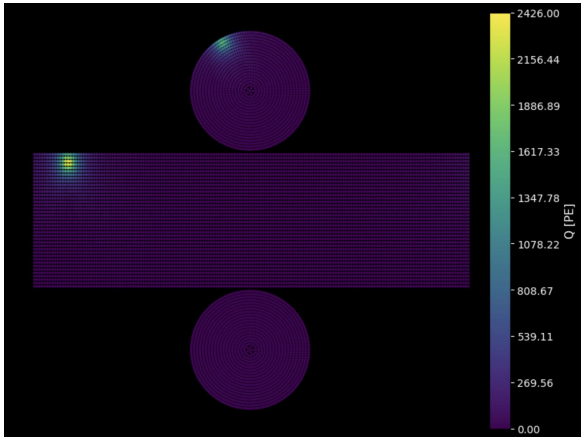
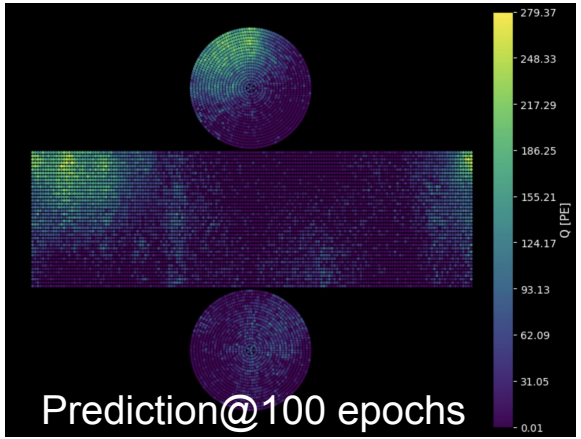
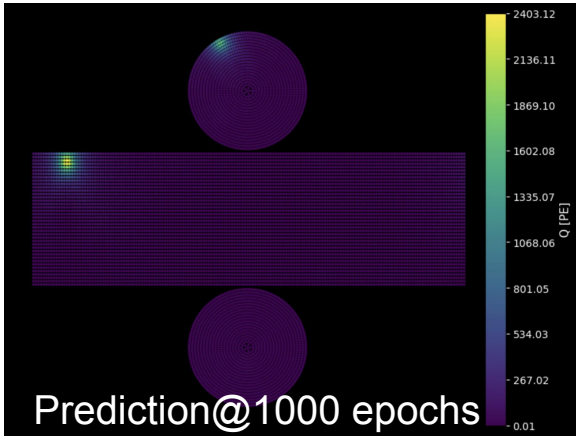
Status report and outlook

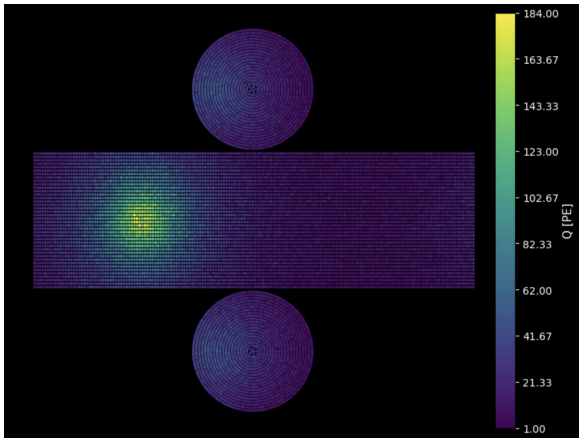
Junjie Xia
March 21 2024, CIDeR-ML

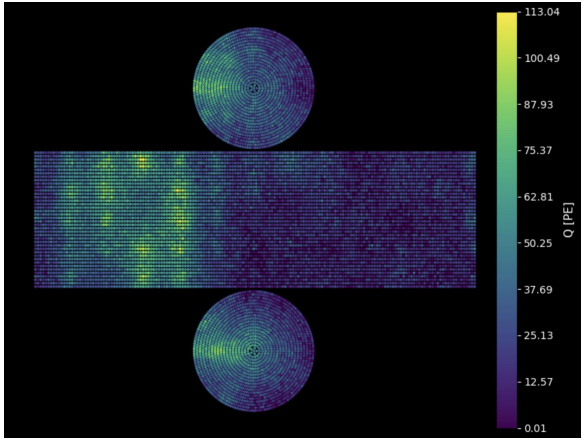
Reminder - Training multiple position DB

