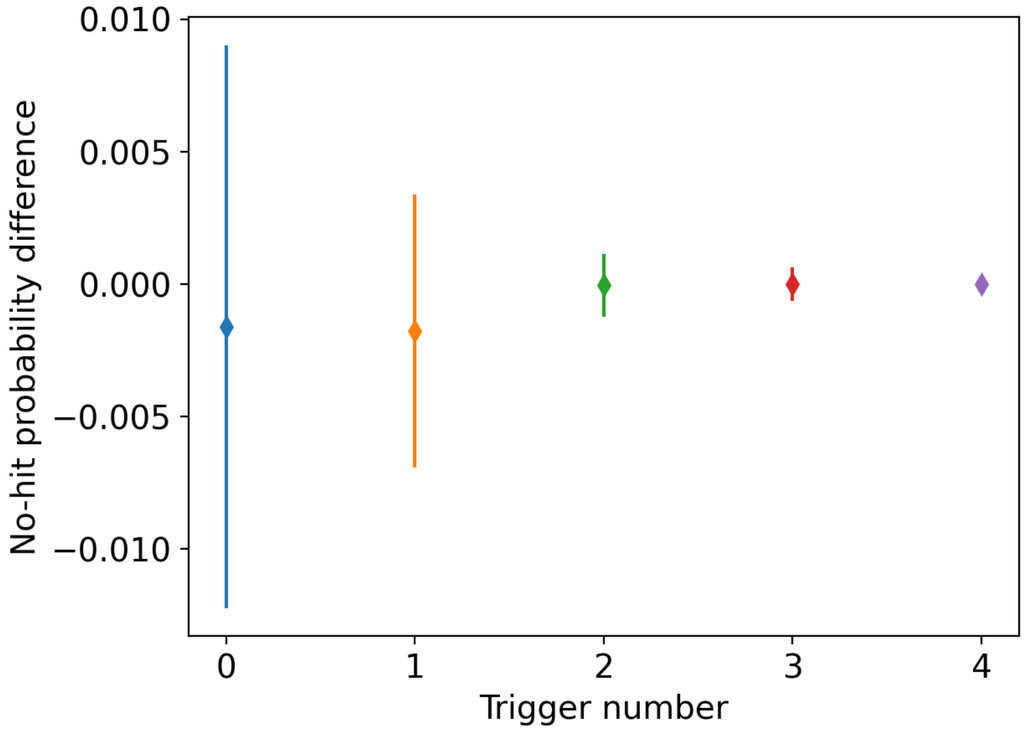
Comparison of the 1D observed distributions (hist) vs probabilistic estimate → very good agreement

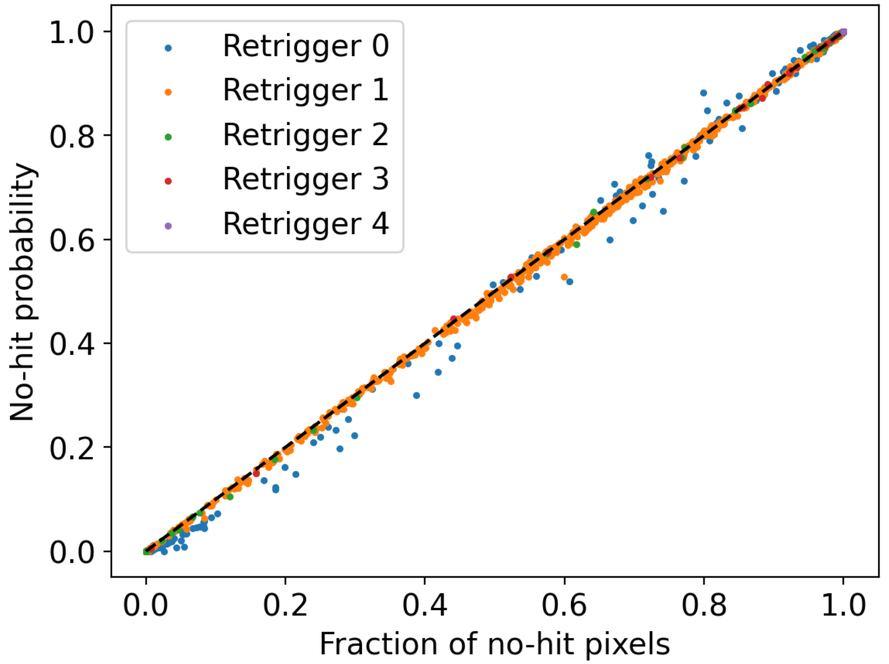


Towards a probabilistic simulator

Results

- Checking the accuracy of the **hit probability** estimate (sum of output probability distribution)
- **< 1% error** on hit probability

