NPML 2024

Neutrino Physics & Machine Learning 2024 *Zürich, 28th June 2024*

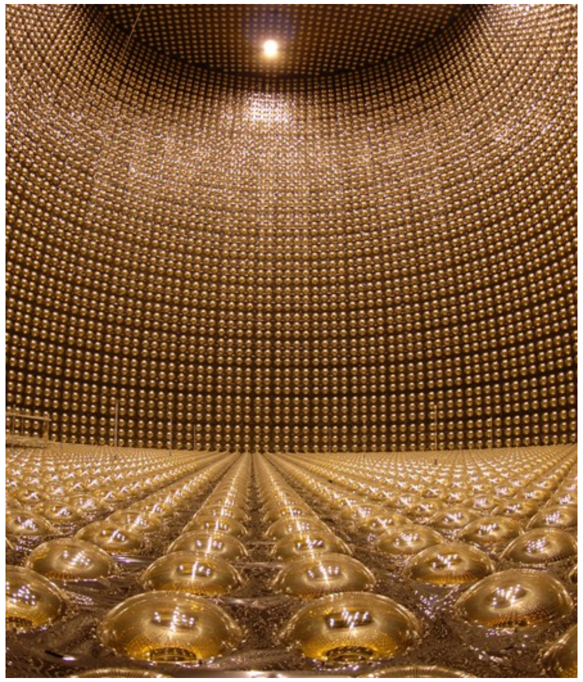
Advancing Detector Calibration and Event Reconstruction in Water Cherenkov Neutrino Detectors with Analytical Differentiable Simulations

César JESÚS-VALLS & Omar ALTERKAIT

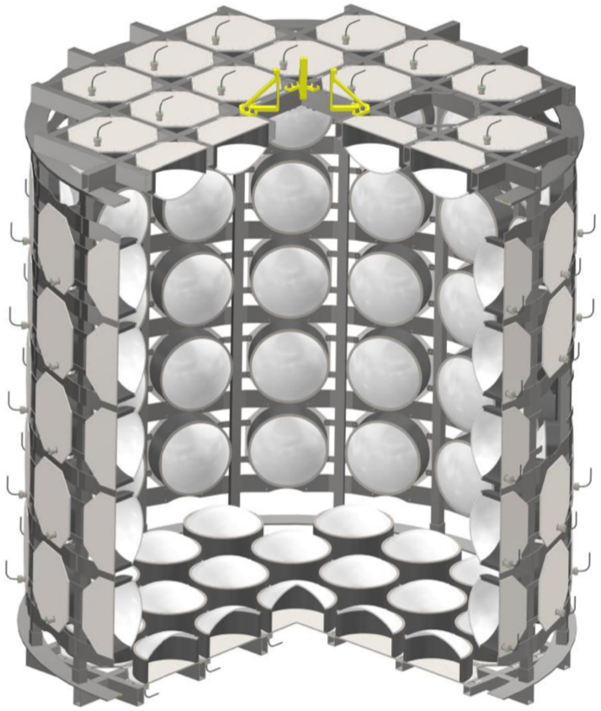
cesar.jesus-valls@ipmu.jp

On behalf of CIDER-ML collaboration

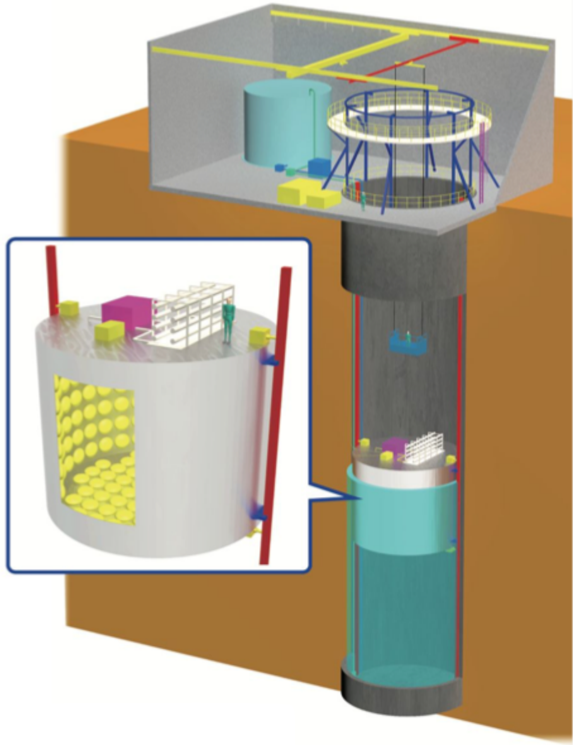
SUPER-K



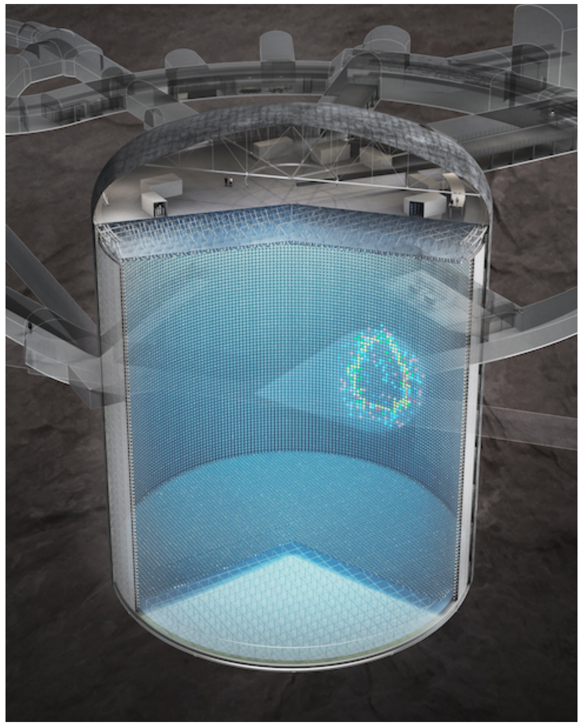
WCTE



IWCD



HYPER-K



Cherenkov detectors are crucial in neutrino physics.

Well understood technology, inheriting software & analysis tools from the past.

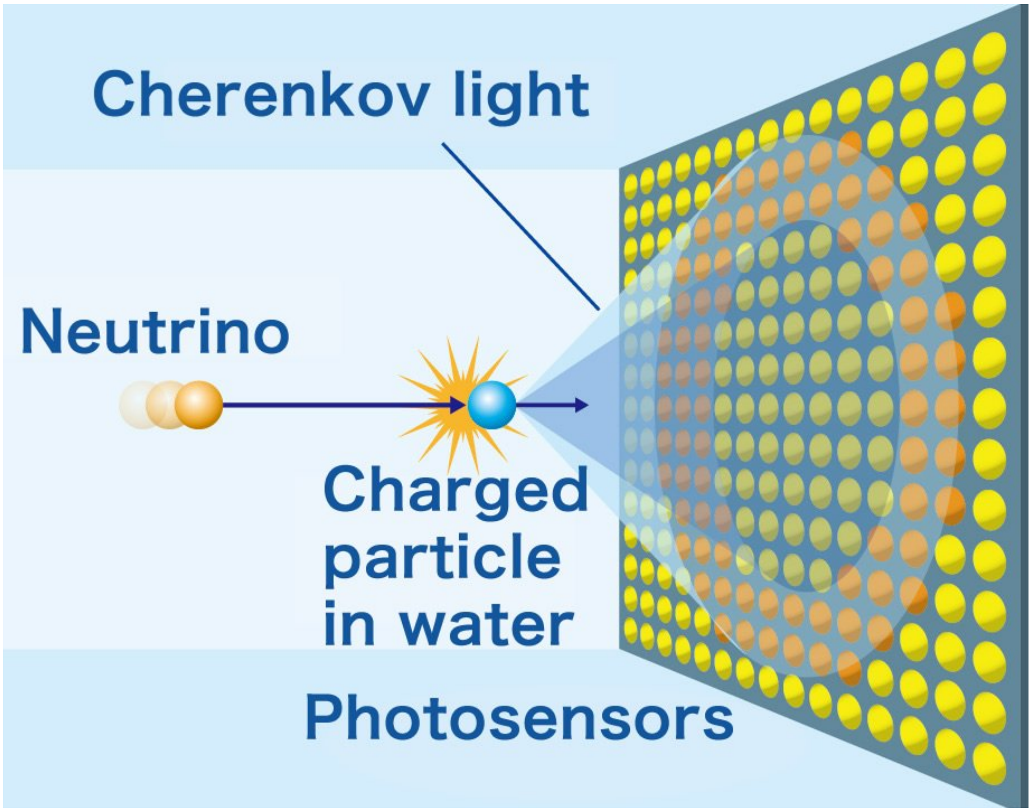
Moving towards the future we want to find ways to benefit from recent developments in AI/ML.

Particles traveling faster than speed of light in the medium produce Cherenkov light.

Cherenkov Threshold ***Emission Angle (usually ~42° in water)***

Track Reconstruction

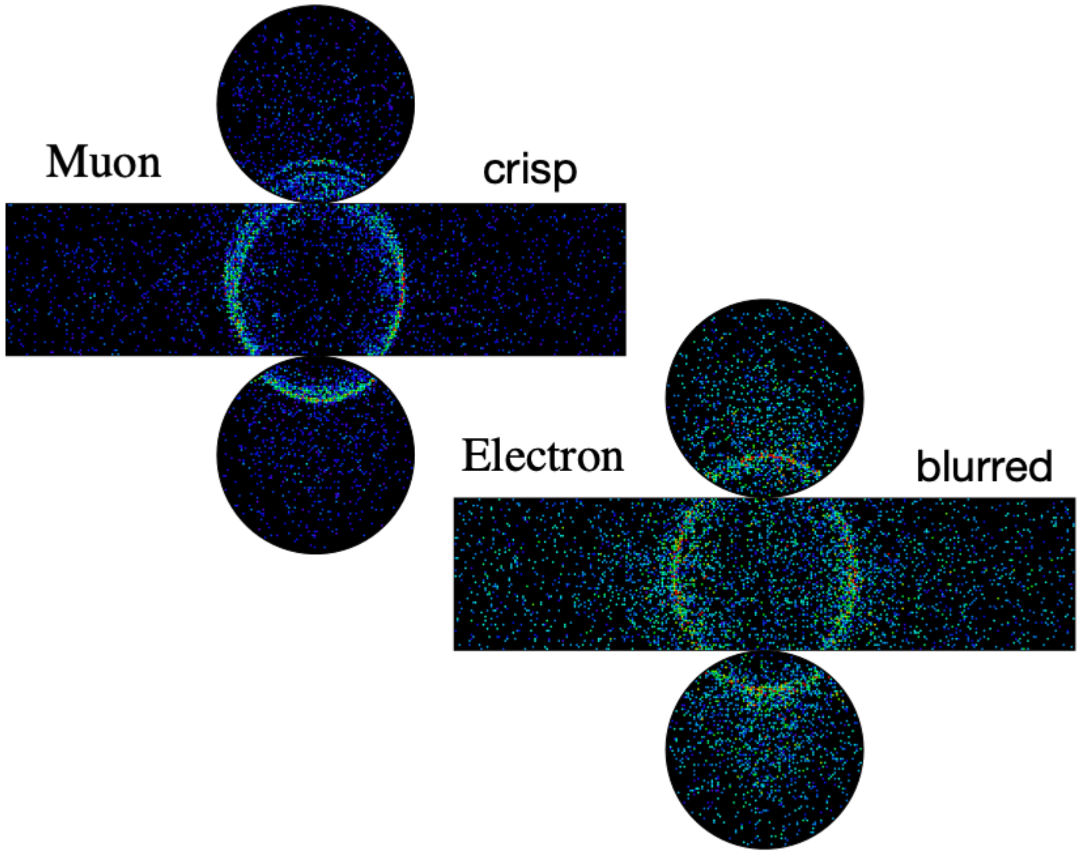
Ring timing	*~vertex position.*
Ring thickness	*~track length.*
Ring orientation	*~track direction.*
Ring shape / density	*~particle type.*



