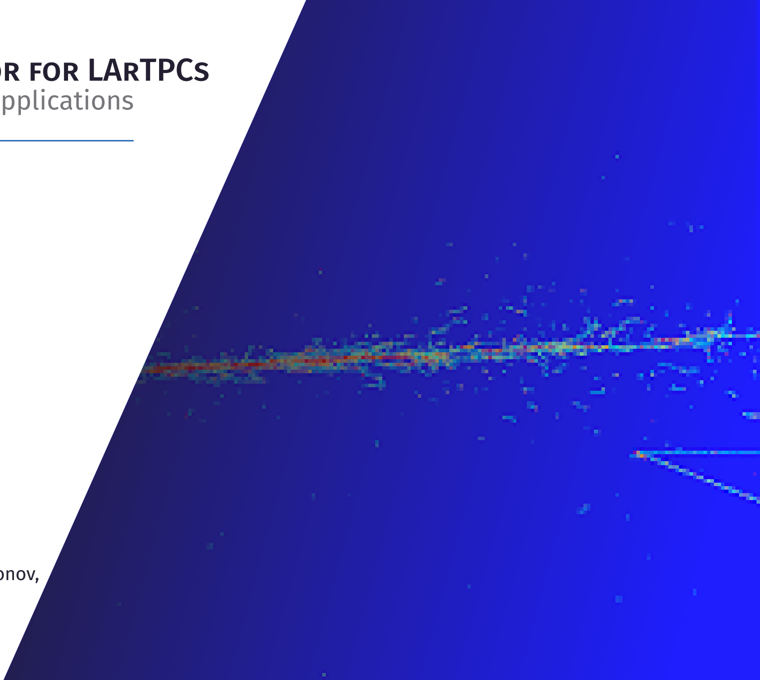
A DIFFERENTIABLE SIMULATOR FOR LARTPCS
from proof-of-concept to real applications

Pierre Granger
granger@apc.in2p3.fr

APC (Astroparticule et cosmologie) - Paris

June 28, 2024

Work from: Sean Gasiorowski, Yifan Chen, Youssef Nashed, Pierre Granger, Camelia Mironov, Daniel Ratner, Kazuhiro Terao, Ka Vang Tsang



Outline

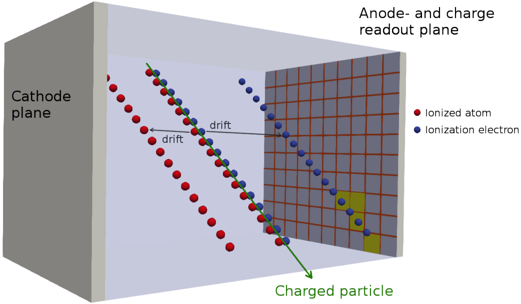
1. Motivation

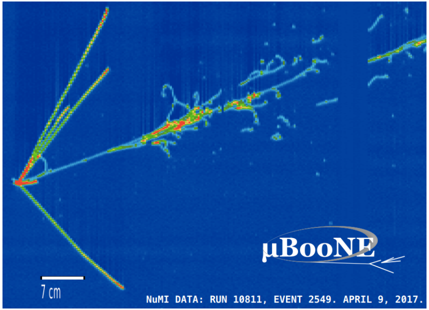
2. Recap of previous work

3. Improving the performance

4. Outlooks

LIQUID ARGON TIME PROJECTION CHAMBERS (LARTPCS)



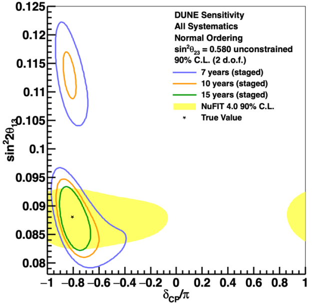


Signal production steps:

- Argon ionisation
- Ionisation electrons drifted by $\mathbf{E}$ field
- Electrons readout on anode plane

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

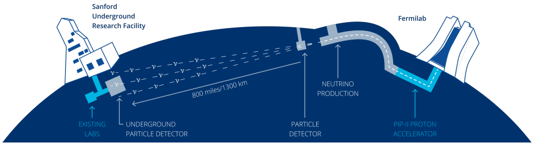
DUNE FOR PRECISION MEASUREMENTS

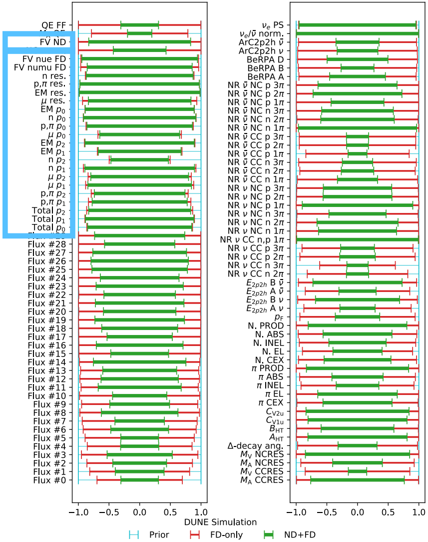


Systematics of detector modeling

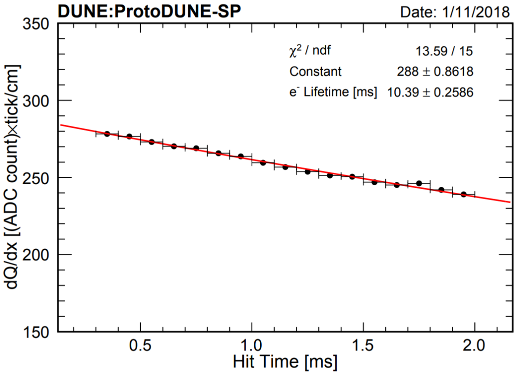
Well-understood detector modeling and calibration are vital