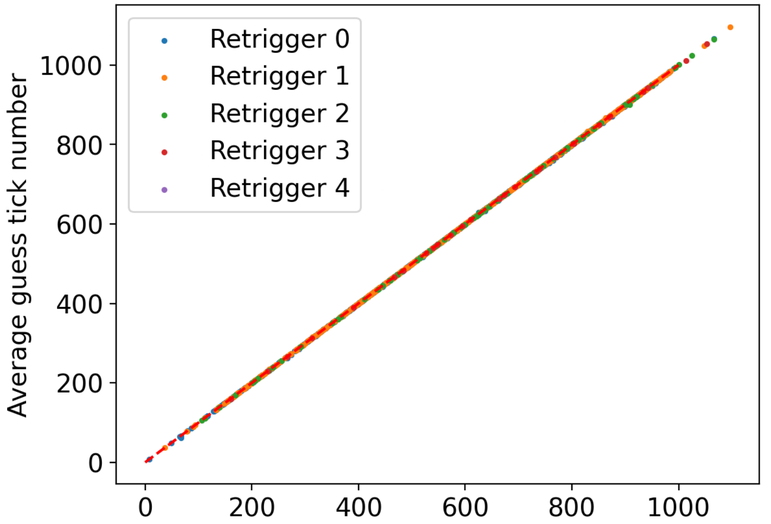


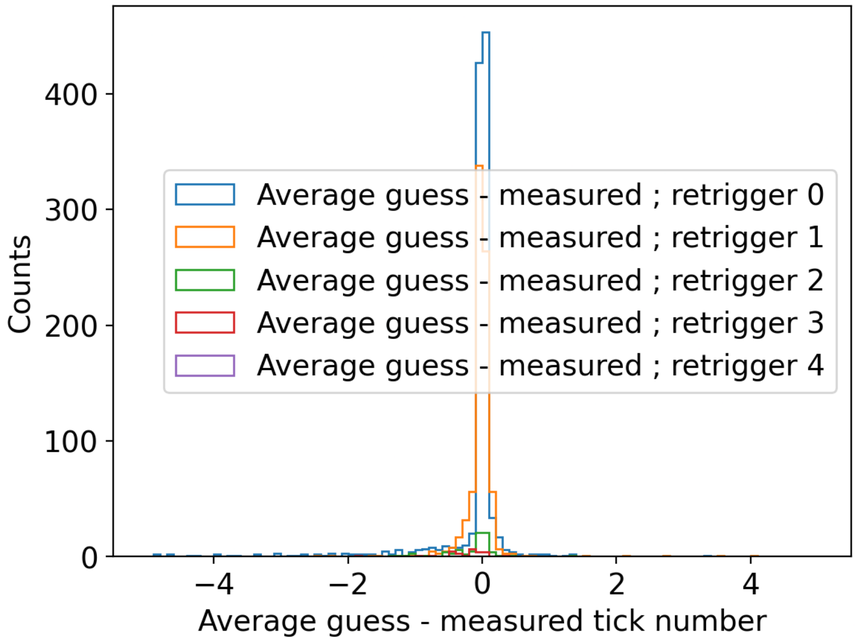


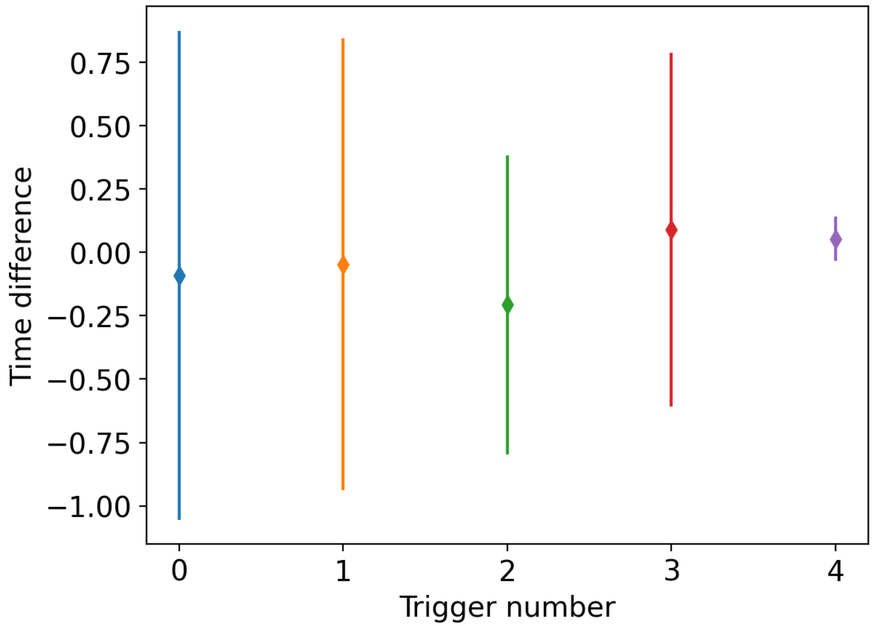
Towards a probabilistic simulator

Results

- Looking at **average expected hit time** (weighted by prob density)
- Error **below 1 tick**