In Medium/Detector Effects

Examples:

Attenuation
Reflections
Detector Anisotropies
Sensor Heterogeneity
Hadron/Lepton SI



STANDARD ANALYSIS WORKFLOW

1 CALIBRATION(S)
Collection of stand-alone scripts

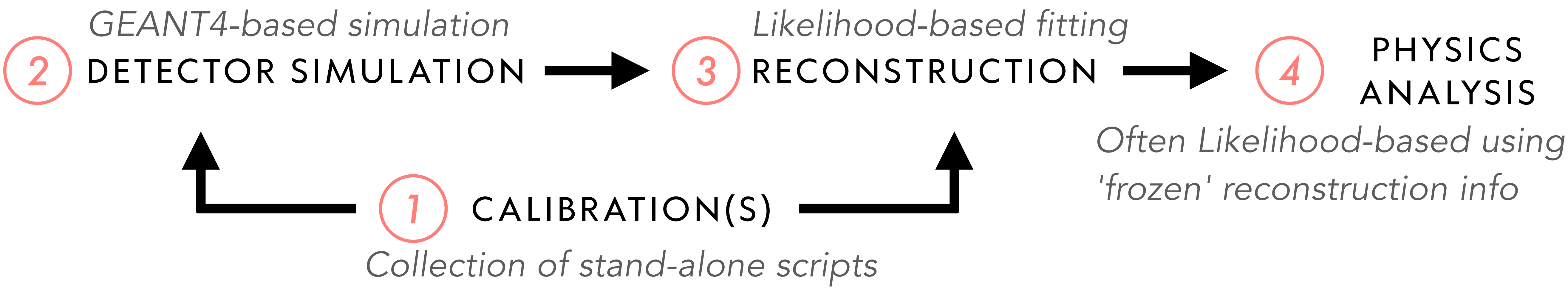
2 DETECTOR SIMULATION
GEANT4-based simulation

3 RECONSTRUCTION
Likelihood-based fitting

4 PHYSICS ANALYSIS
Often Likelihood-based using 'frozen' reconstruction info

CHALLENGES

STANDARD ANALYSIS WORKFLOW



PIPELINE FRAGMENTATION

Running experiments involves developing & maintaining isolated pieces of code, with different dependencies, different programing languages...

EFFORT DUPLICATION

Different runs, involve re-running calibrations, simulations & reconstruction. Time consuming & computationally heavy. Inefficient use of human & computing resources.

CORRELATION BLINDNES

Usual practice is to tune one part, freeze it, tune the next, etc. But what is the parts are co-dependent?

AN ALTERNATIVE APPROACH

PHYSICS IS A 'FORWARD' PROCESS

Physics → *Detector Hardware* → *Data*

ANALYSIS IS A 'BACKWARD' PROCESS

Physics ← *Detector Simulation & Reconstruction* ← *Data*

Currently multi-step & non-differentiable

AN ALTERNATIVE APPROACH

PHYSICS IS A 'FORWARD' PROCESS

Physics → *Detector Hardware* → *Data*

ANALYSIS IS A 'BACKWARD' PROCESS

Physics ← *Detector Simulation & Reconstruction* ← *Data*

Currently multi-step & non-differentiable

What could be better?

USE A DIFFERENTIABLE PHYSICS MODEL

Physics → $f(\vec{\theta})$ *unified & differentiable* → *Data*

Physics ← *Data*

$$\vec{\theta}_t = \vec{\theta}_{t-1} - \lambda \nabla_{\vec{\theta}} \mathscr{L}(x, \vec{\theta})$$

We can try to learn this ~implicitly (see J.Xia's Talk!)
Or (in this talk) we can try to do this analytically (i.e. enforcing our 'knowledge' explicitly.)
Or we can combine both approaches.

WHAT IS EXACTLY $f(\vec{\theta})$?

Implicit: $f(\vec{\theta})$ is a NN.

Generally, NN will be a black-box transformation from physics-space to data-space (with pros & cons). E.g. If you learn $f(\vec{\theta})$ you can perfectly match the performance of your detector, but you can't access intermediate information.

WHAT IS EXACTLY $f(\vec{\theta})$?

Implicit: $f(\vec{\theta})$ is a NN.

Generally, NN will be a black-box transformation from physics-space to data-space (with pros & cons). E.g. If you learn $f(\vec{\theta})$ you can perfectly match the performance of your detector, but you can't access intermediate information.

Analytic: $f(\vec{\theta})$ is a full forward model implementation based on an analytical description of all the processes.

You can optimize and inspect all of the model parameters. You run it on calibration data to tune the simulation. You run it on physics data for reconstruction. It is very easy to propagate systematics.

WHAT IS EXACTLY $f(\vec{\theta})$?

Implicit: $f(\vec{\theta})$ is a NN.

Generally, NN will be a black-box transformation from physics-space to data-space (with pros & cons). E.g. If you learn $f(\vec{\theta})$ you can perfectly match the performance of your detector, but you can't access intermediate information.

Analytic: $f(\vec{\theta})$ is a full forward model implementation based on an analytical description of all the processes.

You can optimize and inspect all of the model parameters. You run it on calibration data to tune the simulation. You run it on physics data for reconstruction. It is very easy to propagate systematics.

A TOY PROBLEM USING CHERENKOV PHYSICS

Our differentiable physics model needs to consider the following elements:

1. DETECTOR GEOMETRY
2. INITIAL CONDITIONS *(Particle(s) Kinematics & Positions)*
3. PARTICLE PROPAGATION
4. PHOTON PROPAGATION
5. DETECTOR RESPONSE

CHOOSING A FRAMEWORK

What simplifications can we do to isolate the 'core' problems of interest?

Priority is to code up differentiable photon propagation & detector response.

What framework should we use?

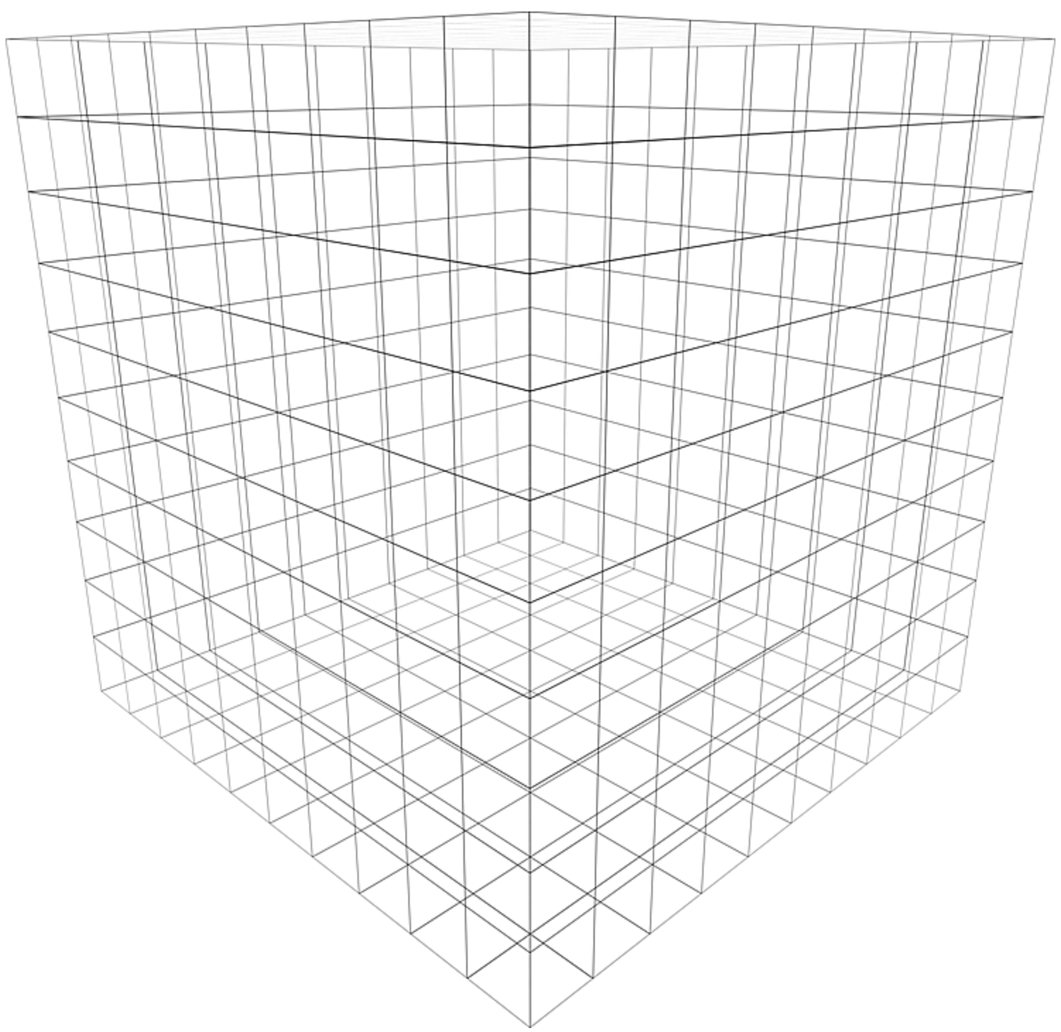
JAX TensorFlow Zygote Clad PyTorch

We want something with 1) built-in autodiff 2) ~geometry functionality.