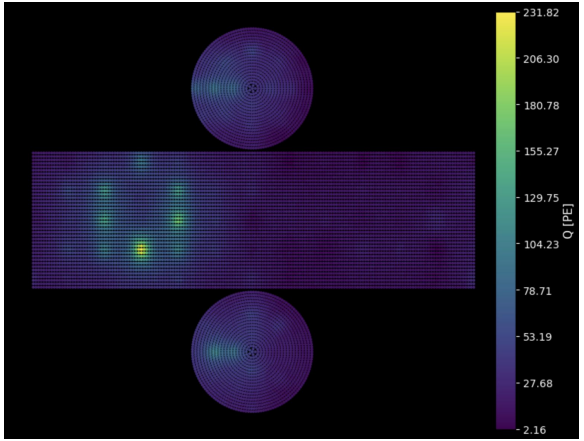






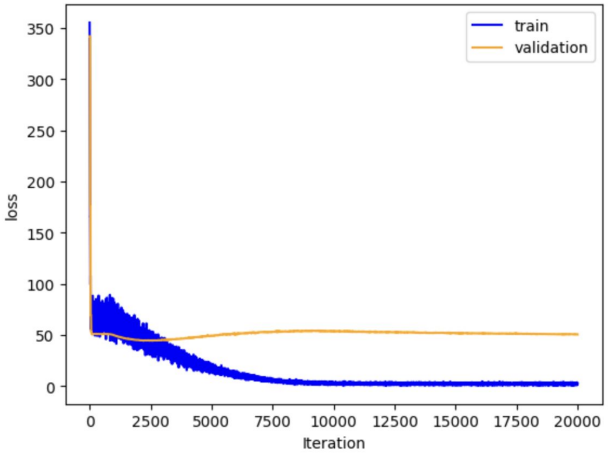




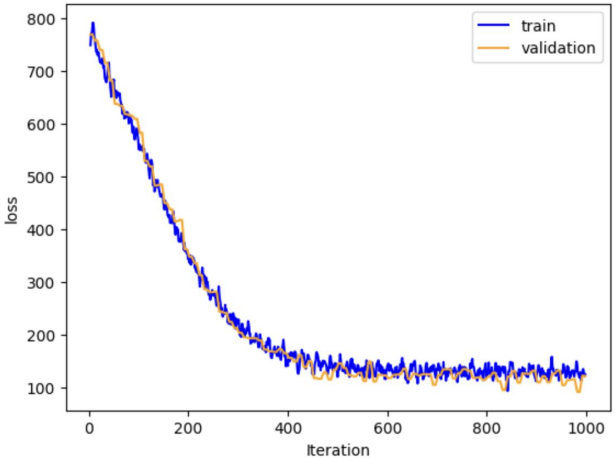


Training multiple position DB

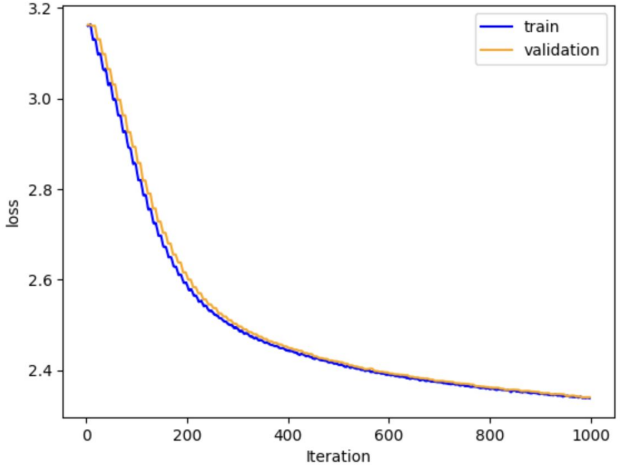
Old non-accumulated MSE loss training with direction input



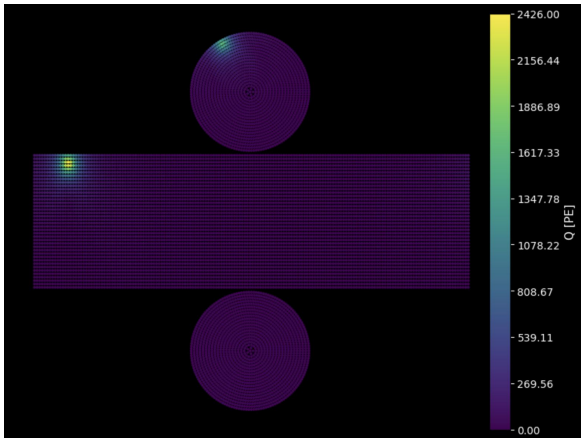
New accumulated MSE loss training without direction input

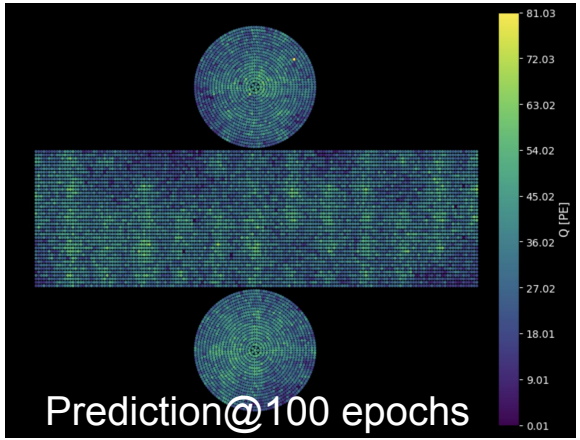


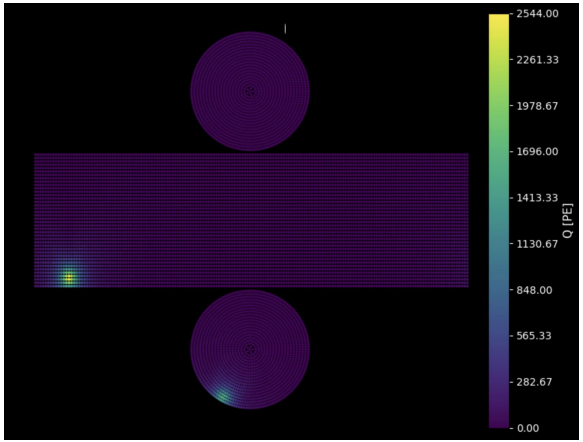
New accumulated Pois loss training without direction input