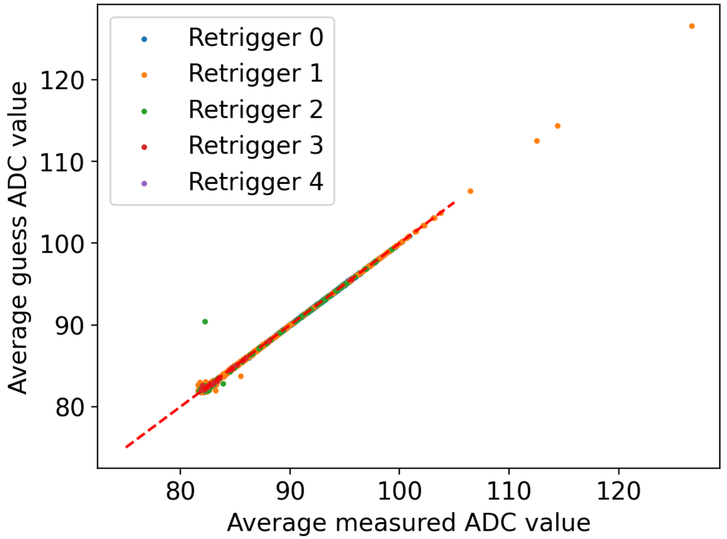


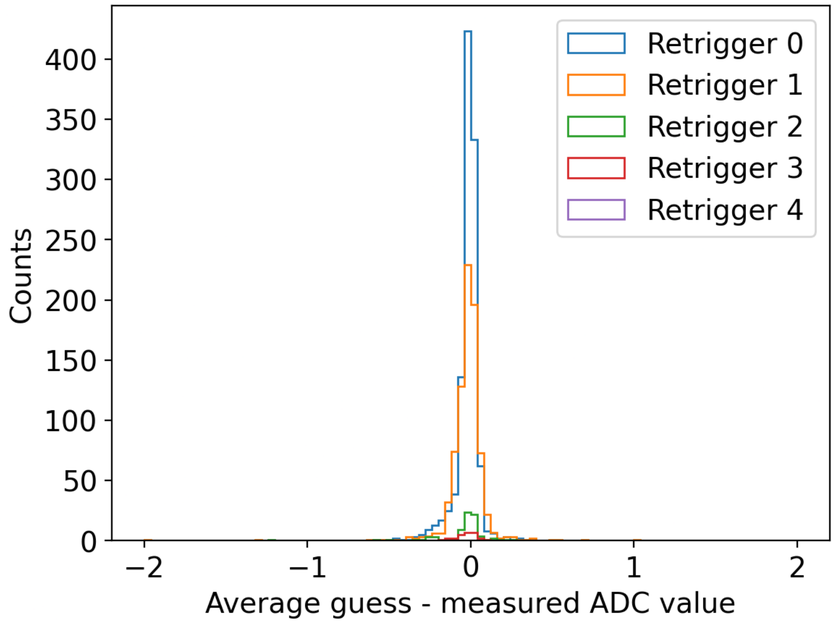


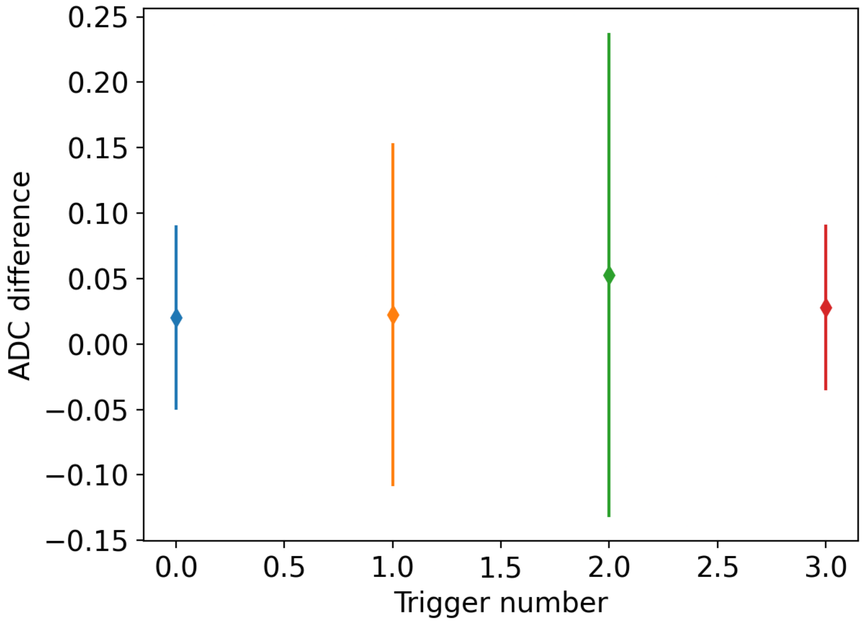
Towards a probabilistic simulator

Results

- Looking at **average expected hit ADC** (weighted by prob density)
- Error **below 0.25 ADC**



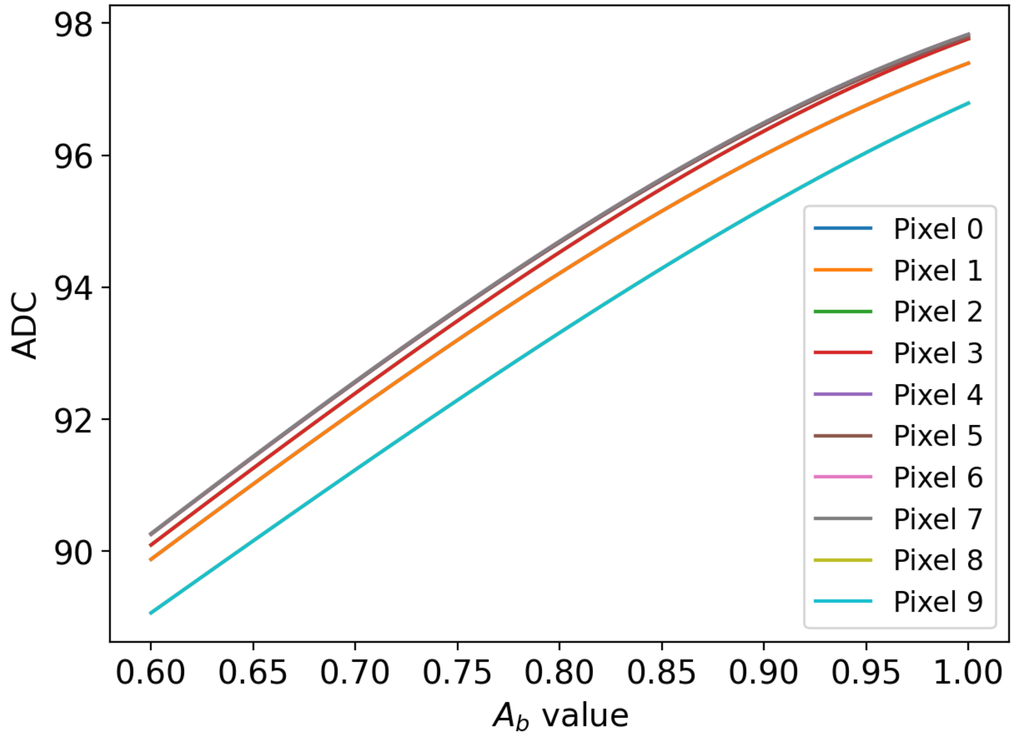


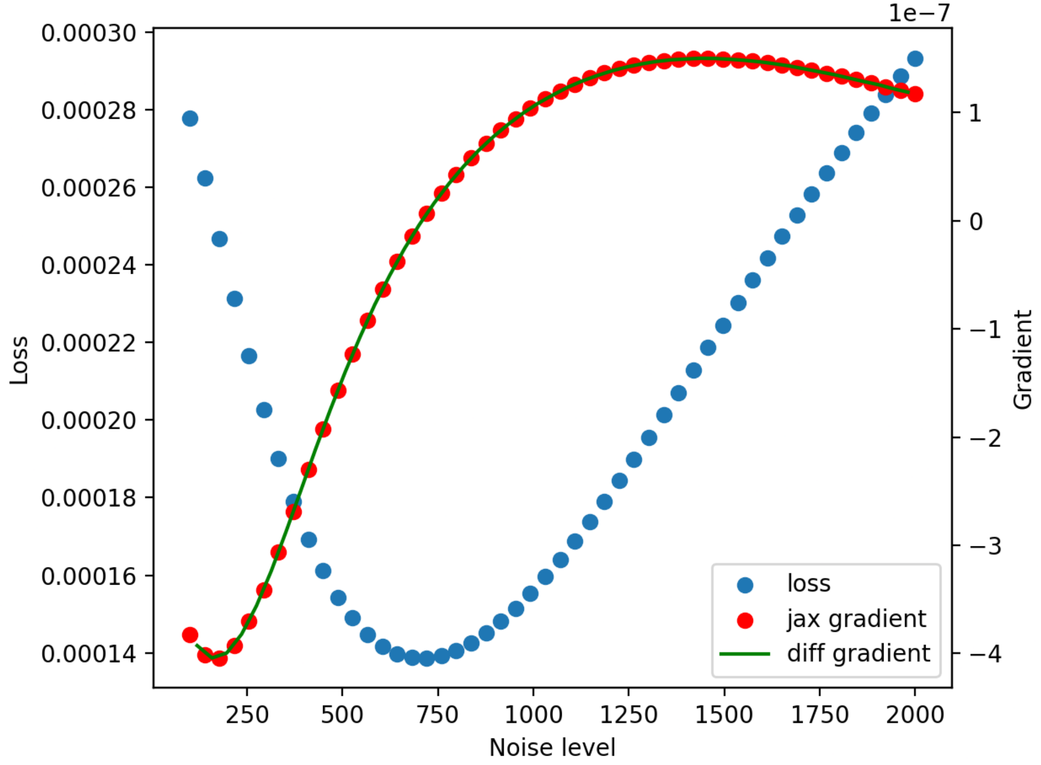


Towards a probabilistic simulator

Checking the gradients

- **Smooth evolution** of the output average values with detector params
- **Accurate gradients**, the noise level can even now be calibrated too