Let's look at Industry, what do they use in ray-tracing applications?

Taichi Lang https://github.com/taichi-dev/taichi

1 DETECTOR GEOMETRY



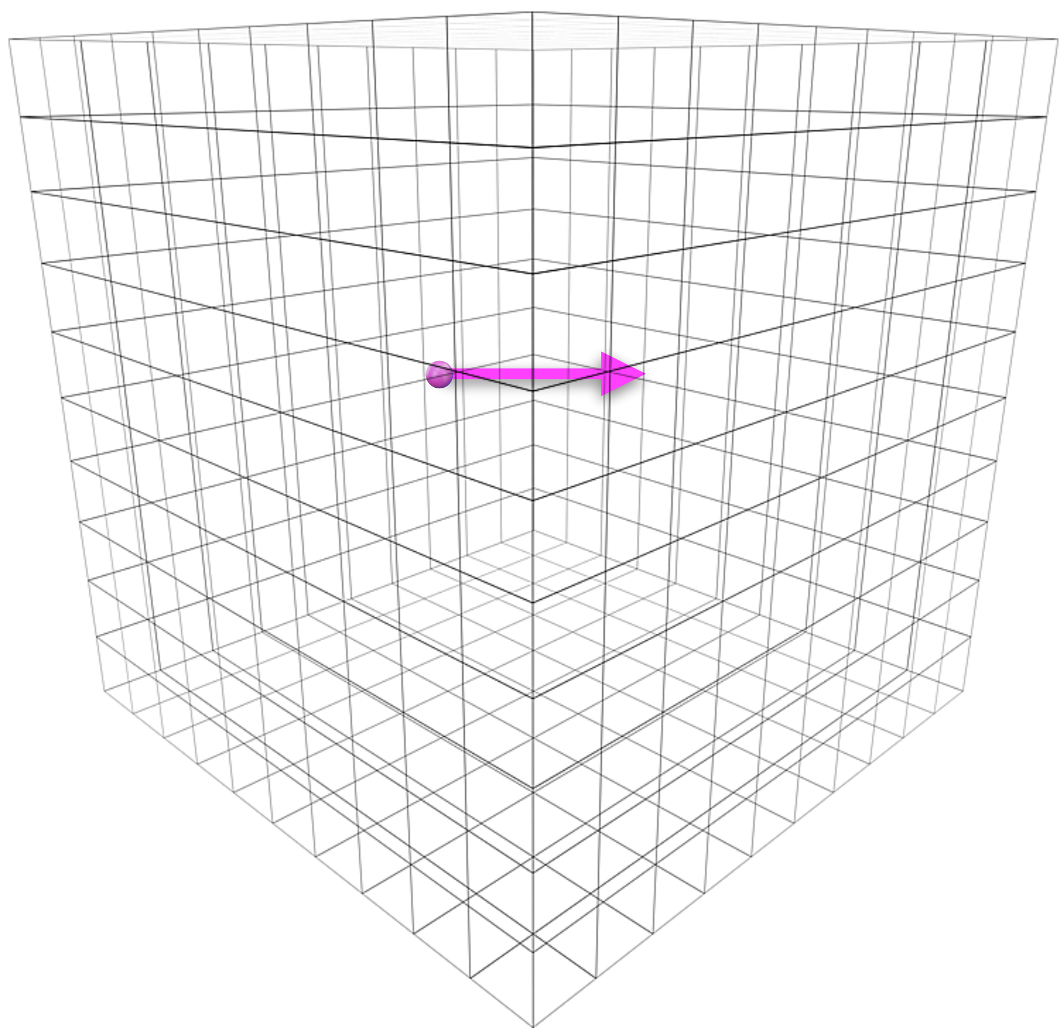
Let's assume a cubic detector made up of NxNxN voxels.
(nominally N=128)

Let's assume there are two types of voxels:

1) Detector Voxels → *All inner voxels*

2) Sensor Voxels → *All voxels in the surface*

2 INITIAL CONDITIONS & 3 PARTICLE PROPAGATION

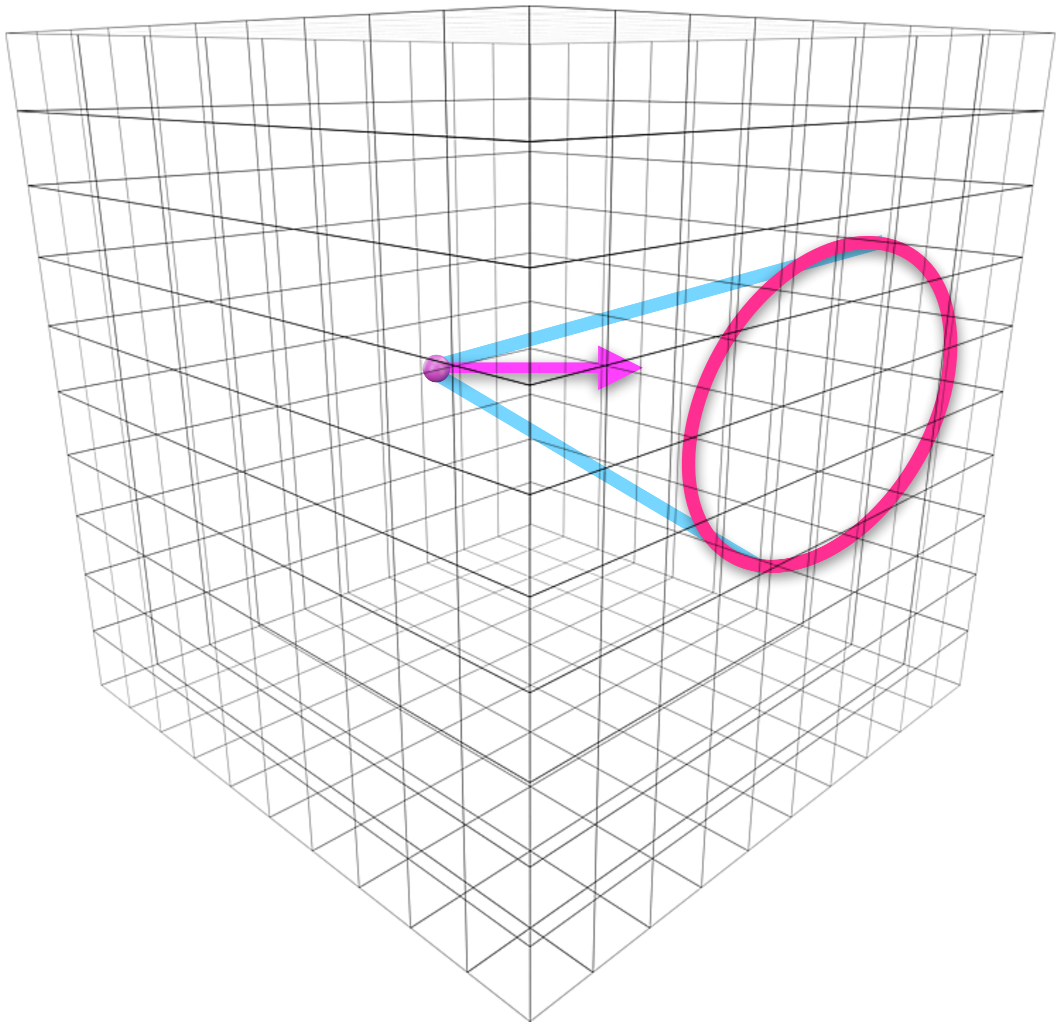


Let's assume 1 particle, starting in any inner voxel with any inner direction.

Let's assume the particle has infinitesimal length (no particle propagation).

6 Degrees of freedom:
(position $\vec{x}$, and direction $\vec{v}$)

4 PHOTON PROPAGATION

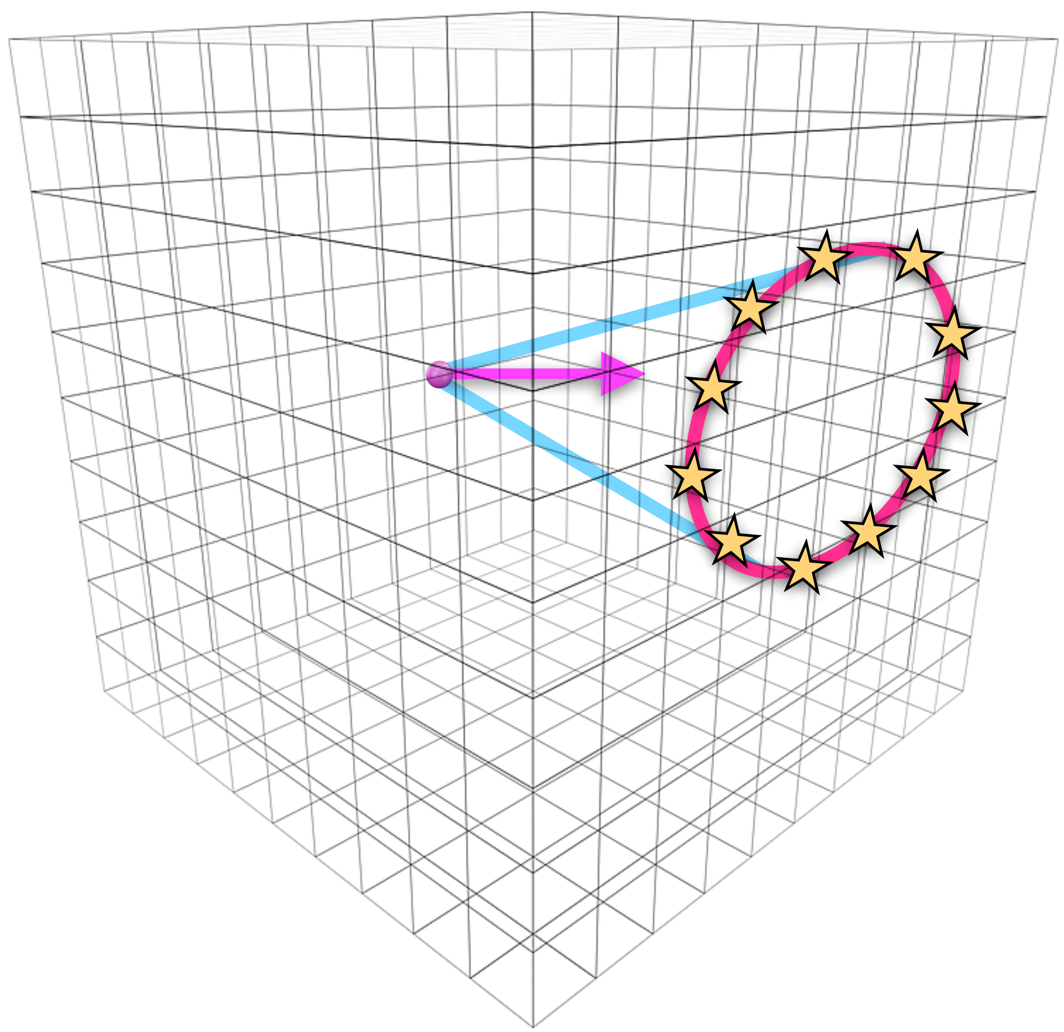


Uniformly sample photons in a cone around track direction (~Cherenkov like emission). Assume fixed angle (but this could be additional degree of freedom).