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

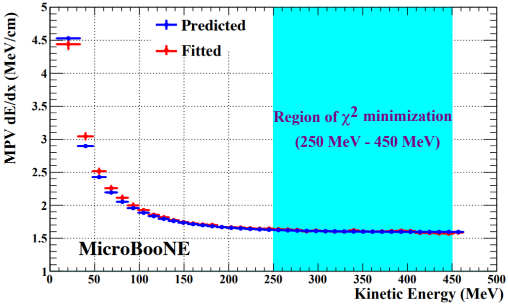




TYPICAL LARTPC CALIBRATION



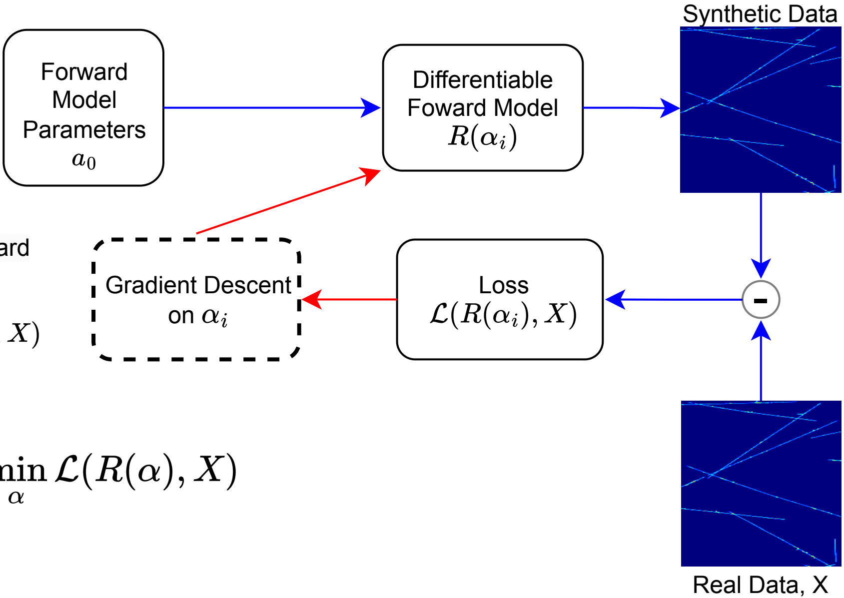
e^- lifetime calibration



Energy conversion calibration.

Calibration of the different physical parameters are typically done in **different studies**.
$\longrightarrow$ **can be simplified with a differentiable simulator**

Using gradient-based optimization



Requires forward model to be differentiable: $\nabla_{\alpha_i} \mathcal{L}(R(\alpha_i), X)$

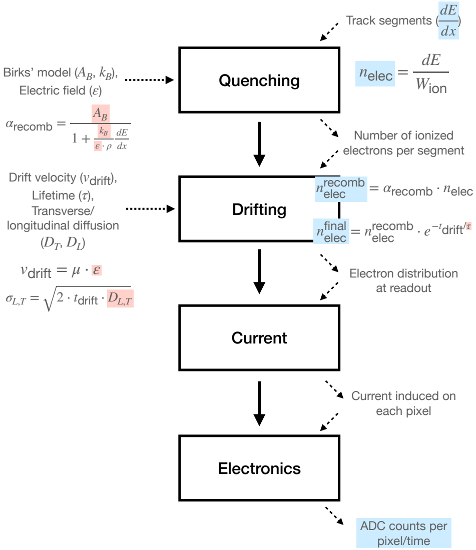
$$\alpha^* = \arg\min_{\alpha} \mathcal{L}(R(\alpha), X)$$

Starting from a non-differentiable LArTPC simulator

Our work: take existing DUNE near-detector simulation (JINST 18 P04034) and **make it differentiable.**

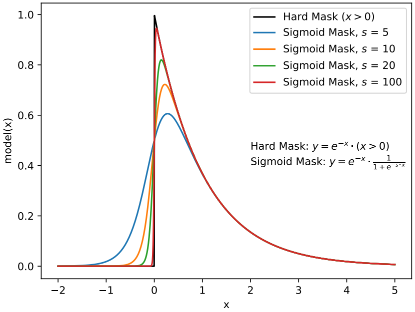
- Retain physics quality of a tool used collaboration-wide while adding ability to calculate gradient
- Demonstrate the use of this differentiable simulation for **gradient-based calibration**

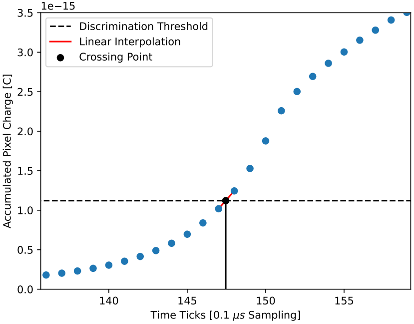
→ **How to do it practice?**



Differentiable relaxations

The base simulation contains **discrete operations** $\rightarrow$ non-differentiable.
Requires differentiable relaxations to be able to get usable gradients.