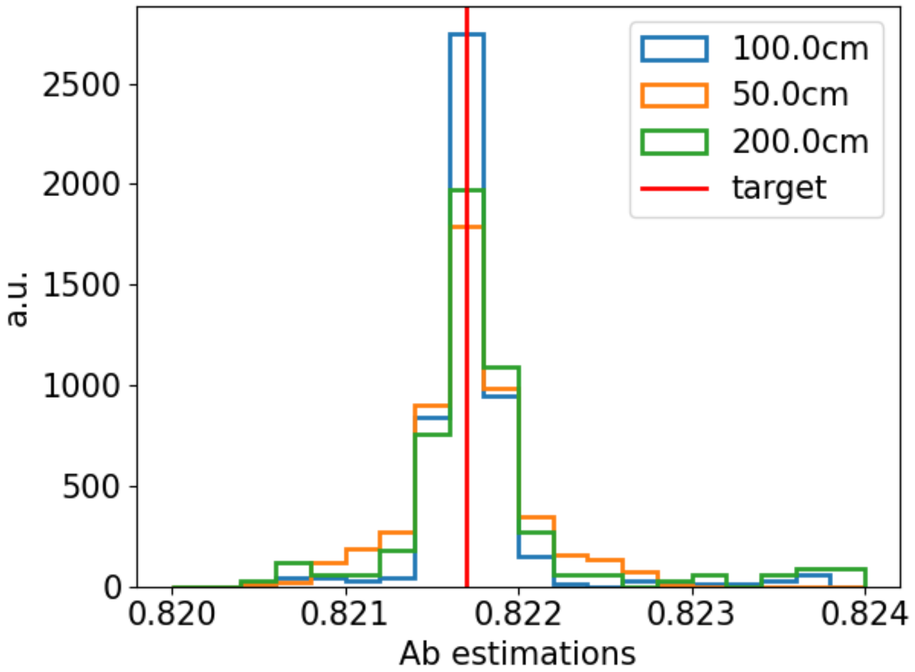
Calibration

Estimating the uncertainty

Option 1: Doing **ensembling** by conducting several fits with different data shuffling seeds and initial parameters

Caveats:

- Computationally intensive
- Overkill as you are only interested by the behaviour of your function near the minimum
- **Good for cross-checks**

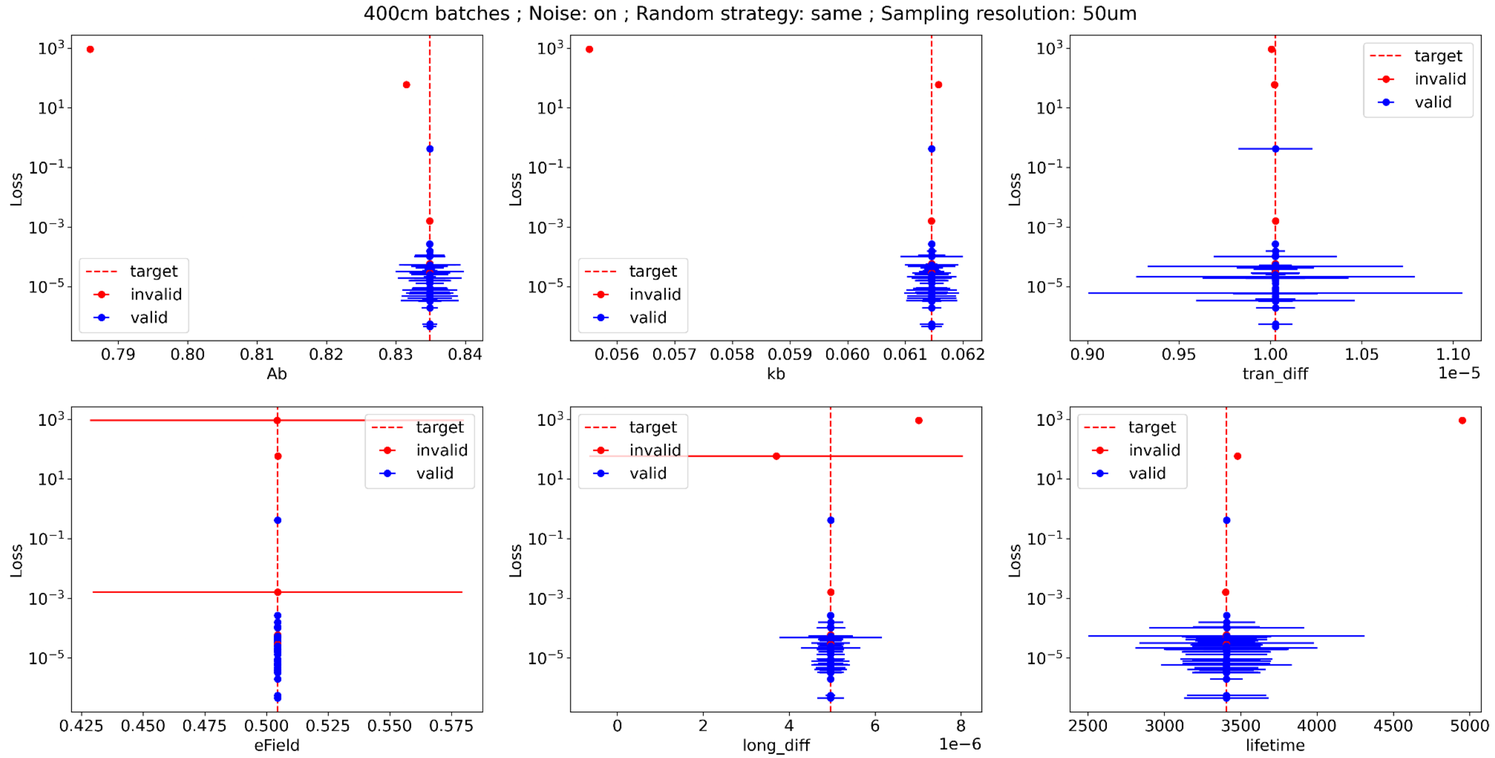


Calibration

Estimating the uncertainty

Option 2: Analyse the **loss function near its minimum**. Using Minuit fits here (with autograd speedup) to extract the uncertainty.