Propagate N_{phot} photons (assumptions discussed later) until a photosensor (surface) is reached.
Step-by-step voxel propagation thanks to Taichi capabilities for sparse computation.

N_{phot} is an additional degree of freedom.

5 DETECTOR RESPONSE



Count how many photons arrive to each photosensor.

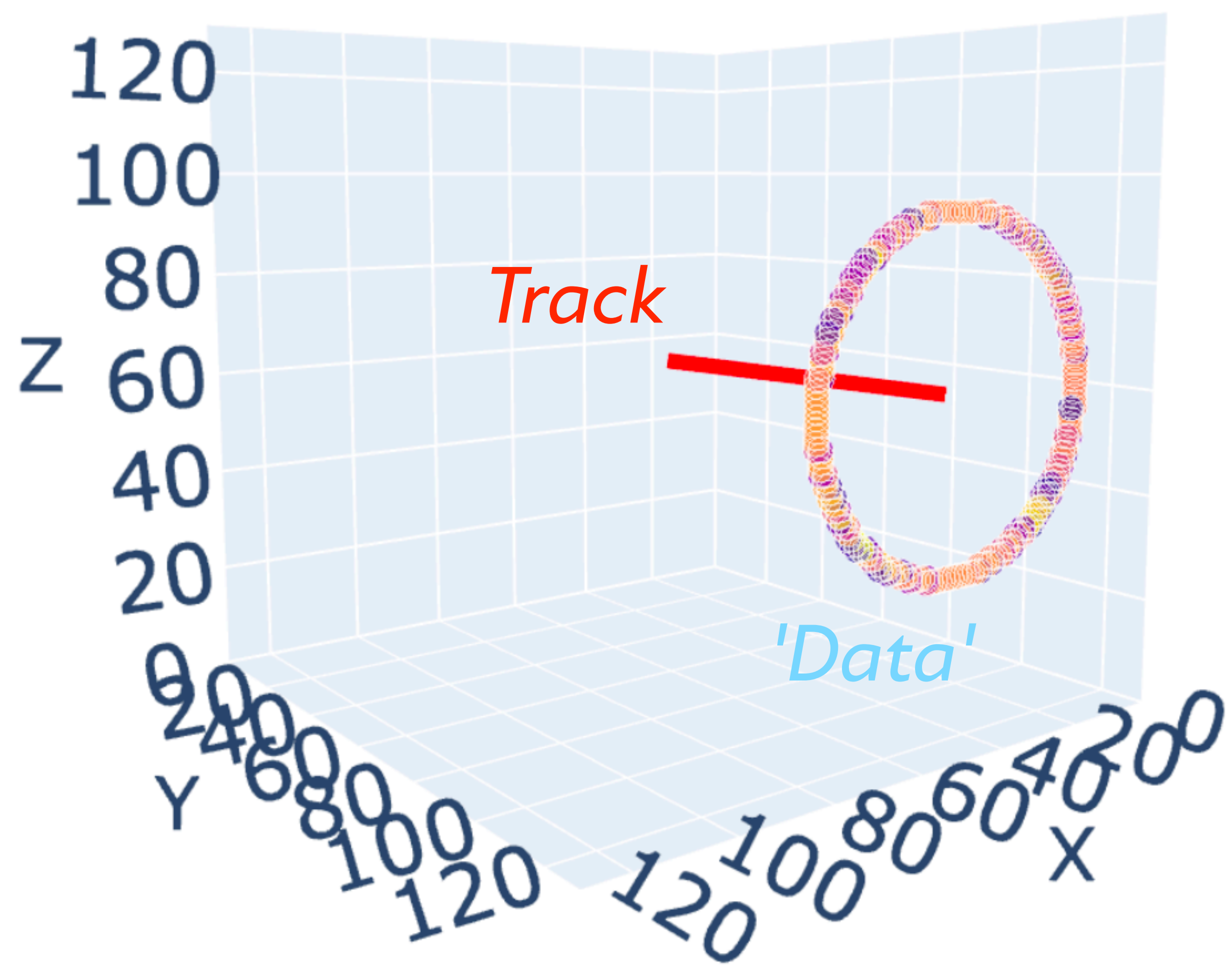
Assume that each photon contribute $e^{-L/\lambda_{att}}$ counts. With L being the photon path length and λ_{att} the attenuation length, which is an additional degree of freedom.

1. *Select **true** parameters:*

   ($\vec{x}$, $\vec{v}$, λ_{att} and N_{phot} *8 params in total)*

2. *Create one event display by running the forward.*

   Data ≡ Retain only the number of photons in every sensor voxel.



OPTIMIZATION

Repeat until convergence:

1. *Select **reco** parameters:*

   ($\vec{x}$, $\vec{v}$, N and N_{phot} 8 params in total)

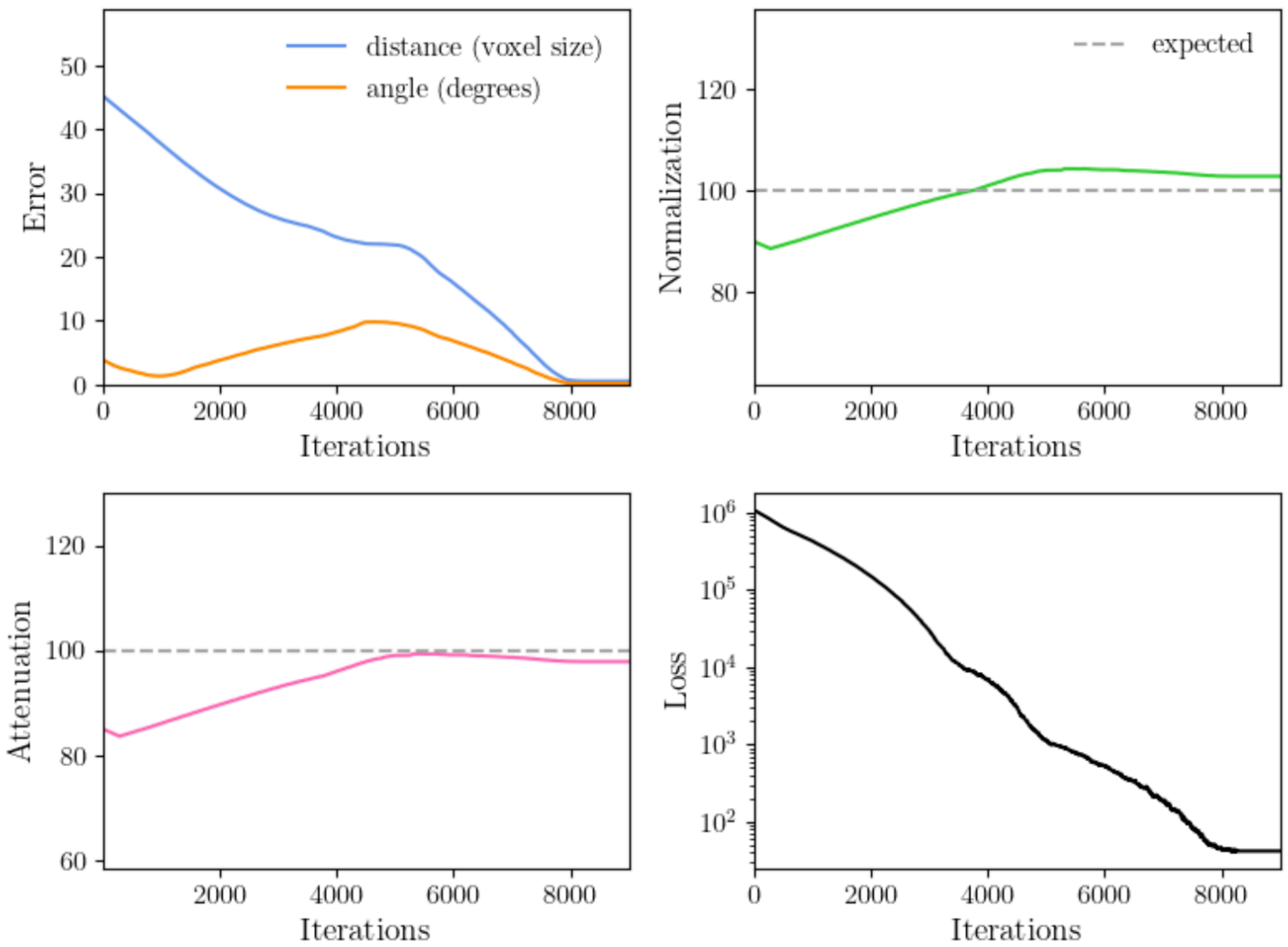
2. *Calculate photons final position by running forward for the 8 reco parameters*