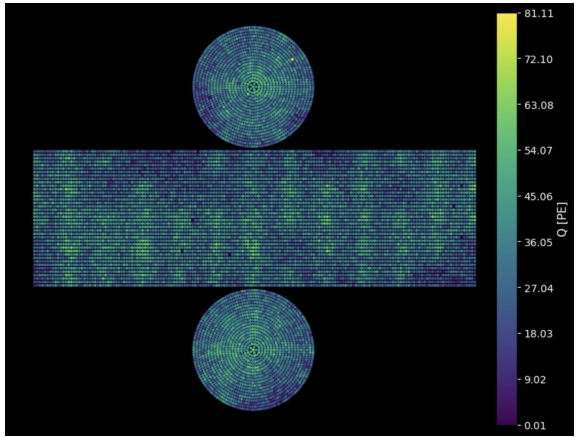


Training multiple position DB - accumulated MSE loss w/o dir

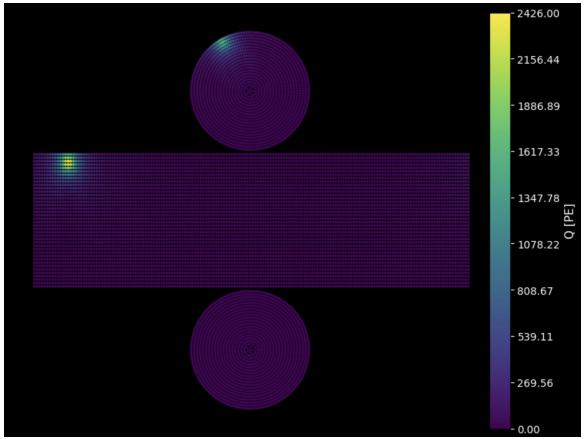


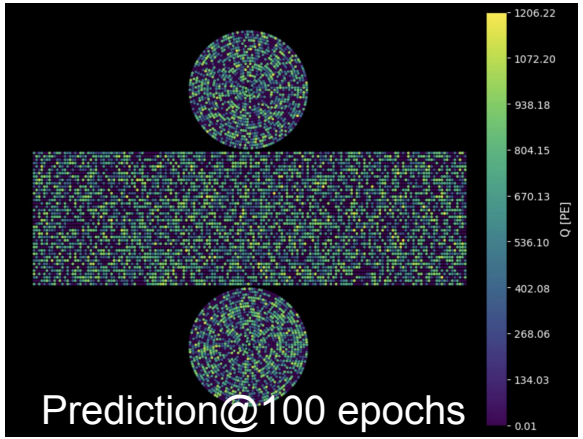


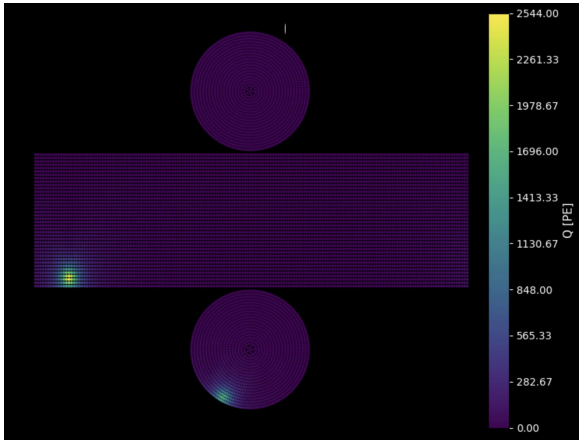


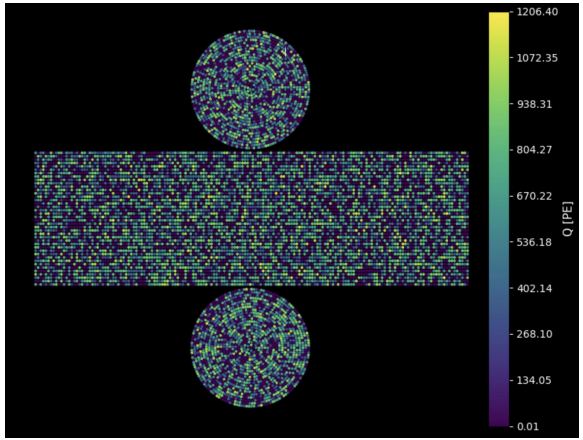