A bit slow but Minuit does all the work for me…



Calibration

Estimating the uncertainty

Option 3: Use the differentiability to **automatically get the Hessian matrix** at the best fit point

Leverages the capabilities of the differentiable simulator to get **much faster results**, straightforward* with Jax

Challenges:

- Relies on the accuracy of the derivatives (more than for calibration)
- Needs a loss well-behaved around minimum (Fisher information approximation)

$$\mathbf{H}_f = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1\,\partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1\,\partial x_n} \\ \frac{\partial^2 f}{\partial x_2\,\partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2\,\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n\,\partial x_1} & \frac{\partial^2 f}{\partial x_n\,\partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix} \cdot E = \begin{bmatrix} \sigma_1^2 & \sigma_{12}^2 & \cdots \\ \sigma_{21}^2 & \sigma_2^2 & \\ \vdots & & \ddots \end{bmatrix} = 2H_f^{-1}$$

```
Dense Hessian Matrix:
[[ 2.5739556e+05 -6.9323450e+05  1.8631449e+00]
 [-6.9323450e+05  2.0078755e+06 -5.0062737e+00]
 [ 1.8631449e+00 -5.0062737e+00  1.9412024e-05]]

Covariance Matrix:
[[ 6.3438725e-05  1.9216084e-05 -1.0775292e+00]
 [ 1.9216086e-05  7.1027016e-06 -1.1967621e-02]
 [-1.0775286e+00 -1.1967417e-02  1.4440898e+05]]

Parameter Uncertainties (Standard Deviations):
Ab kb lifetime
[7.9648430e-03 2.6650894e-03 3.8001181e+02]
```

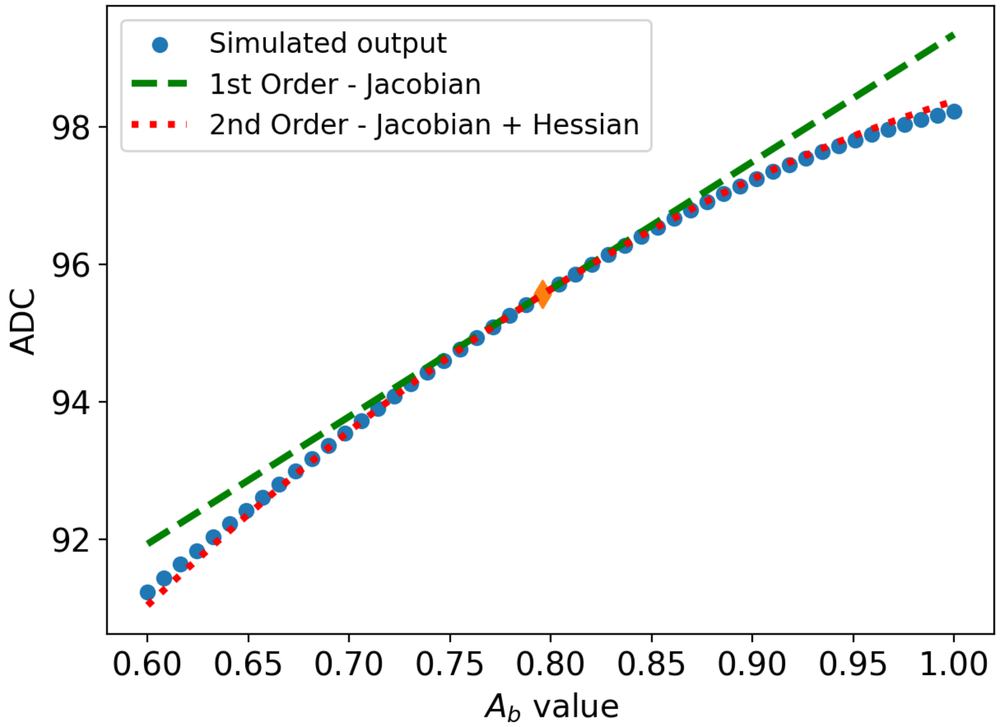
in theory, in practice memory management can become painful…

Propagating uncertainties

The theory

- Using the derivative information from the simulator, it is possible to perform **error propagation using Taylor expansion**.

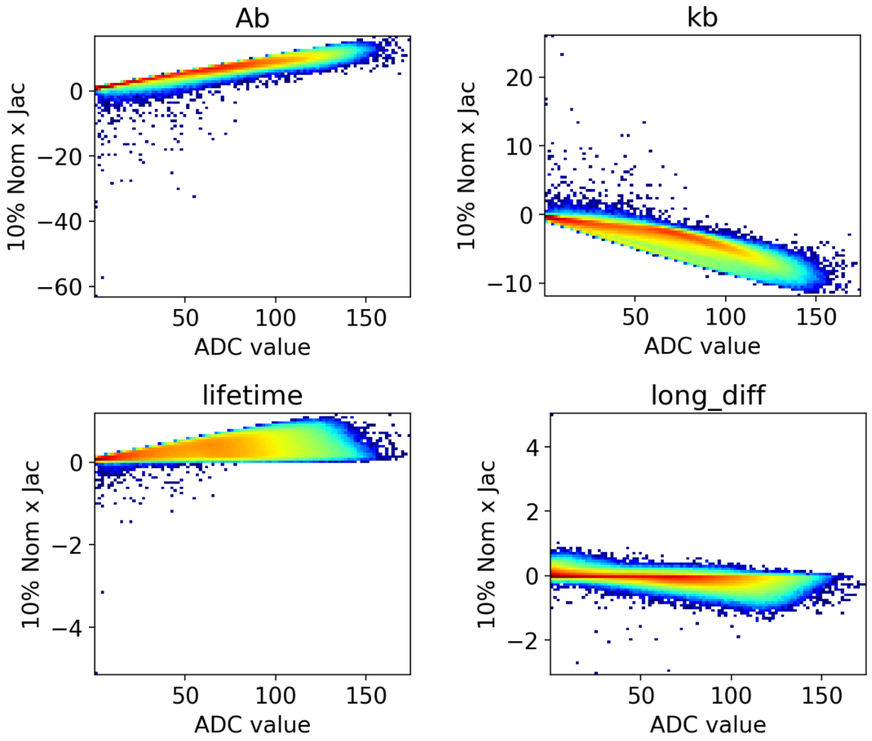
- For a given simulation output $f(\chi, \theta) = X$, we can automatically compute the **Jacobian** $J_{ij} = \frac{\partial X_i}{\partial \theta_j}$ that can be used to propagate uncertainty from input quantities to output ones: $\Sigma^{\text{syst}} = J \Sigma_\theta J^T$