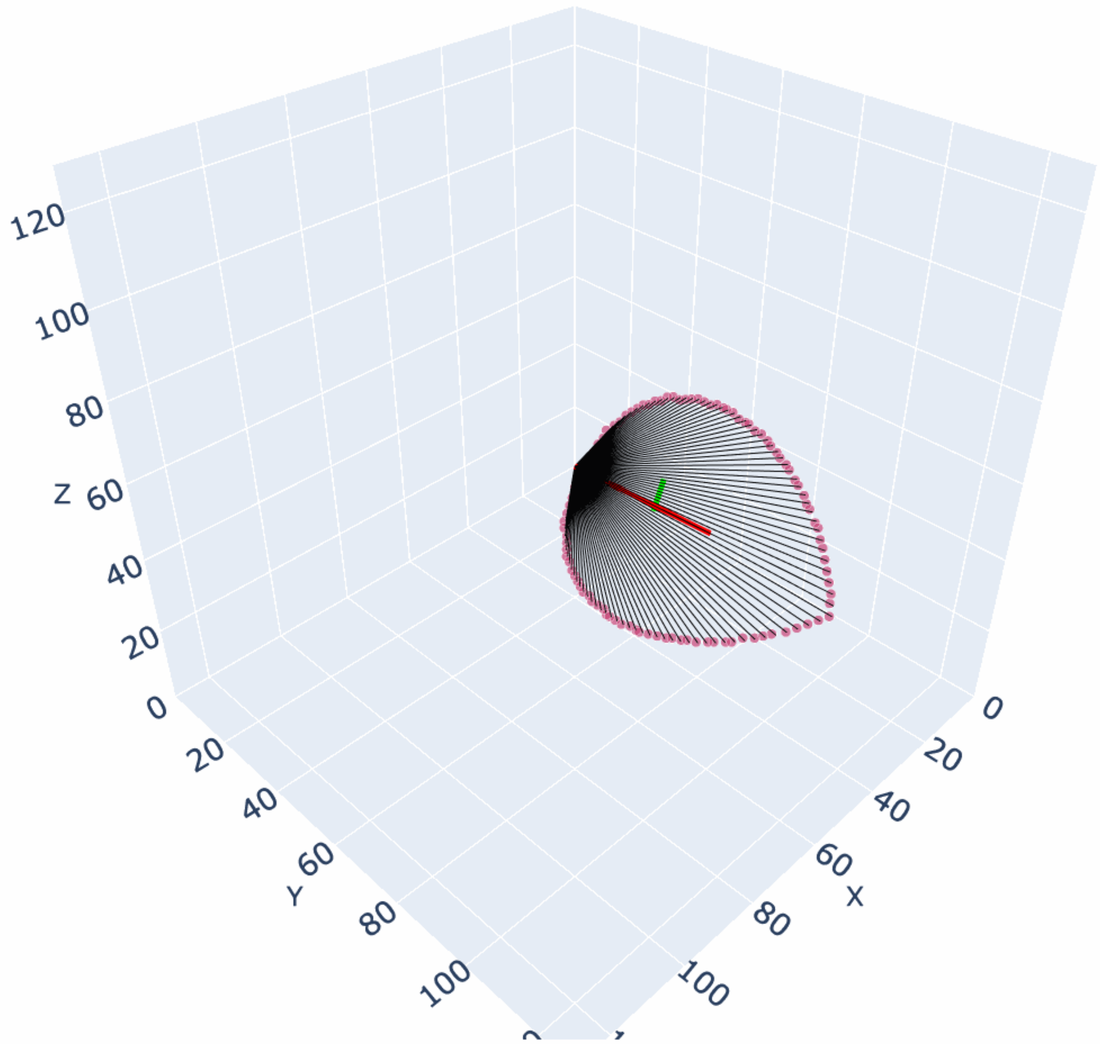
3. *Calculate loss.*

4. *Use gradients to update the reco parameters.*

RESULTS

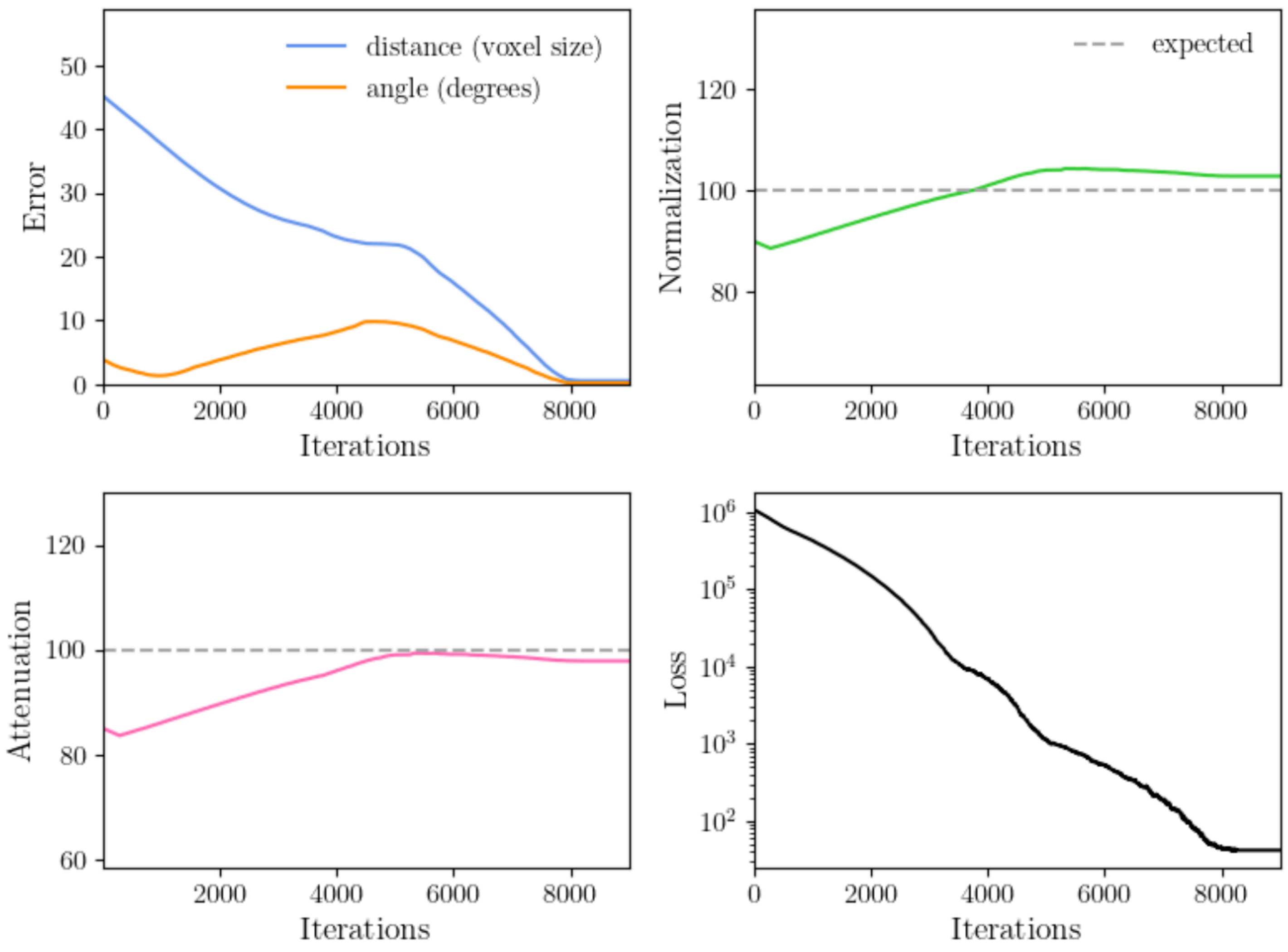
We can simultaneously minimize calibration-like quantities (e.g. the light attenuation constant) and reconstruction-like quantities (track parameters) via gradient descent.

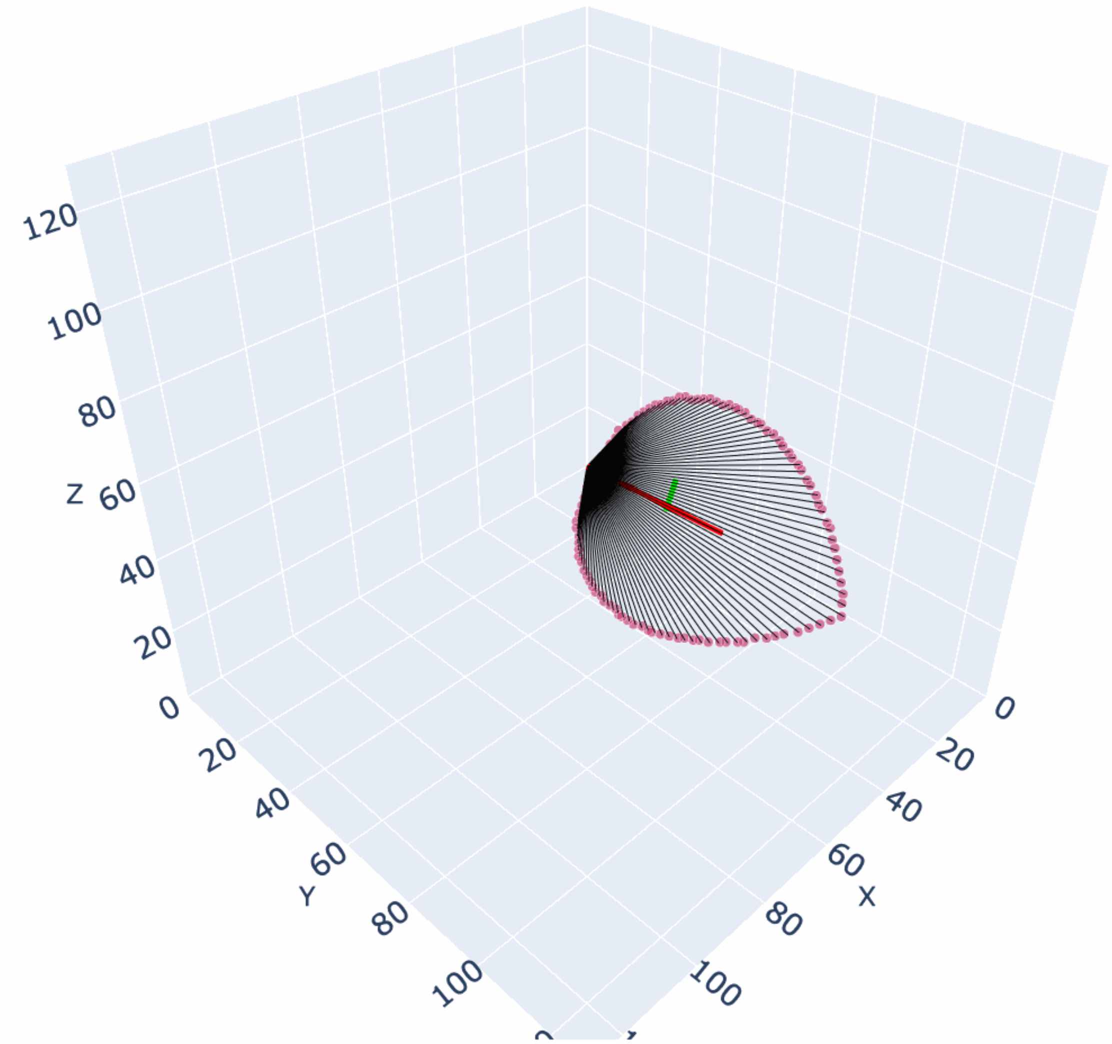




RESULTS

We can simultaneously minimize calibration-like quantities (e.g. the light attenuation constant) and reconstruction-like quantities (track parameters) via gradient descent.





Current results support viability of further exploring differentiable physics models as a future solution to integrate calibration & reconstruction in a single framework.