- Cuts (e.g. x > 0) $\rightarrow$ smooth sigmoid threshold
- Integer operations (e.g. floor division) $\rightarrow$ floating point (e.g. regular division)
- Discrete sampling $\rightarrow$ interpolation

Rewriting the simulator

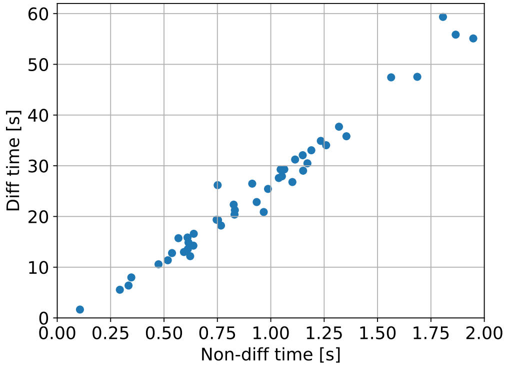
Numba code using **CUDA JIT compiled kernels** → Framework change for diff version:

- Differentiable version rewritten using EagerPy(backend agnostic)/PyTorch, which is based around tensor operations → use of autograd for automatic gradient calculations
- New version rewritten in a **vectorized** way to fit within these frameworks

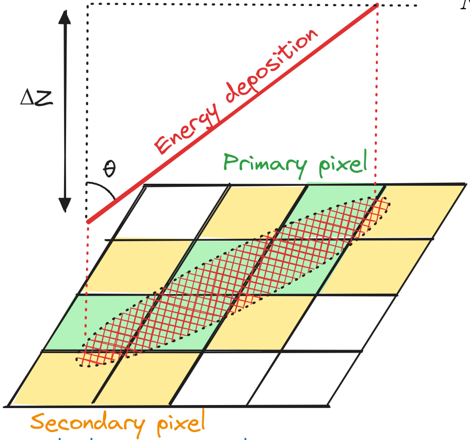
Performance drawbacks:

- Use of dense tensors to represent a sparse problem
- Moving from **CUDA JIT compiled dedicated kernel** to a **long chain of generic kernels** (vectorized operations).

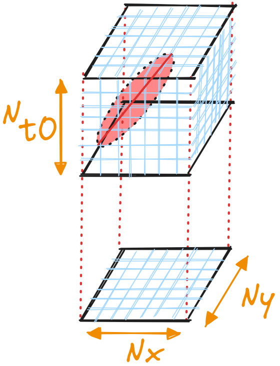
→ **also impacting memory usage**

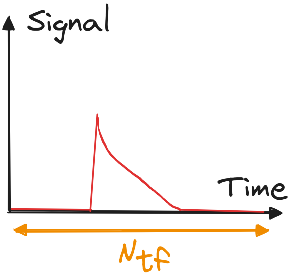


The simulator in more details



$N_{t0} \propto \sqrt{1 + \cot^2 \theta}$ $\quad N_{tf} \propto \Delta z$



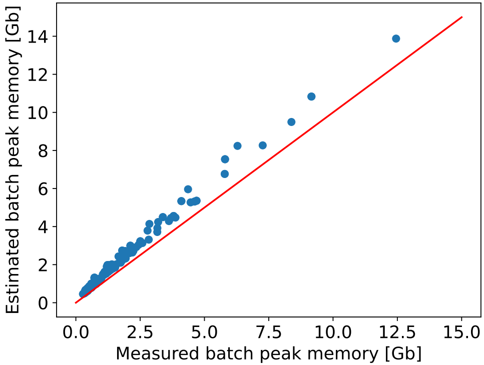


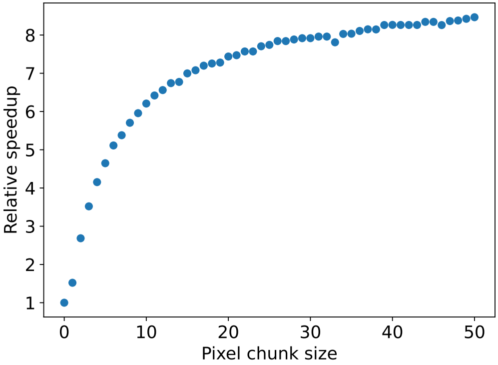
1. Calculation of N electrons
2. Estimation of charge spreading
3. Finding primary pixels
4. Finding secondary pixels
5. For each segment-pixel pair
6. Compute the intersection
7. Compute charge per voxel (Δx Δy Δt0)
8. Compute signal for all tf

MEMORY CHALLENGE

$$\mathcal{M} = \overbrace{N_{\text{segments}} \times N_{\text{pixels}}}^{\text{chunk}} \times N_{t0} \times N_{tf} \times N_x \times N_y$$

Because of the use of dense tensors, **memory** $\propto \boldsymbol{\Delta_z} \times \sqrt{\mathbf{1 + \cot^2 \theta}}$. (length in drift direction and angle) $\rightarrow$ introduced **automatic memory estimation** for each batch to estimate best pixel chunk size.