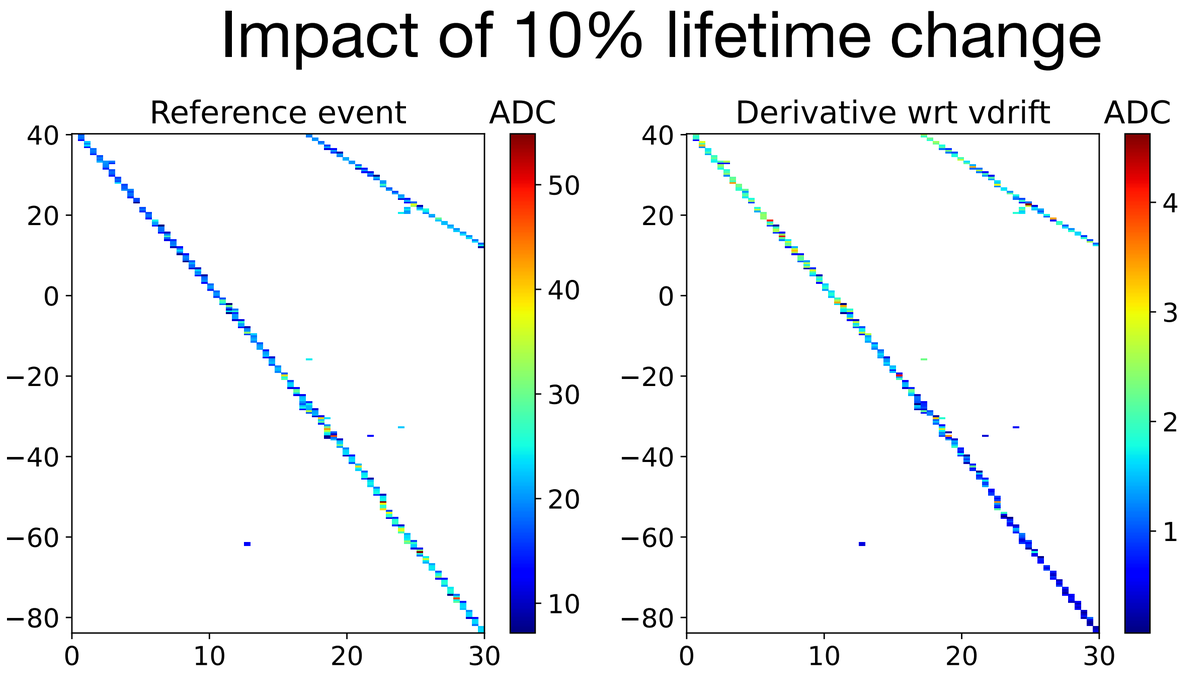


Propagating uncertainties

In practice

Here showing the impact of a **10% change** in one of the simulation parameters on the output charge of the pixels using the Jacobian

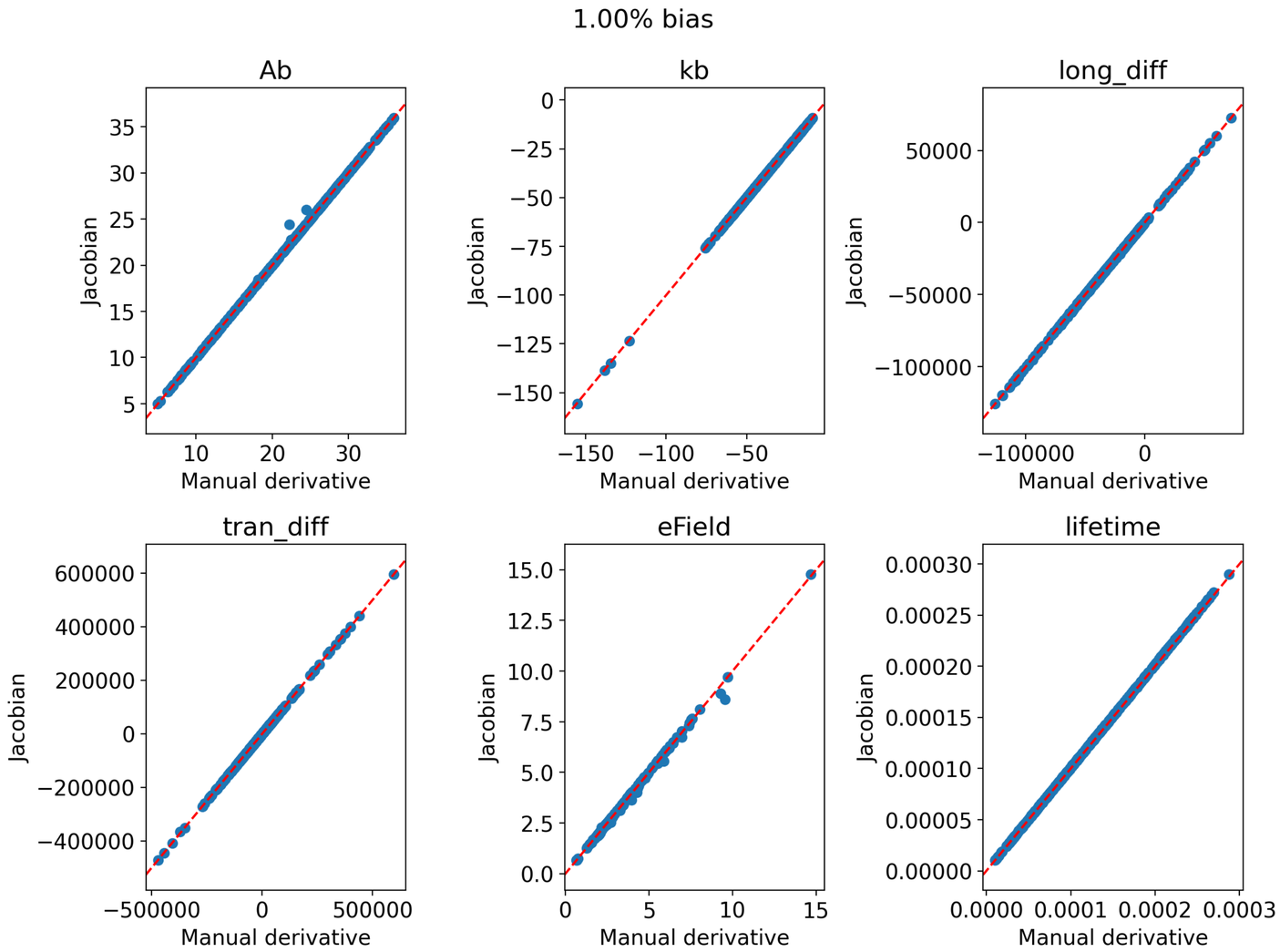




Propagating uncertainties

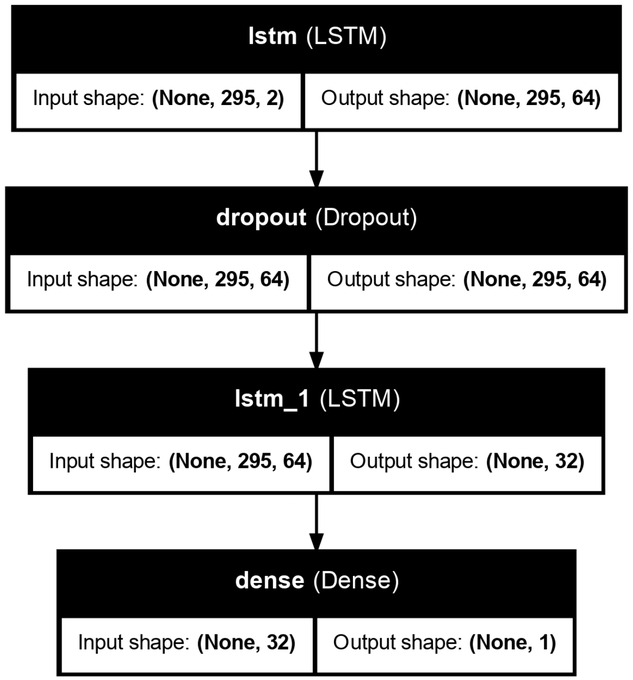
Checking the correctness

- Checking the accuracy of the Jacobian calculation against finite difference
- **Very good agreement found!**
- **Many many orders of magnitude faster than regenerating different samples**