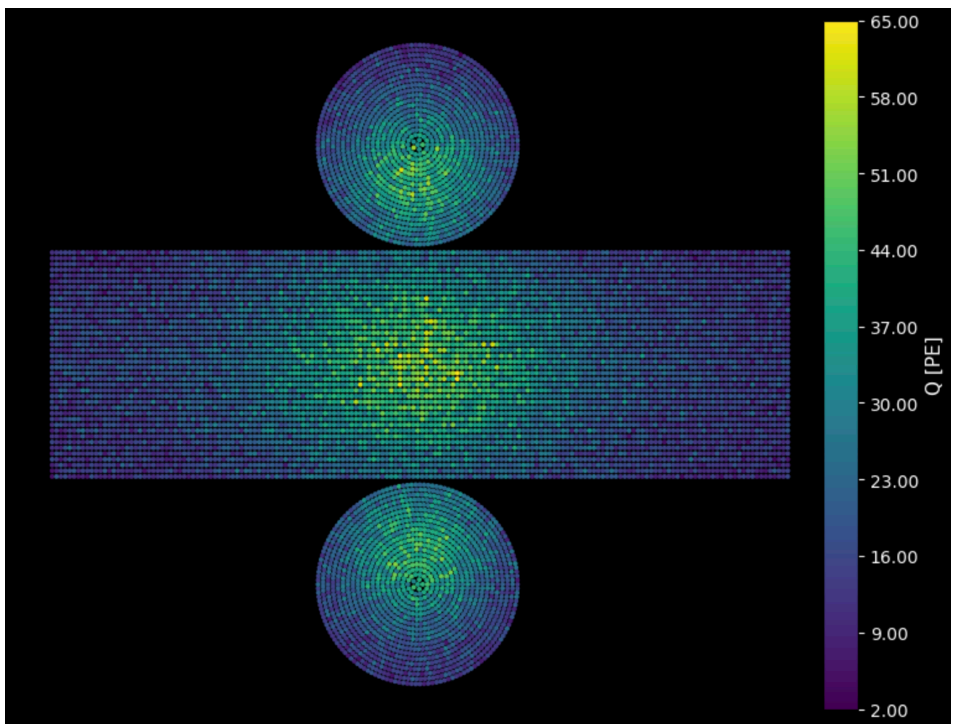
Major challenge: In the exploration so far we have not included (or have bypassed) dealing with processes that are inherently stochastic (e.g. scattering, reflections…).

Taichi does not yet support auto differentiation for sampling operations.

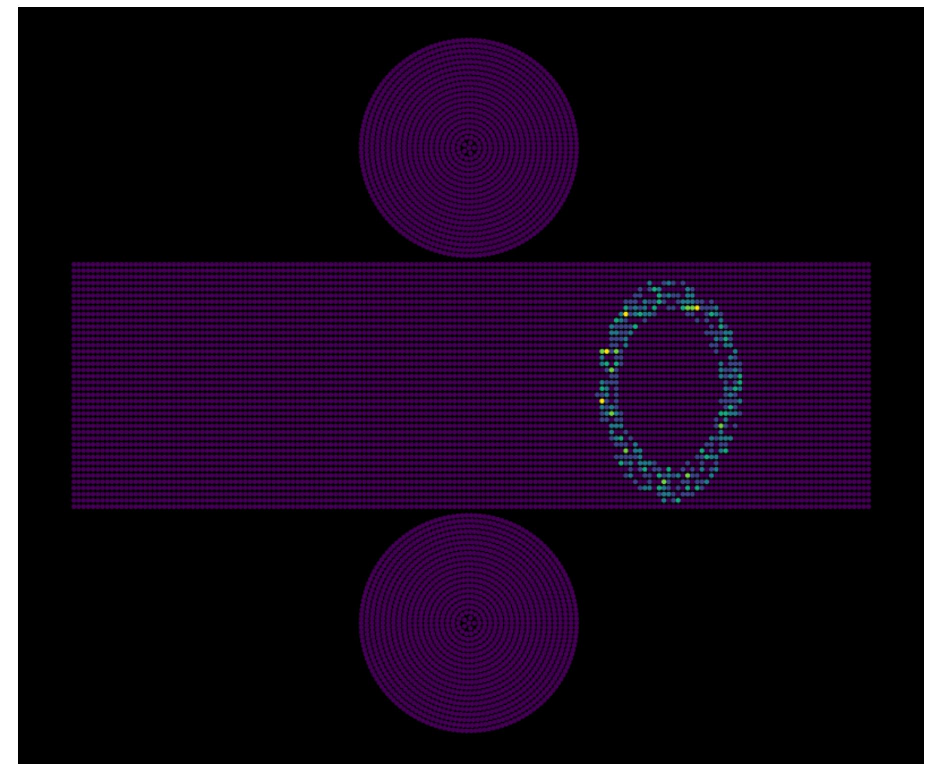
Let's start from scratch using

Not truly from scratch...

Developed non-differentiable simple Cherenkov simulator for other projects in CIDER-ML using numpy. *(see J.Xia's Talk!)*



Example of a diffuse event (isotropic light with common origin)

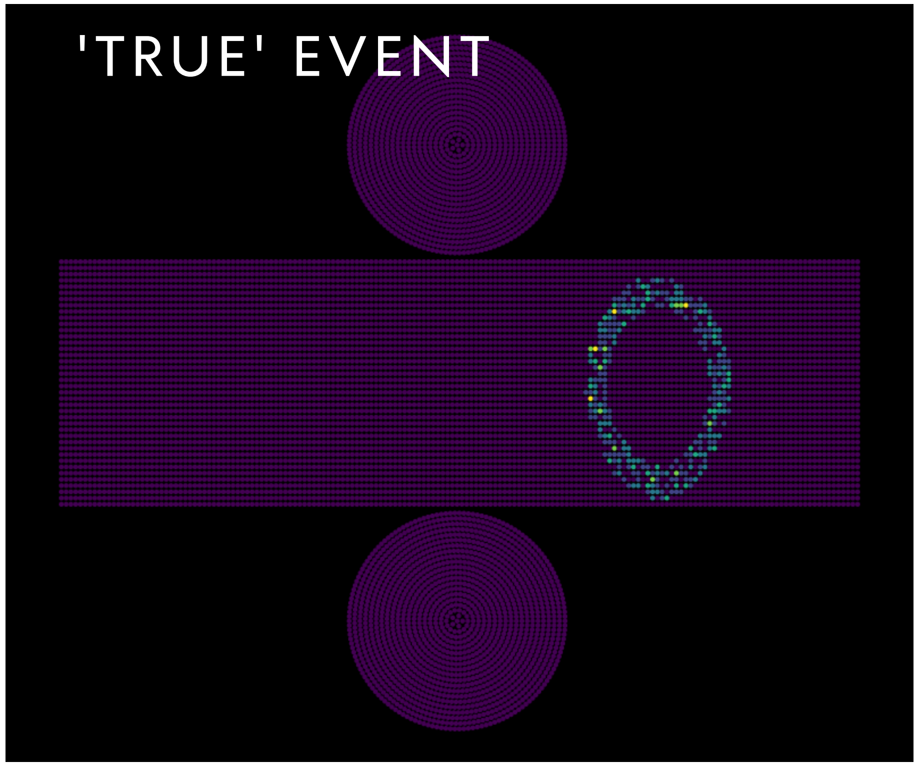


Example of a Cherenkov-like event

Let's translate numpy to JAX, and use it for our purposes.

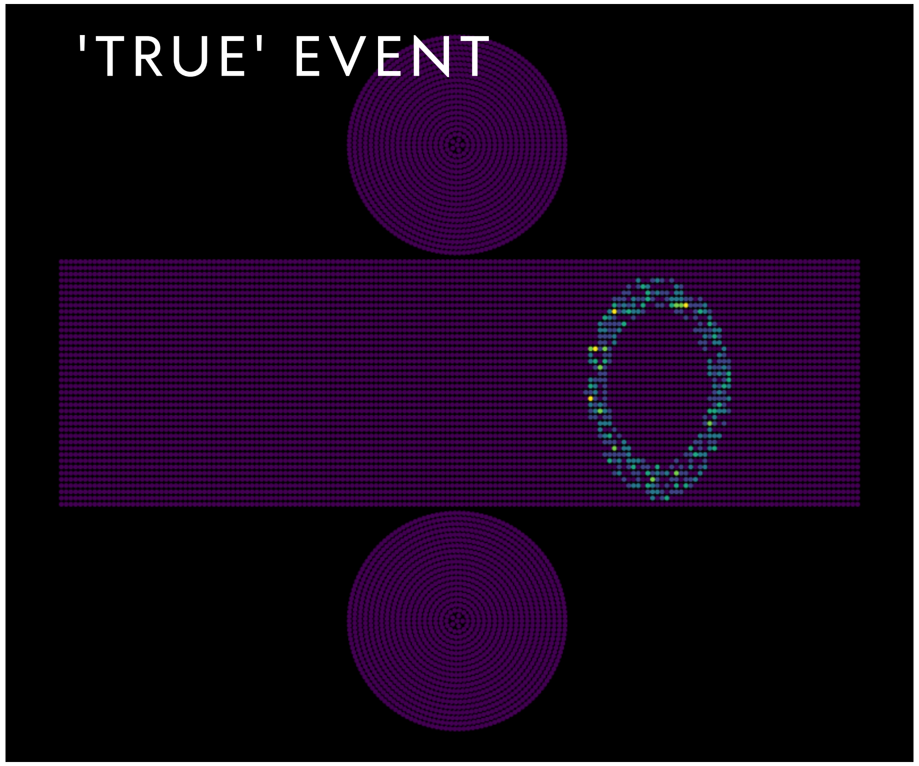
RESULTS

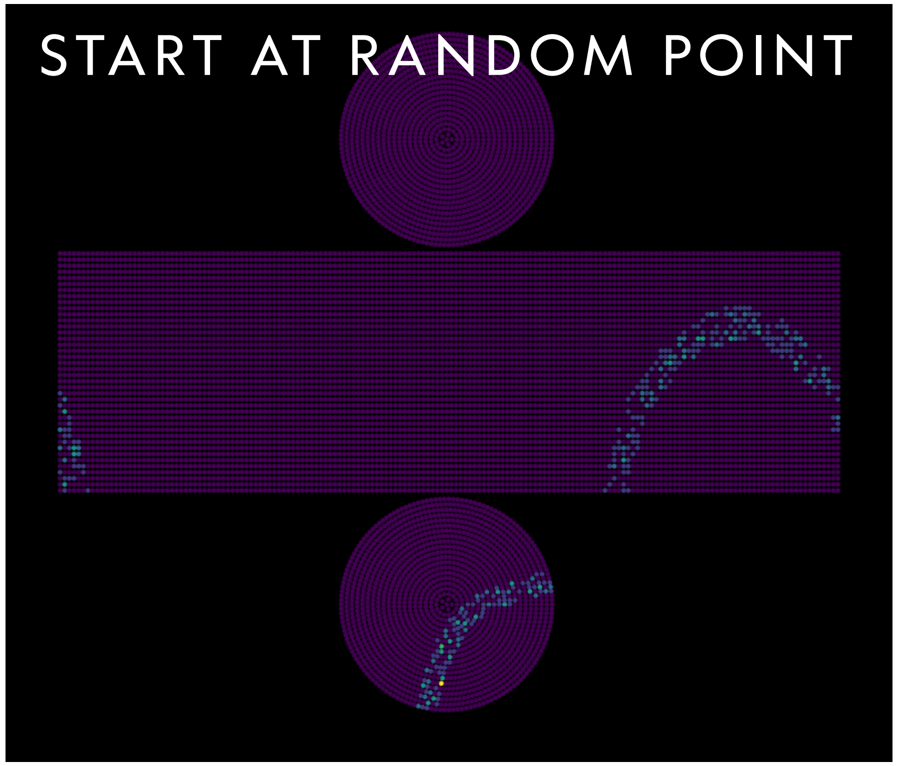
Select ***true*** *parameters:* $(\vec{x}, \vec{v}, \theta_{CH})$