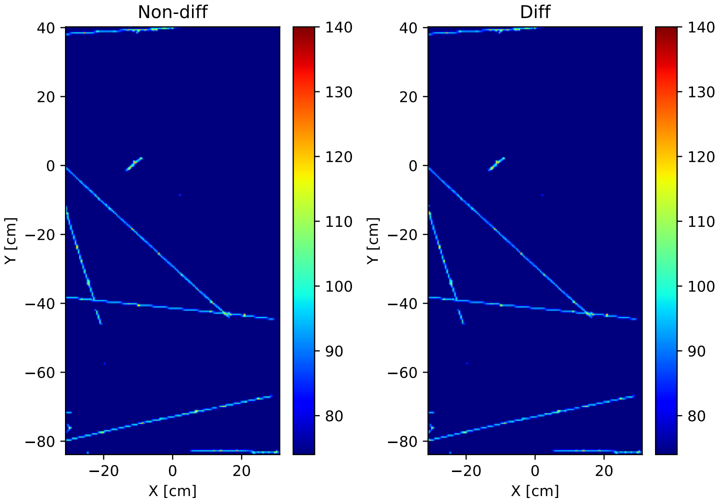


Trade-off between memory and computation time $\rightarrow$ use of gradient checkpointing

Checking the result

Checking that the **relaxations don't modify the simulator output.**

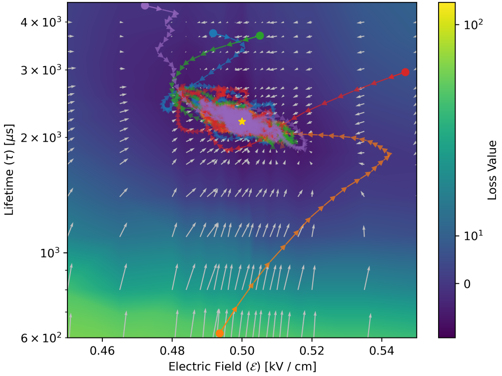
Average deviation of 0.04 ADC/pixel $\rightarrow$ well below the typical noise level of few ADCs.



Optimization of the Model Parameters

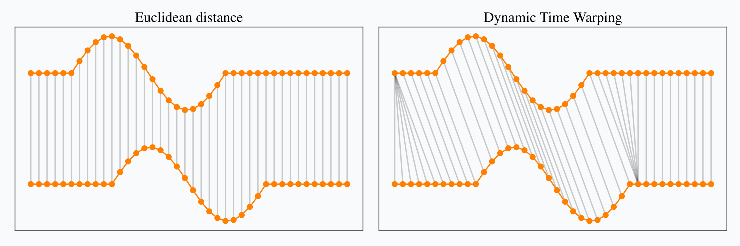
- Input particle segments (position and energy deposition): χ
- Model parameters: θ
- Differentiable simulation: $f(\chi, \theta)$
- Target data: F_{target}

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 $\mathcal{L}(f(\chi, \theta_0), F_{\text{target}})$
4. Calculate gradients for the parameters $\nabla_\theta \mathcal{L}(f(\chi, \theta_0), F_{\text{target}})$
5. Update parameter values $\theta_0 \to \theta_i$ to minimize the loss
   Iterate step 2. to 5.



Optimization choices: Loss function

Loss function choice is crucial for minimization quality



https://rtavenar.github.io/blog/dtw.html

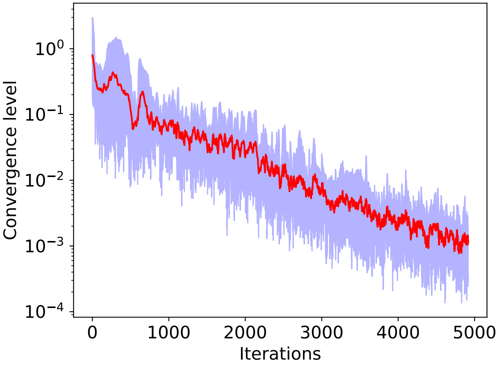
Two main ways of computing the loss:

- Comparison of 3D voxel grids of charges (x, y, t $\rightarrow$ z, q).
  - Difficulty of taking gradients through discrete pixelization.
  - Risk of flat loss if not enough overlap in distributions.
- **Considering the waveforms for each pixel (time sequence) and using Dynamic Time Warping**
  - Using a relaxed SoftDTW version that is differentiable.

Results

Mach.Learn.Sci.Tech. 5 (2024) 2, 025012

- Input sample consisting of 1 GeV simulated muon tracks
- Second sample of muons, pions and protons (1 GeV to 3 GeV)
- Geometry of a DUNE ND-LAr-prototype module: 60 cm $\times$ 60 cm $\times$ 120 cm
- Noise model available in simulator but not used.



6D simulteanous fit converging under L_∞

Doing a "closure test" based on simulated data, $F_{\text{target}} = f(\chi, \theta_{\text{target}})$:
$\rightarrow$ Fit of 6 physical parameters **simulteanously** on simulated data for multiple targets.