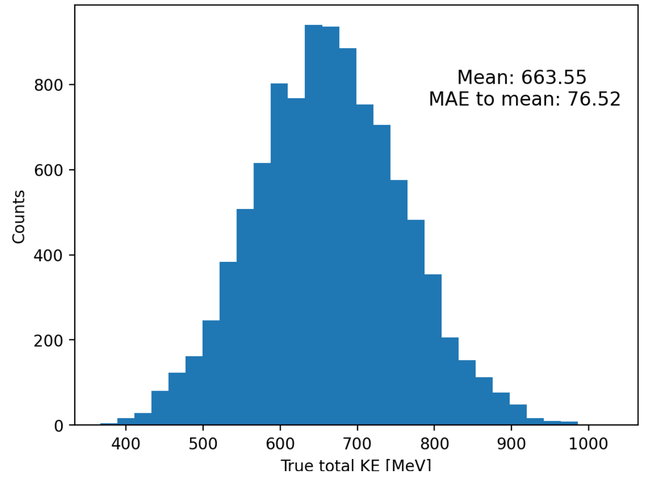


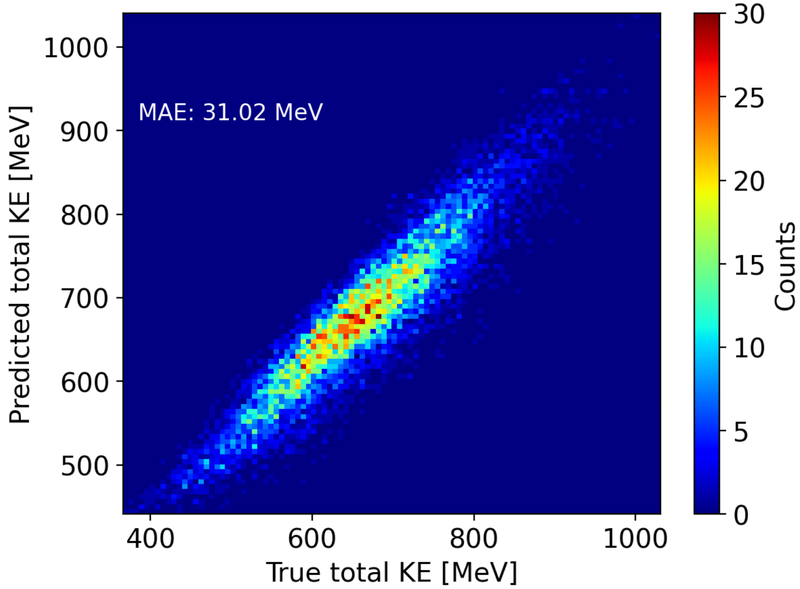
Adding reconstruction

Probably the most ridiculous model of the workshop



- Using 10k proton events in the 2x2 geometry
- Implemented some very basic toy reco model to **regress the particle energy** from the simulation output hits (ADC, time)





no probabilistic electronics simulation used here

Propagation to reco

Propagating uncertainties down to reach quantities

- By combining a differentiable reconstruction to the simulator, we can **propagate the derivatives up to the reconstructed quantities**
- Our reco network estimates the energy of the particle E based on the event output X : $g(X, \varepsilon) = \hat{E}$
- We can apply the same linear error propagation as before:

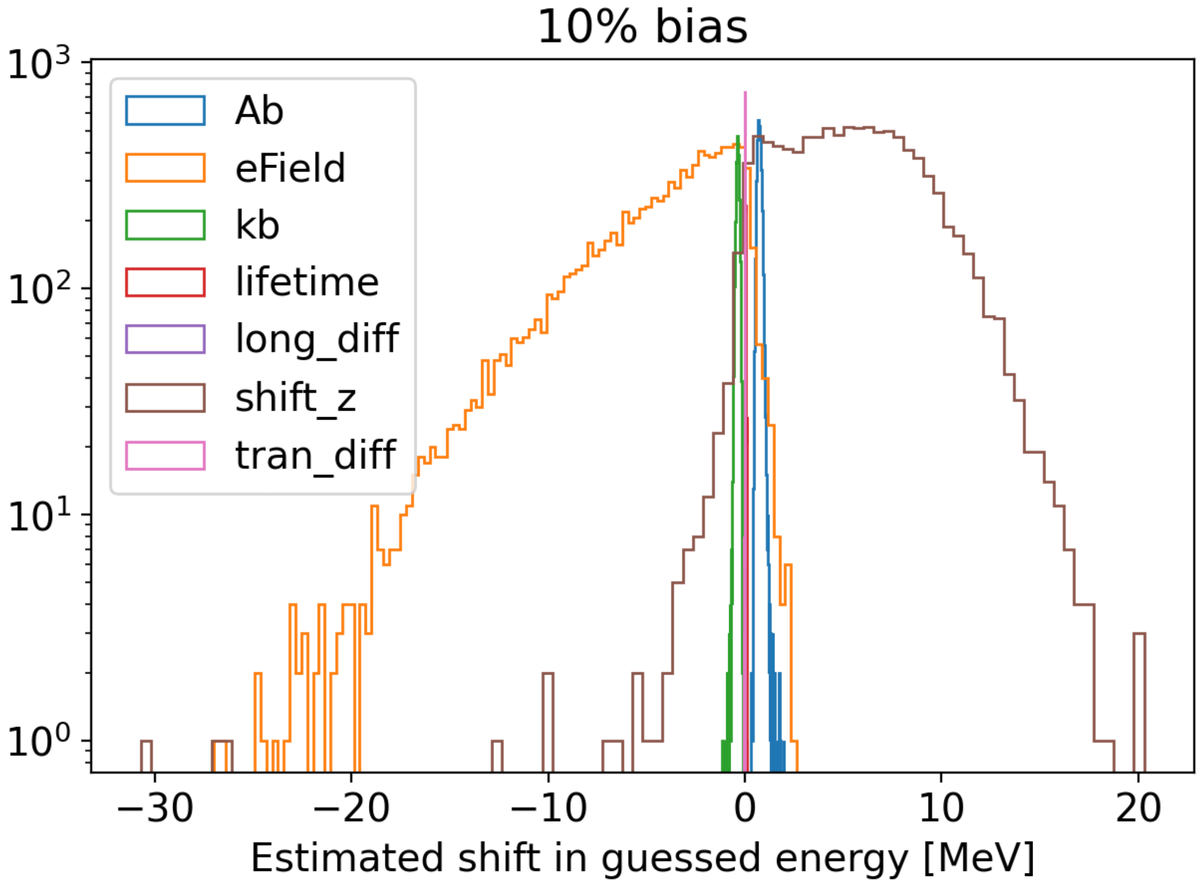
$$g(f(\chi, \theta + \delta\theta), \varepsilon) \simeq \hat{E} + \frac{\partial g}{\partial X}\frac{\partial f}{\partial \theta}\delta\theta$$