Training multiple position DB - accumulated Poisson loss w/o dir

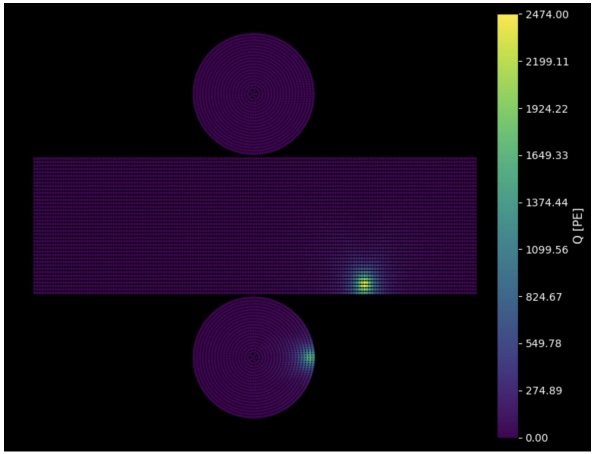


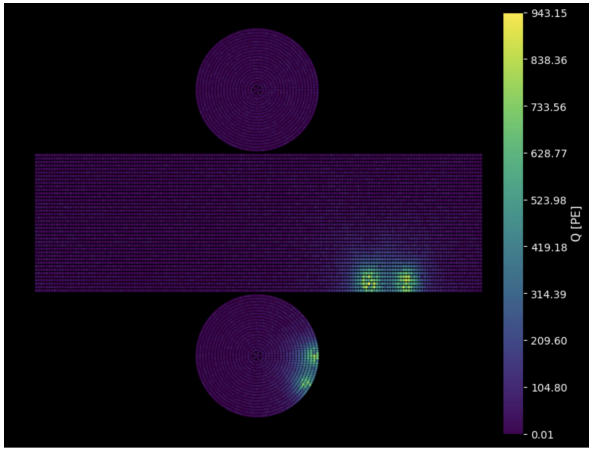


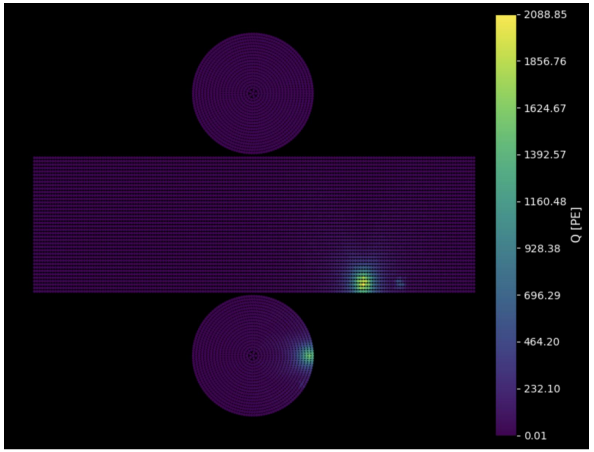


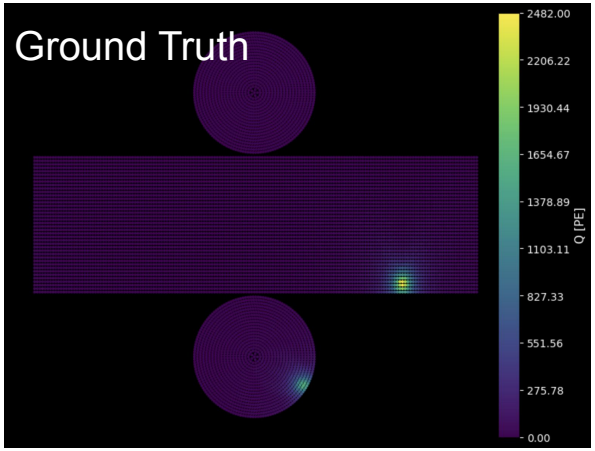


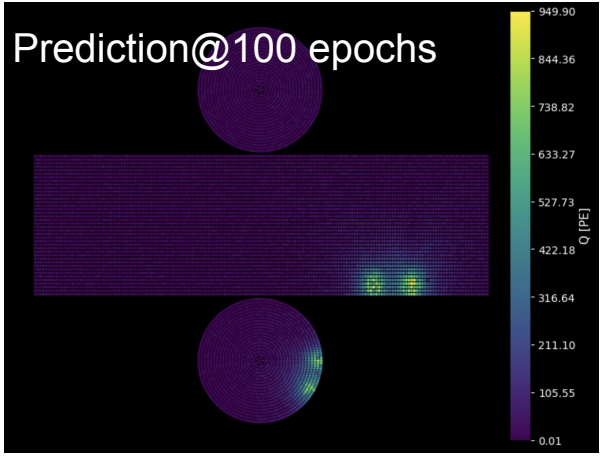
Training 2-position SimpleSim DB - MSE loss w/o dir