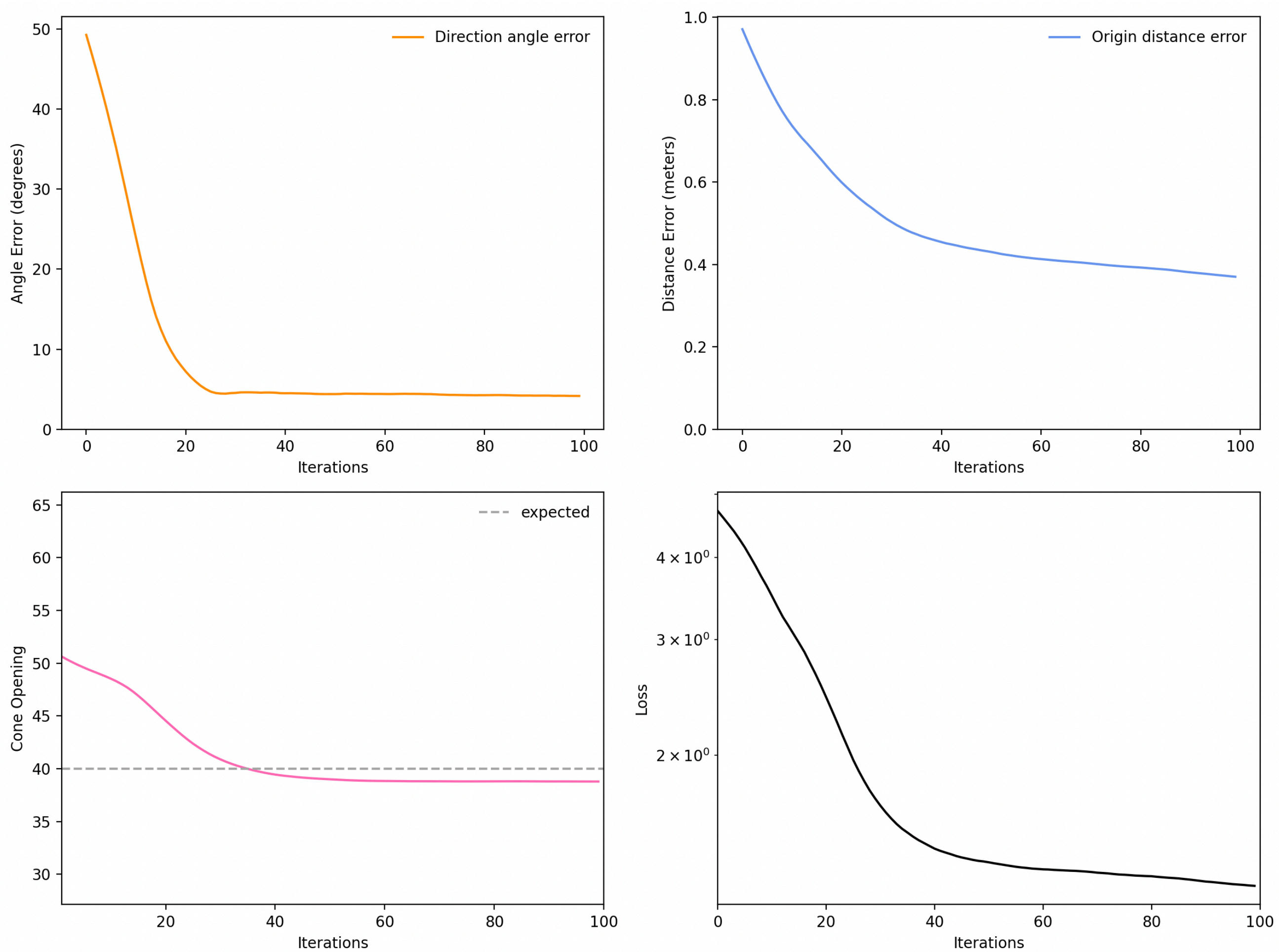
RESULTS

Select ***true*** *parameters:* $(\vec{x}, \vec{v}, \theta_{CH})$

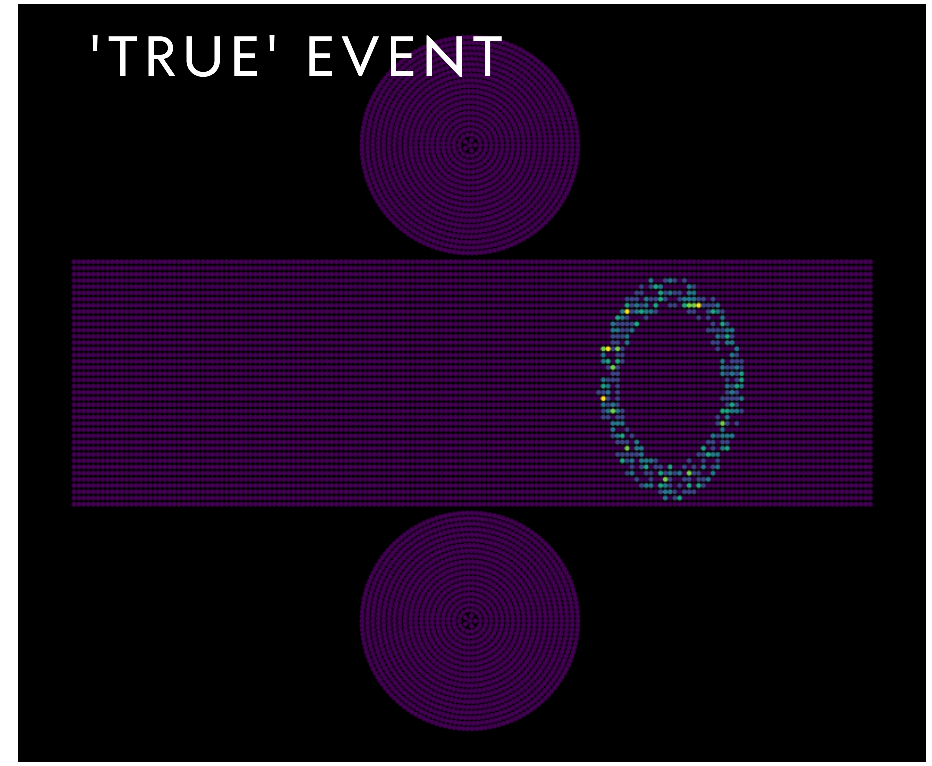


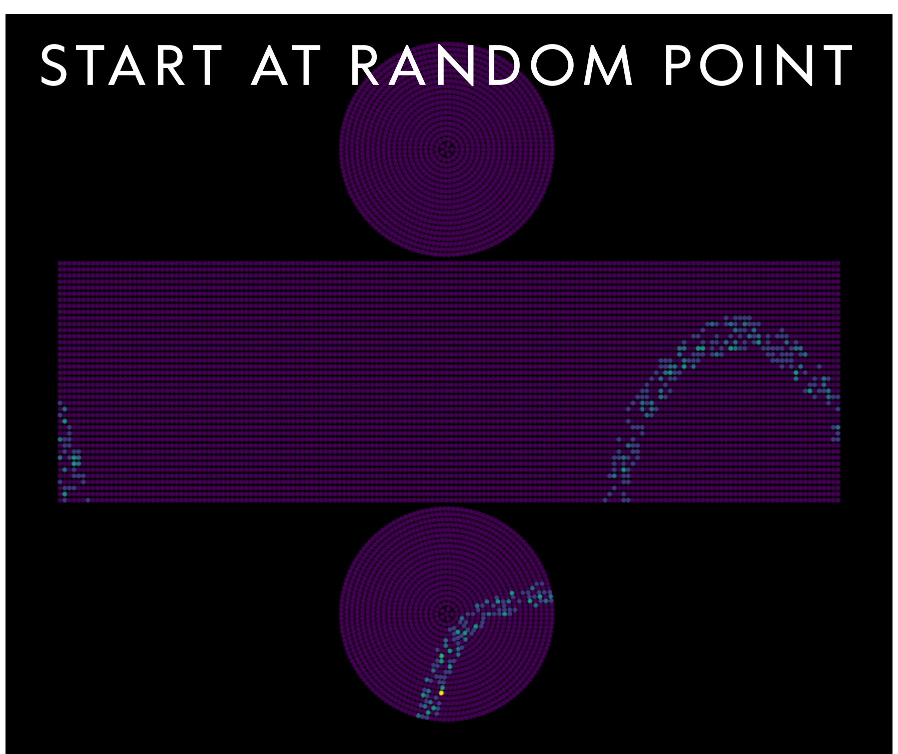


RESULTS

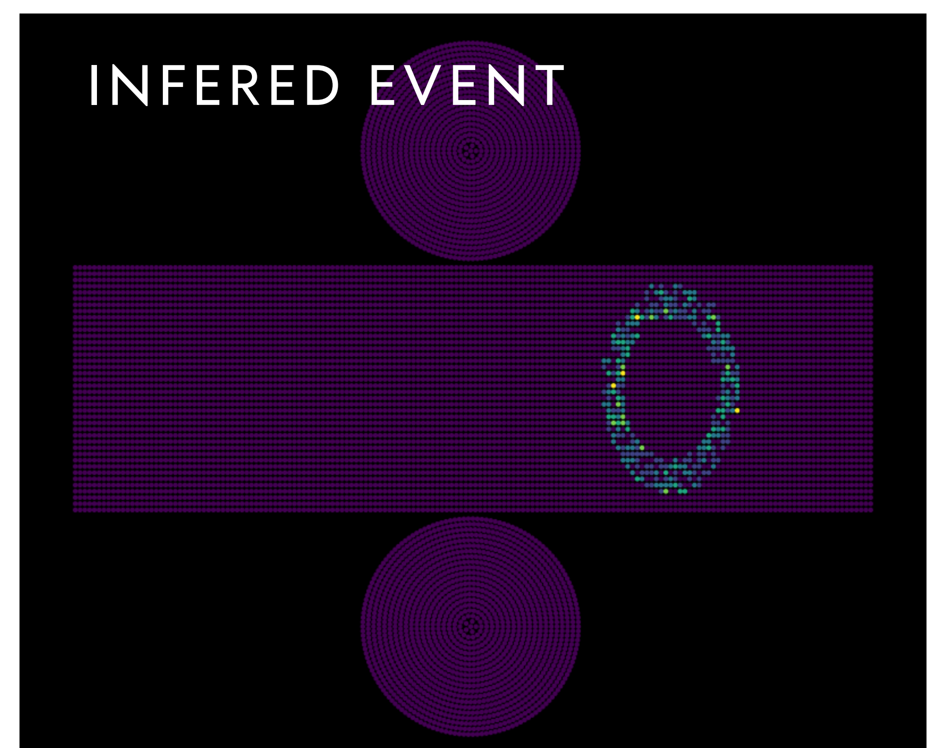


Select **true** parameters: $(\vec{x}, \vec{v}, \theta_{CH})$





Optimize



NEXT STEPS & CONCLUSIONS

We have implemented two independent differentiable models of a toy Cherenkov detector.