1. Motivation

2. Recap of previous work

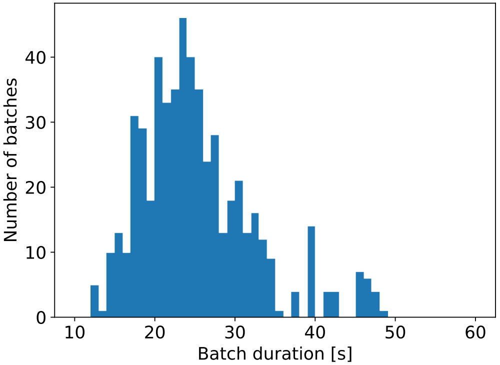
3. Improving the performance

4. Outlooks

Improving the performance: why?

Current simulator performance are limiting for future applications:

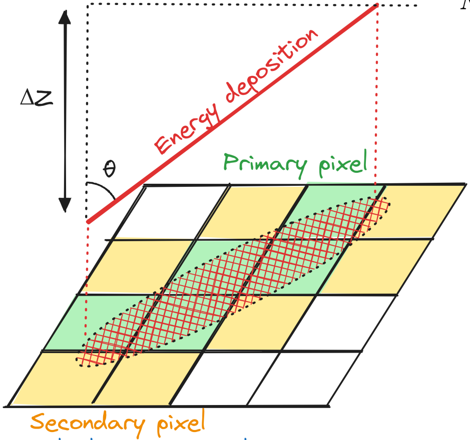
- Application to real data (Yifan pres.) $\rightarrow$ large batches and quantity of data required to mitigate the effects of electronics noise
- Being able to have more complicated physical models: inhomogeneous drift fields, space charge effect, ...
- Running the code on less demanding hardware (major limitation on memory)
- Allowing to ease uncertainty quantification: running multiple fits with different seeds, computing result on whole distributions, ...

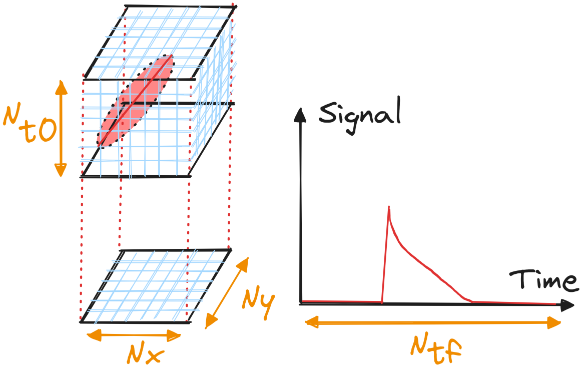


$\sim$ 25 s to process a 100 cm batch $\rightarrow$ $\sim$ 30 h for a full fit (5000 iterations)

Bottleneck of the initial code

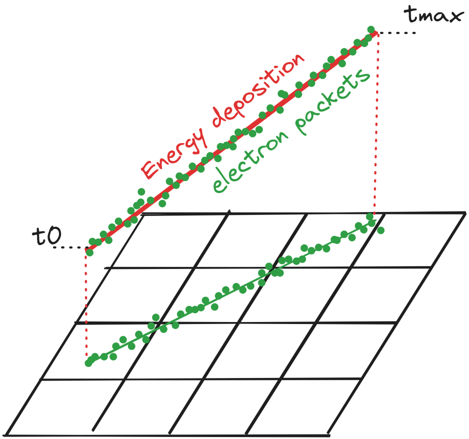
$N_{t0} \propto \sqrt{1+\cot^2\theta}$ $\qquad$ $N_{tf} \propto \Delta z$

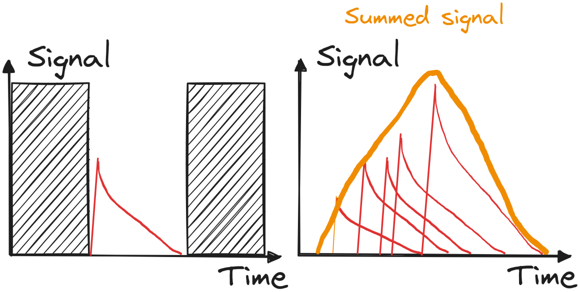




1. Calculation of N electrons
2. Estimation of charge spreading
3. Finding primary pixels
4. Finding secondary pixels
5. For each segment-pixel pair
6. Compute the intersection
7. Compute charge per voxel (Δx Δy $\Delta t0$)
8. Compute signal for all tf

Redisigning the simulation code





1. Calculation of N electrons
2. Generation of MC e- packets
3. Get t0, x, y of e- packet
4. Get associated xpad ypad
5. For each e- packet compute waveform
6. Realign and sum waveforms per pixel

Change of framework

Benefit of the code redesign to rewrite with a new framework: JAX

Why JAX:

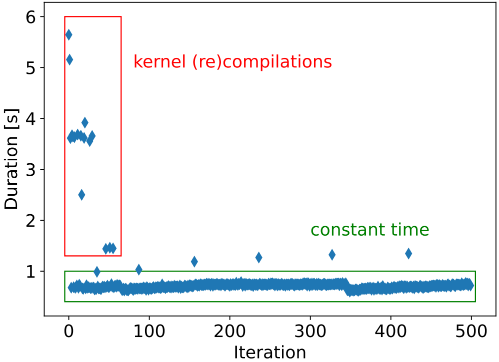
- Allows for ”easy” Just In Time kernel compilation
- Efficient calculation of the gradient calculation graph (XLA)
- Runs indifferently on CPU/GPU