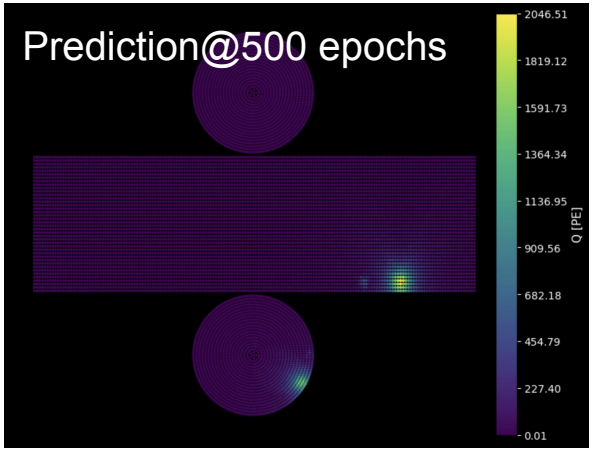




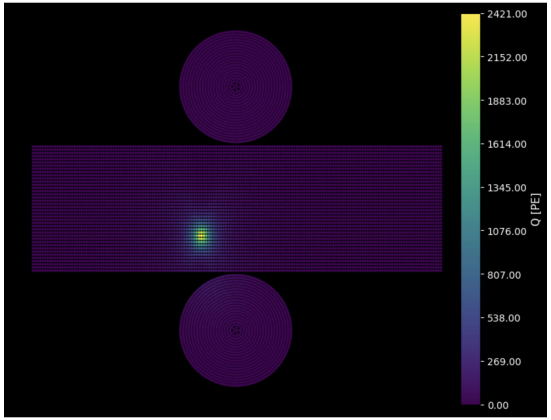


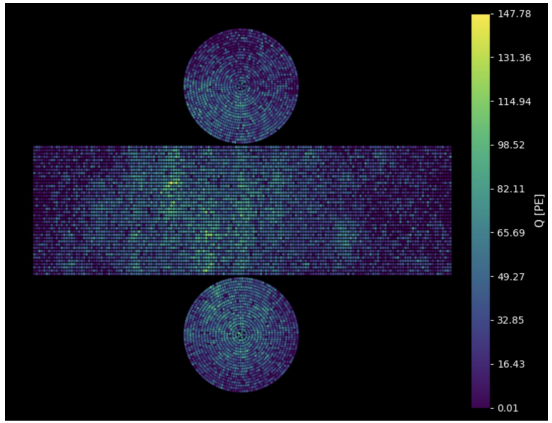


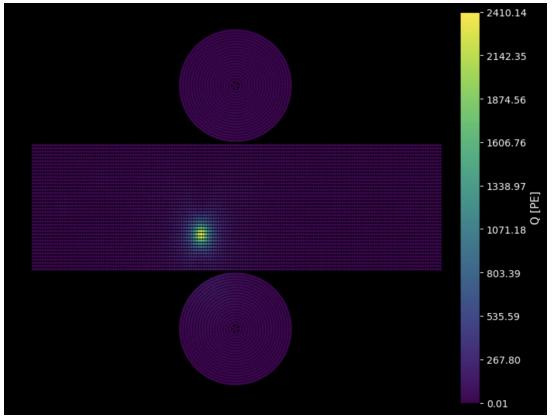


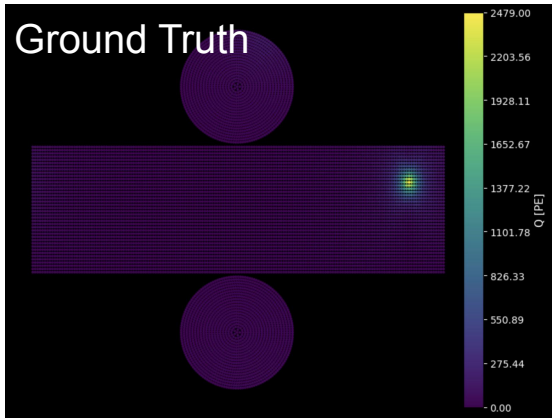


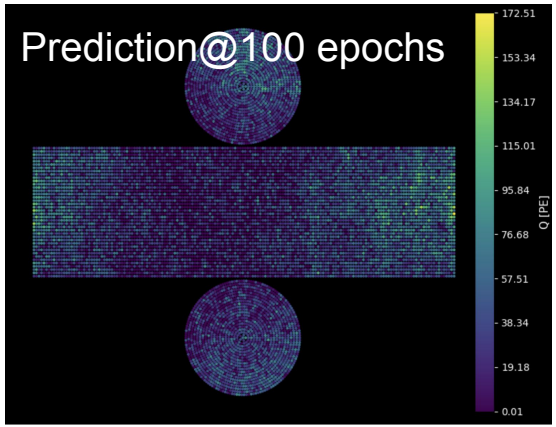
Training 125-position SimpleSim DB - MSE loss w/o dir, 4x larger LR