https://github.com/CIDeR-ML/simpleCherenkovSim/tree/autodiff_test
https://github.com/CIDeR-ML/taichi-cher-sim/tree/main

Next step is to add up stochastic processes and optimize related physics parameters (scattering, reflections).

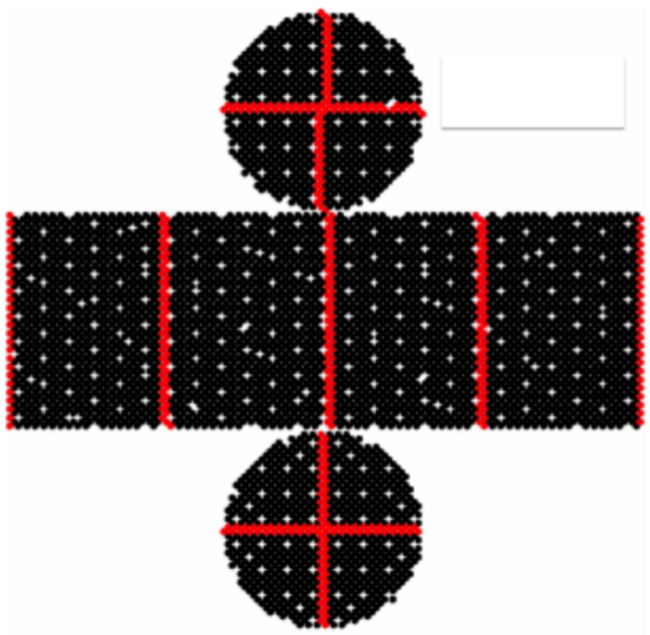
Are we ready to build end-to-end analytical models? No.
But do we want to keep working in the same way in the next 20 years?

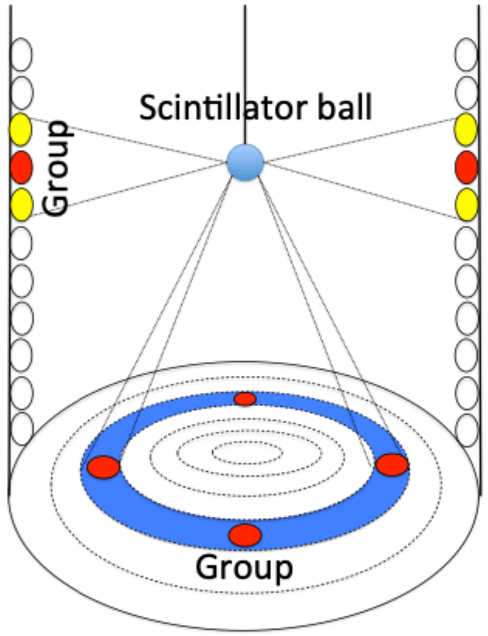
Walking steps in the direction of alternative solutions can help us to understand their limitations & advantages. Differentiable detector simulations have the potential to redefine old-existing paradigms in HEP-ex.

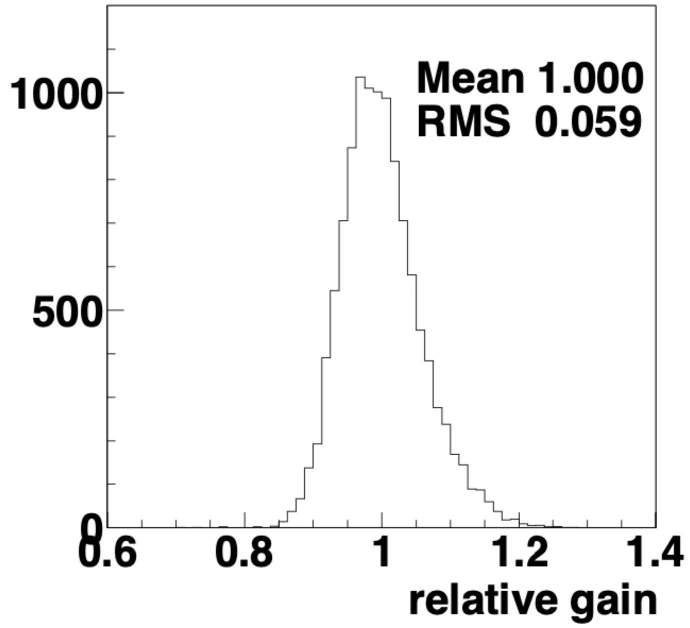
Back Up

SUPER-K CALIBRATION AS AN EXAMPLE arXiv:1307.0162

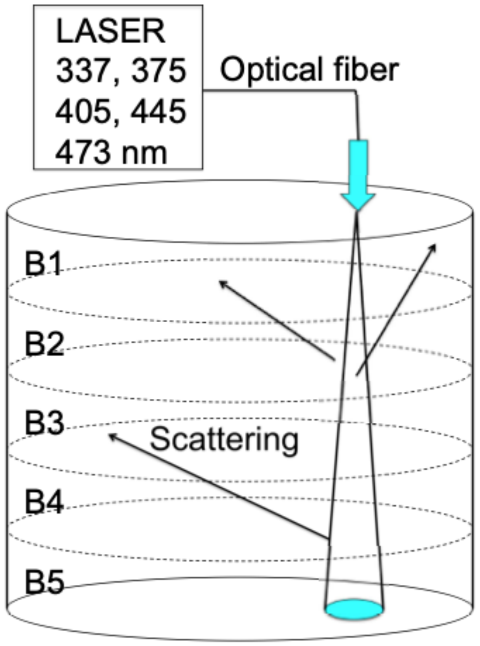
CHANNEL BY CHANNEL INFORMATION





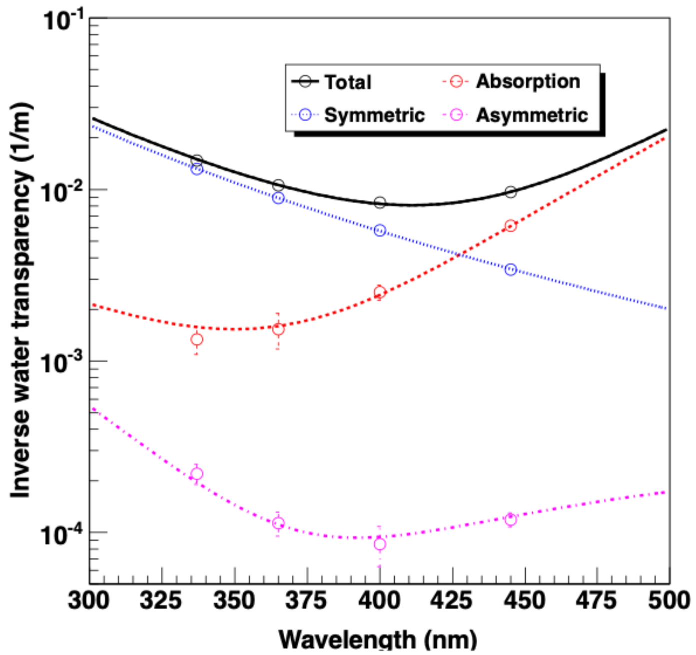


DETECTOR LEVEL INFORMATION



$$I(\lambda) = I_0(\lambda) e^{-\frac{l}{L(\lambda)}}$$

$$L(\lambda) = \frac{1}{\alpha_{abs}(\lambda) + \alpha_{sym}(\lambda) + \alpha_{asy}(\lambda)}$$



SUPER-K RECONSTRUCTION AS AN EXAMPLE *developed from MiniBooNE arXiv:0902.2222*

$$L(\mathbf{x}) = \prod_{j}^{unhit} P_j\left(unhit|\mathbf{x}\right) \prod_{i}^{hit} P_i\left(hit|\mathbf{x}\right) f_q\left(q_i|\mathbf{x}\right) f_t\left(t_i|\mathbf{x}\right)$$

- L: Likelihood to maximise
- $\mathbf{x}$: Candidate event hypothesis
- $P_j\left(unhit|\mathbf{x}\right)$: Probability of no hit at PMT
- $P_i\left(hit|\mathbf{x}\right)$: Probability of hit at PMT
- $f_q\left(q_i|\mathbf{x}\right)$: Hit charge probability density
- $f_t\left(t_i|\mathbf{x}\right)$: Hit time probability density

- For each particle type option $(\mu, e, \pi...)$ maximize over track params (kinematics).

Repeat until convergence:

1. *Select **reco** parameters:*

   ($\vec{x}$, $\vec{v}$, N and N_{phot} 8 params in total)

2. *Calculate photons final position by running forward for the 8 reco parameters*

3. *Calculate loss.*

   Example:

$$Loss = \sum_{i}^{N\,photons} distance\_to\_closest\_PMT(p_i) \times PMT\_loss\_term$$

$$PMT\_loss\_term = \sum_{j}^{fired\ PMTs} time\_distance\_to\_closest\_photon_j$$

$$time\_distance\_to\_closest\_photon_j = abs(time\_PMT_j - time\_closest\_photon_j)$$

In the future we plan to have an optimal transport inspired loss using Wasserstein distance between predicted & data event.

4. *Run backward & update the reco parameters.*