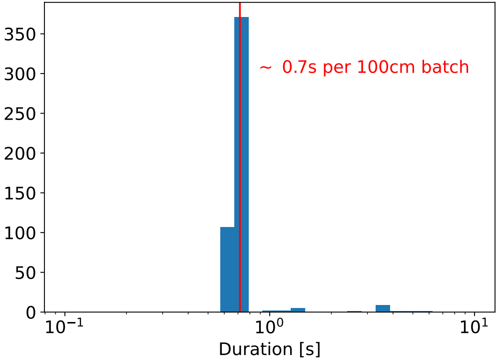
Requirements for ”easy” JIT

- No use of basic control-flow
- Loops must have a defined number of iterations
- No dynamic-shapes: the shape of all the tensors must be known at compile time $\Longrightarrow$ recompilation of kernels on each shape change

$\rightarrow$ implemented a ”shape memory” to pad inputs to nearby shapes if already compiled kernels exist to limit the number of kernel recompilations (computationally expensive)

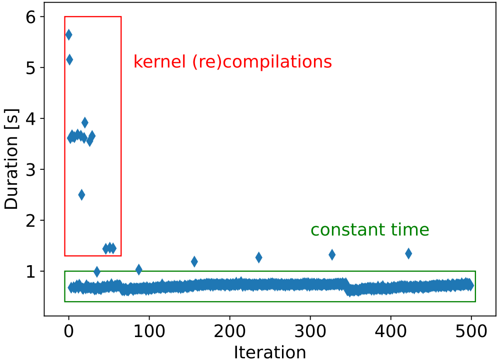
NEW PERFORMANCE

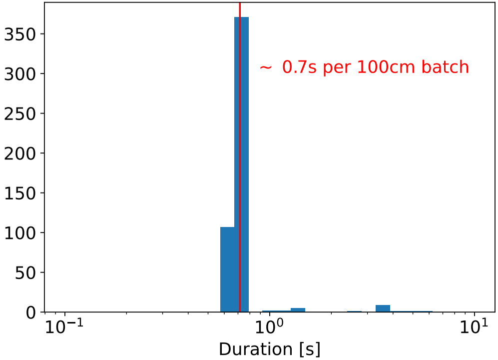




- After some iterations, kernels are already compiled for a wide range of shapes $\rightarrow$ no more overhead
- With the rework, the computation time is very strongly driven by the batch size only $\rightarrow$ almost constant computation time
- Speedup of $\sim \times 35$ $\rightarrow$ allows for a full fit in $\sim 1\,\mathrm{h}$

New performance

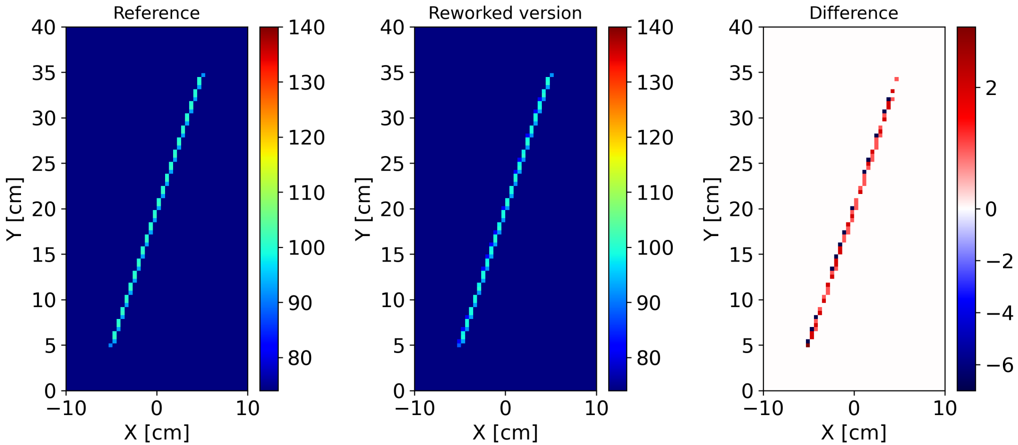




- After some iterations, kernels are already compiled for a wide range of shapes $\rightarrow$ no more overhead
- With the rework, the computation time is very strongly driven by the batch size only $\rightarrow$ almost constant computation time
- Speedup of $\sim \times 35$ $\rightarrow$ allows for a full fit in ~ 1 h
- On CPU only...