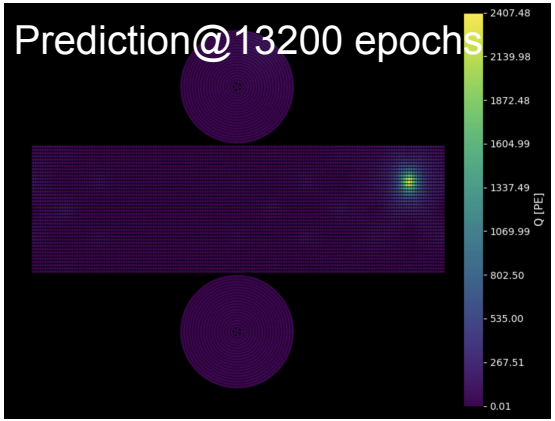




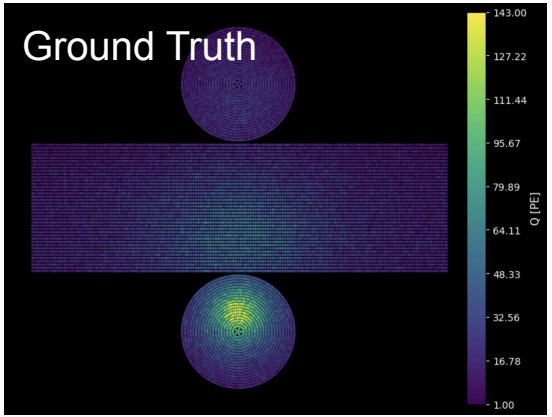


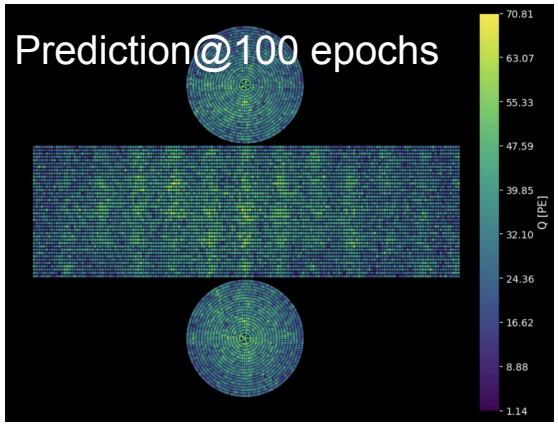


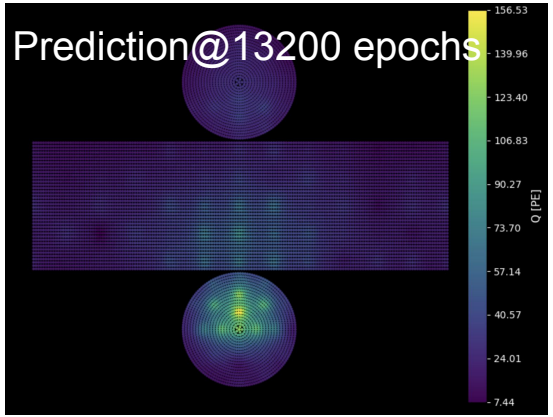


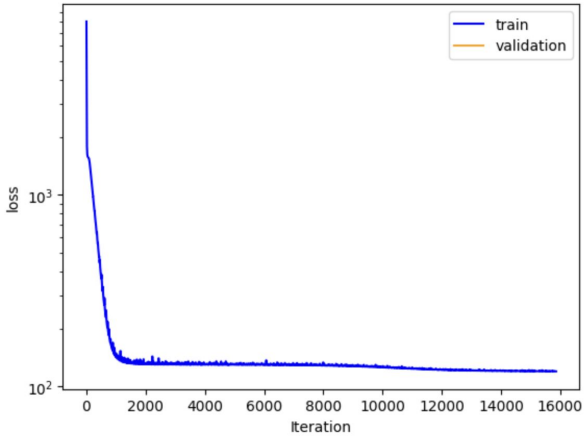


Training 125-position SimpleSim DB - MSE loss w/o dir, 4x larger LR









Observations and thoughts

- After >15k epochs SIREN still cannot predict the “diffuse” PMT hits very well - although the prediction does have the highest intensity PMT around the correct position and the “neighboring hits” are dimmer

- The loss is still converging

    - Check the distance difference between the diffuser ball locations for a “hot-spot” hit and a “diffused” hit
    - Quantify the performance as a function of trained epochs

- Is this how you accumulate gradients for one epoch?

```
batch.forward()
loss.backward()
If one epoch is finished:
    Model.param.grad /= len(epoch)
    optimizer.step()
```

Observations and thoughts

- But anyway I think the performance with SimpleSim DB is mostly well understood and the remaining issues can be studied with real physics MC
- Moved on to check the WCSim isotropic MC and photon shotgun (provided by Patrick)
- Can load the isotropic MC with the current dataloader, will perform more checks
- Photon shotgun MC needs some improvements (getting 0 for all PMT charges)
- Check the existing photon shotgun files on s3df as well

Memory usage observation - batch size 5, no direction input

1st iteration

Before running model torch.cuda.memory_allocated: 5.014648MB
Before running model torch.cuda.memory_reserved: 22.000000MB
==========
After running model torch.cuda.memory_allocated: 154.167969MB
After running model torch.cuda.memory_reserved: **5936.000000MB**

2nd iteration

Before running model torch.cuda.memory_allocated: 13.269043MB
Before running model torch.cuda.memory_reserved: **13528.000000MB**
==========
After running model torch.cuda.memory_allocated: 15.174316MB
After running model torch.cuda.memory_reserved: 13532.000000MB

3rd iteration

Before running model torch.cuda.memory_allocated: 13.648926MB
Before running model torch.cuda.memory_reserved: 13532.000000MB
==========
After running model torch.cuda.memory_allocated: 15.174316MB
After running model torch.cuda.memory_reserved: 13532.000000MB

6GB gradients?
X2 forward and backward?