- Can be pushed to higher order and with **parameters correlations** by using directly the post-calibration obtained correlation matrix (through Hessian)

Propagation to reco

Practical application

- Using previous formula we can derive the estimated shift in the guessed energy when applying a **10% shift** to the simulated parameter
- The sensitivity to the different parameters is variable



Making the reco more robust

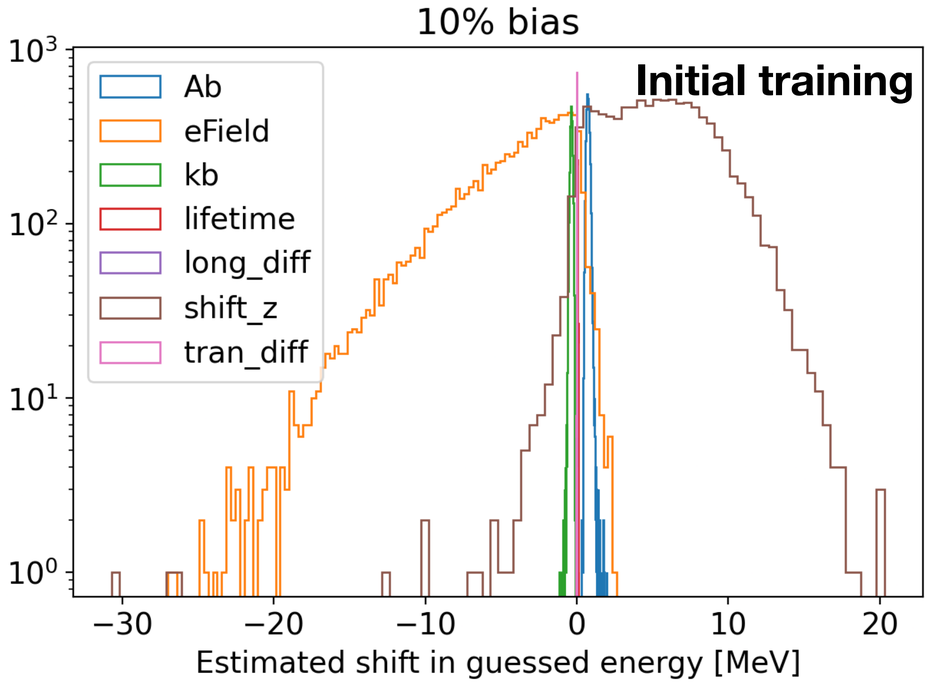
Penalizing large response variations

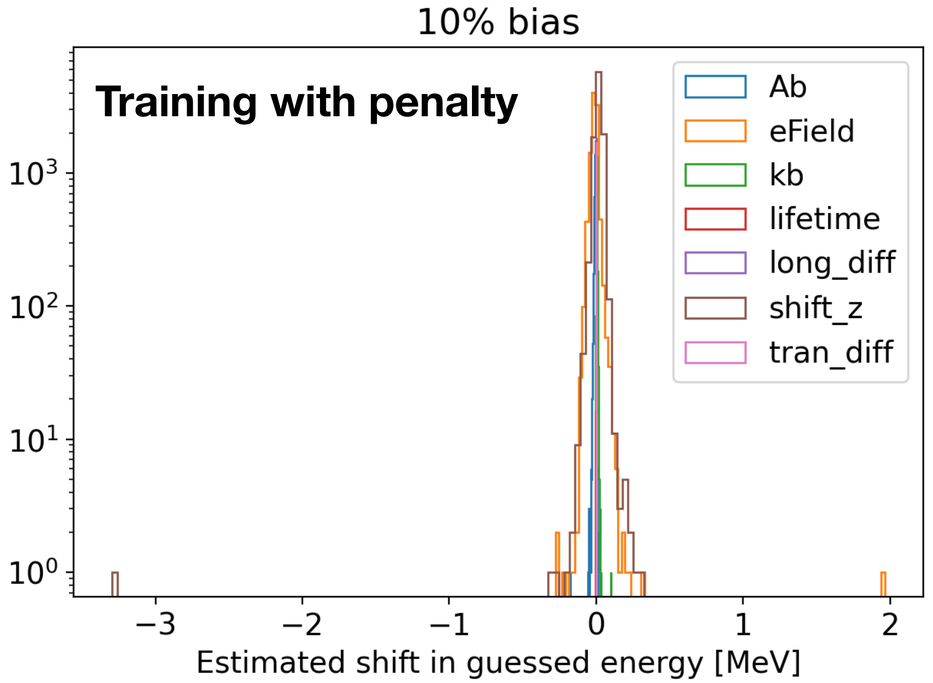
- Making the reconstruction **less sensitive** to the detector systematic uncertainties implies $\frac{\partial g}{\partial \theta} \simeq 0$
- This gradient can directly be obtained from the fully differentiable pipeline and it ca be added to the loss as a **penalty term** during the network training to have it small: $\mathscr{L} = \|g(X, \varepsilon) - E\| + \lambda \left\| \frac{\partial g}{\partial X} \frac{\partial X}{\partial \theta} \right\|$
- Effectively **teaching the network about possible variations of the data without requiring to run many variation simulations!**

Making the reco more robust

Results

- The network became **much less sensitive to variations** in simulation parameters after the addition of the penalty term
- It became more robust to known variations, while keeping **similar performance**





Summary

What we saw

- We have a **performant differentiable simulator** for pixel-based LArTPCs
- Still trying to improve it further to have **more accurate/stable gradients**
- Moving towards a **probabilistic treatment** of the electronics response
- The **seamless propagation of derivatives allows for many applications** such as calibration (see next talk), uncertainty propagation, reconstruction, …
- **Outlooks**: moving from preliminary studies to more direct applications of the uncertainty propagation, doing more verifications, …
- **Probably more possible applications** (open to suggestions!)