Current simulation

- Running the reworked code on GPU is $\sim 5\times$ slower than on CPU... To be investigated.
  $\rightarrow$ possibly leaves open greater computation time improvements if understood/solved
- Final checks on the correctness wrt previous simulation



Complex comparison: fixes applied wrt "Reference", $\neq$ signal simulation wrt newest larnd-sim

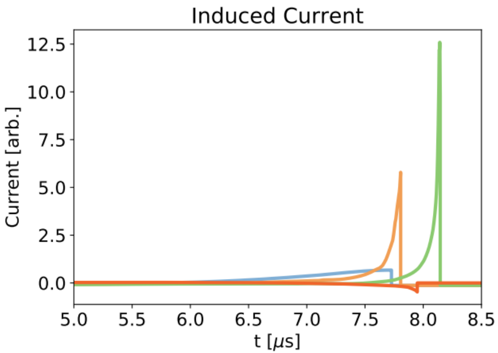
Current simulation

Old larndsim

$$I(x,y,t,t_0) = \frac{P(x,y)}{R(x,y)} e^{\frac{t_0+Q(x,y)-t}{R(x,y)}} + \frac{1-P(x,y)}{S(x,y)} e^{\frac{t_0+Q(x,y)-t}{S(x,y)}}$$

Analytical approximation of the induced current

New larndsim

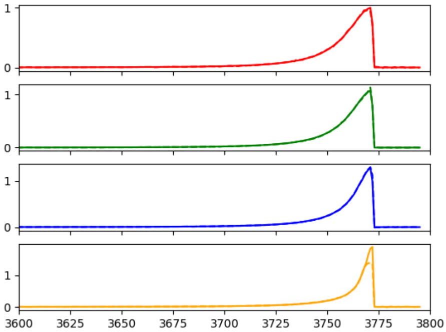


Pre-computed signals as lookup table (more accurate)

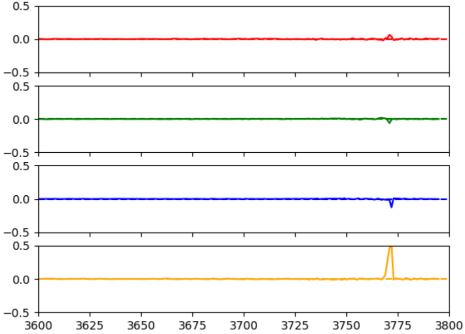
Lookup table is not differentiable, need to find another implementation

Outlooks

Ongoing work from Dan Douglas to develop a surrogate to replace the lookup table using SIREN



Waveforms: LUT (plain) and SIREN (dashed)



Residuals

Surrogate not ideal yet → probably due to the too coarse sampling of LUT

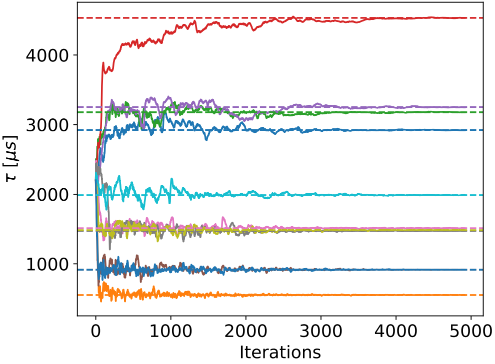
Re-running the field simulation to have a smoother input to train SIREN on

Uncertainty quantification: why?

We can make successfully make a calibration fit on simulation. How to quote an uncertainty on the obtained value?

Several uncertainties we might want to take into account:

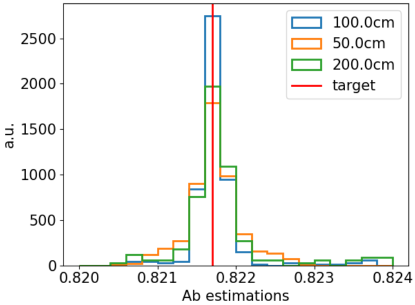
- Uncertainty on the physical parameters / physical processes
- Uncertainty on the true energy deposits (inaccessible in data)
- Stochasticity due to noise



UQ: Uncertainty on the calibrated parameters

Estimating the uncertainty on the calibrated parameters:

- Computing the Hessian matrix to estimate the parameters error (easily accessible in a differentiable simulator)
- Profiling the fitted value after convergence
- Running multiple fits in parallel and compare the convergences (ensembling)



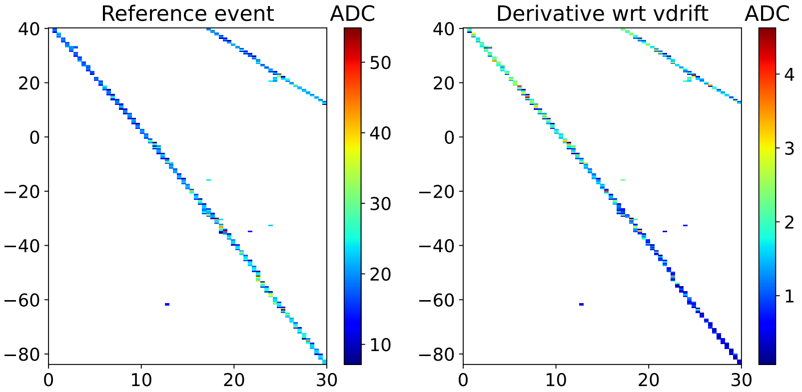
$$\mathbf{H}_f = \begin{bmatrix} \dfrac{\partial^2 f}{\partial x_1^2} & \dfrac{\partial^2 f}{\partial x_1\,\partial x_2} & \cdots & \dfrac{\partial^2 f}{\partial x_1\,\partial x_n} \\[2.2ex] \dfrac{\partial^2 f}{\partial x_2\,\partial x_1} & \dfrac{\partial^2 f}{\partial x_2^2} & \cdots & \dfrac{\partial^2 f}{\partial x_2\,\partial x_n} \\[2.2ex] \vdots & \vdots & \ddots & \vdots \\[2.2ex] \dfrac{\partial^2 f}{\partial x_n\,\partial x_1} & \dfrac{\partial^2 f}{\partial x_n\,\partial x_2} & \cdots & \dfrac{\partial^2 f}{\partial x_n^2} \end{bmatrix}.$$

$$E = \begin{bmatrix} \sigma_1^2 & \sigma_{12}^2 & \cdots \\ \sigma_{21}^2 & \sigma_2^2 & \\ \vdots & & \ddots \end{bmatrix} = 2H_f^{-1}$$