Memory usage observation - batch size 5, no direction input

Iter 1 - Before running model torch.cuda.memory_allocated: 5.02MB
Iter 1 - Before running model torch.cuda.memory_reserved: 22.00MB
Using position input only.
Iter 1 - After running model torch.cuda.memory_allocated: 154.36MB
Iter 1 - After running model torch.cuda.memory_reserved: 5936.00MB
Iter 2 - Before running model torch.cuda.memory_reserved: 5940.00MB
Iter 2 - After running model torch.cuda.memory_allocated: 303.33MB
Iter 2 - After running model torch.cuda.memory_reserved: 7838.00MB
Iter 3 - Before running model torch.cuda.memory_allocated: 302.19MB
Iter 3 - Before running model torch.cuda.memory_reserved: 7838.00MB
Iter 3 - After running model torch.cuda.memory_allocated: 451.34MB
Iter 3 - After running model torch.cuda.memory_reserved: **7838.00MB**
…

Accumulate loss for 1 epoch
and then run .backward()

Iter 20 - Before running model torch.cuda.memory_allocated: 2819.28MB
Iter 20 - Before running model torch.cuda.memory_reserved: 9824.00MB
Iter 20 - After running model torch.cuda.memory_allocated: 2968.44MB
Iter 20 - After running model torch.cuda.memory_reserved: 9824.00MB
Epoch 0 : 100%|[redacted]| 20/20
Epoch 1 : 0%| | 0/20 [00:00<?, ?batch/s]
Iter 21 - Before running model torch.cuda.memory_allocated: 22.53MB
Iter 21 - Before running model torch.cuda.memory_reserved: **17420.00MB**
Iter 21 - After running model torch.cuda.memory_allocated: 169.40MB
Iter 21 - After running model torch.cuda.memory_reserved: 17420.00MB
Iter 22 - Before running model torch.cuda.memory_allocated: 168.26MB
Iter 22 - Before running model torch.cuda.memory_reserved: 17420.00MB
Iter 22 - After running model torch.cuda.memory_allocated: 319.34MB
Iter 22 - After running model torch.cuda.memory_reserved: 17420.00MB
Iter 23 - Before running model torch.cuda.memory_allocated: 318.19MB
Iter 23 - Before running model torch.cuda.memory_reserved: 17420.00MB
Iter 23 - After running model torch.cuda.memory_allocated: 467.35MB
Iter 23 - After running model torch.cuda.memory_reserved: 17420.00MB

Benchmark

GPU Memory Allocated	GPU Memory Reserved	SimpleSim DB Batch size	Direction input?
~150 MB	~ 7 GB	5	❌
~150 MB	~ 7 GB	5	✔
~300 MB	~ 14 GB	10	❌
~300 MB	~ 14 GB	10	✔

Seems like the model parameters are taking up most of the memory

This is a 64-neuron-5-hidden-layer SIREN model??

Memory usage observation - batch size 10, no direction

64*5

In siren - Before running model torch.cuda.memory_allocated: 11.50MB
In siren - Before running model torch.cuda.memory_reserved: 24.00MB
In siren - After running model torch.cuda.memory_allocated: 7892.66MB
In siren - After running model torch.cuda.memory_reserved: 7928.00MB

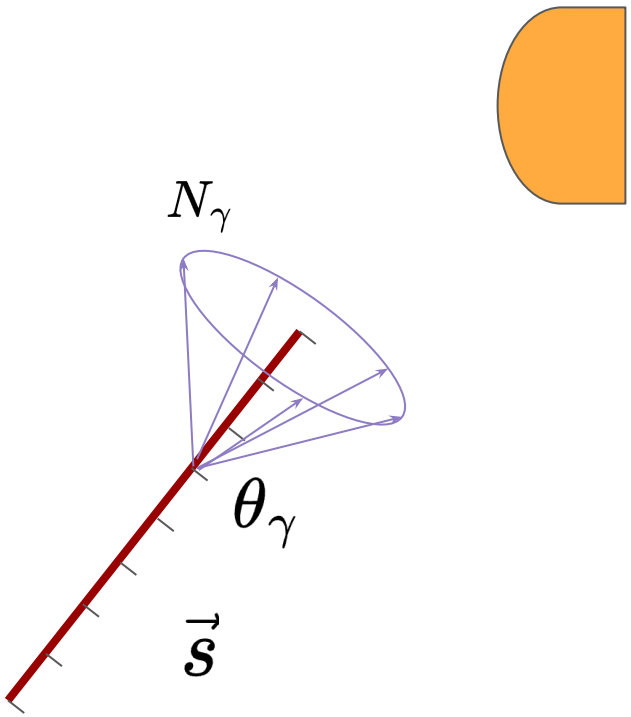
In siren - Before running model torch.cuda.memory_allocated: 311.07MB
In siren - Before running model torch.cuda.memory_reserved: 11788.00MB
In siren - After running model torch.cuda.memory_allocated: 7899.26MB
In siren - After running model torch.cuda.memory_reserved: 11790.00MB

32*5

In siren - Before running model torch.cuda.memory_allocated: 9.01MB
In siren - Before running model torch.cuda.memory_reserved: 24.00MB
In siren - After running model torch.cuda.memory_allocated: 7743.69MB
In siren - After running model torch.cuda.memory_reserved: 7770.00MB

In siren - Before running model torch.cuda.memory_allocated: 162.51MB
In siren - Before running model torch.cuda.memory_reserved: 11630.00MB
In siren - After running model torch.cuda.memory_allocated: 7750.70MB
In siren - After running model torch.cuda.memory_reserved: 11632.00MB

Plan of track reconstruction with OpticSiren

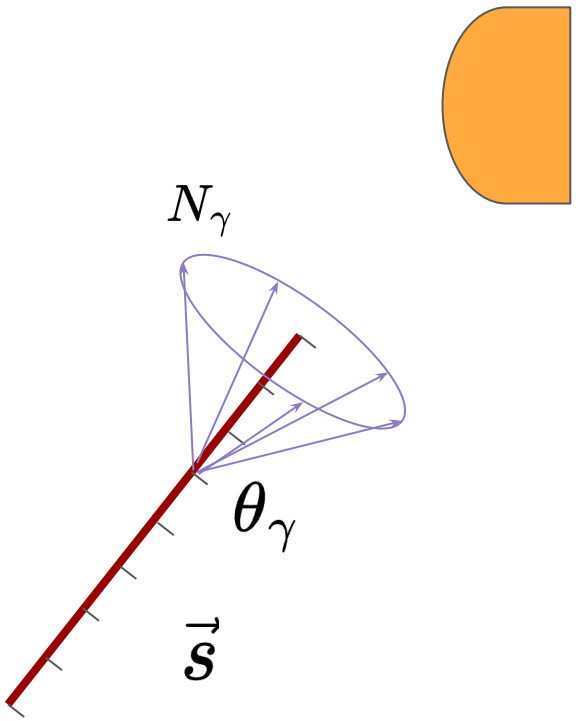


OpticSiren can predict (Q, T) responses in all PMTs given the photon emission position & direction, and the number of photons emitted.

Relativistic charged particles continuously emit Cherenkov radiation until they lose the energy to below their Cherenkov threshold.

The photons are emitted at an open angle related to the particle momentum and the material being traversed thru.

Plan of track reconstruction with OpticSiren



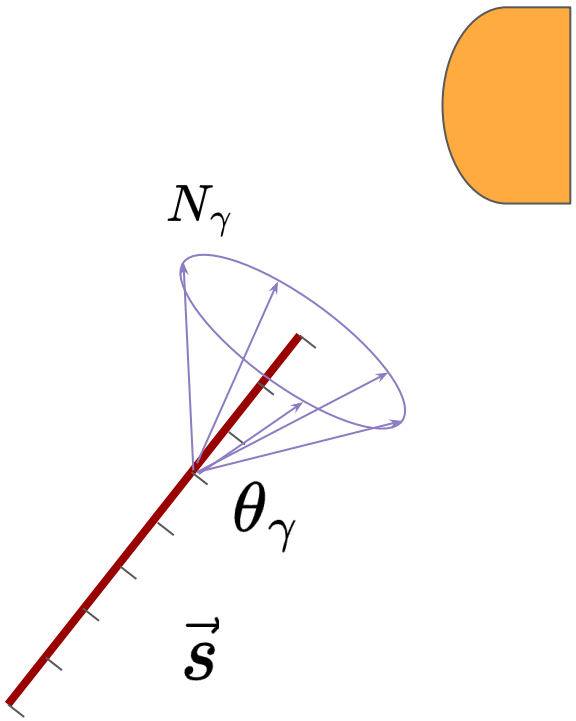
To reconstruct particle track using OpticSiren, break the particle trajectories into small segments and treat them as point sources.

At each point source, suggest N photons evenly distributed in a cone of open angle θ. Generate PMT responses to each segment, sum them up and compare with truth.

Parameter known: Number of segments

Parameters to be optimized: N_γ, θ_γ, segment positions

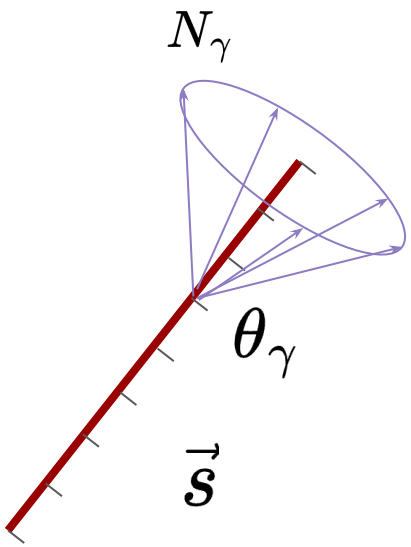
Plan of track reconstruction with OpticSiren



Can use the loss of OpticSiren as the minimizing parameter, e.g. propose a segment position and photon cone angle, generate the PMT hits, and compare to the truth. For now if we don't consider particle scattering, the first segment at the origin will be the only free parameter since all subsequent segments can be inferred from the origin.

But can we do better than this? Can we improve the minimization efficiency by some reasonable initial guess instead of randomly guessing the segment location in the whole detector? Or adding more constraints to the minimization quantity?

Finding segment position - vertex fitting



Method of pre-fitting in fiTQun:

$$G(\boldsymbol{x}, t) = \sum_{i}^{\text{hit}} e^{(-(T_i^{\text{res}}/\sigma)^2/2)},$$

where

$$T_i^{\text{res}} = t_i - t - |\boldsymbol{R}_i^{\text{PMT}} - \boldsymbol{x}|/c_n$$

Involves an empirical measurement of PMT time resolution σ, assuming all PMTs are the same -> would like to avoid this.

Finding segment position - vertex fitting