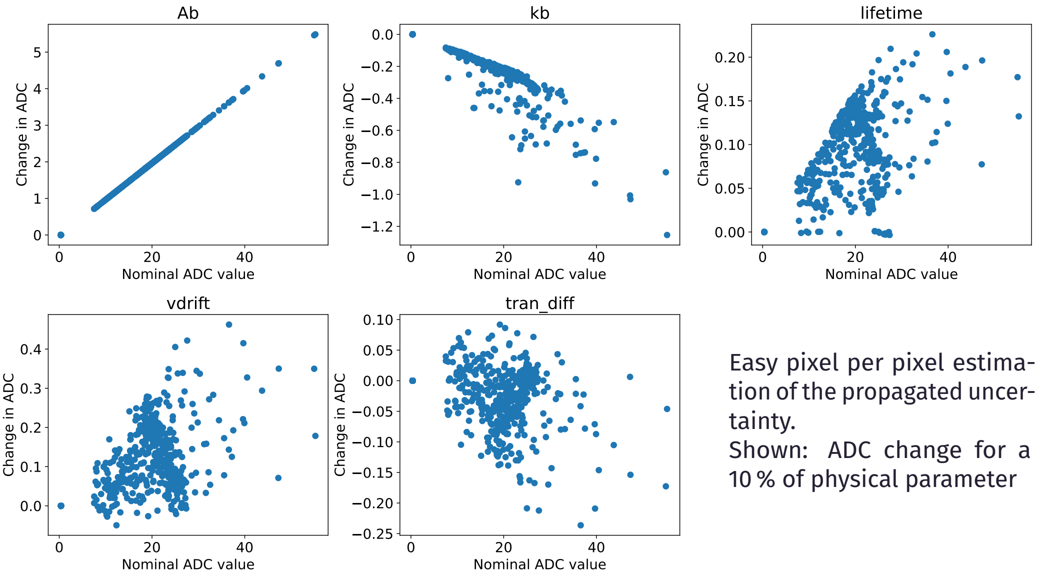
UQ: Linear error propagation

$$q(x_i) \implies \sigma_q^2 = \sum_i \left(\frac{\partial q}{\partial x_i} \sigma_{x_i} \right)^2$$

Linear error propagation allows for an estimation of the output uncertainty based on the input parameters uncertainties $\rightarrow$ Only requires the already available derivatives



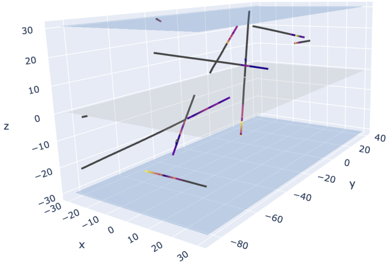
UQ: Linear error propagation



Easy pixel per pixel estimation of the propagated uncertainty.
Shown: ADC change for a 10 % of physical parameter

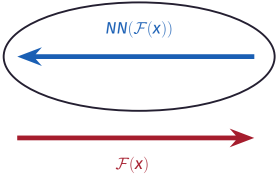
GOING FURTHER

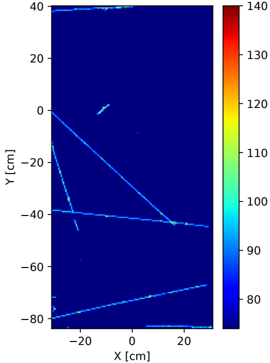
Combining our differentiable simulator with an inverse mapping would allow for direct model constraining, fully data driven: $\mathcal{L}_{CC} = (\mathcal{F}(NN(y_{\text{data}})) - y_{\text{data}})^2$



Energy deposits $\mathrm{d}E/\mathrm{d}x$
(inaccessible in data)

Inverse mapping step to developp





Detector readout

Might allow to improve the calibration by reconstructing the true energy deposits → important role of uncertainties (see Dan's talk today)

Conclusions

Shown here:

- Proof of concept for the calibration of a LArTPC using a differentiable simulator.
- Multidimensional fit converging correctly on simulated data with the differentiable simulator.
- Simulator rewriting allows to reach way better performance $\rightarrow$ will be important for application to real data

Upcoming challenges:

- Applying this framework to real data (DUNE 2x2 ND data) $\rightarrow$ see Yifan's talk
- Fitting more physical parameters (such as Efield map)
- Uncertainty quantification and propagation
- Inverse problem solving in the future

Pierre Granger

June 28, 2024

granger@apc.in2p3.fr