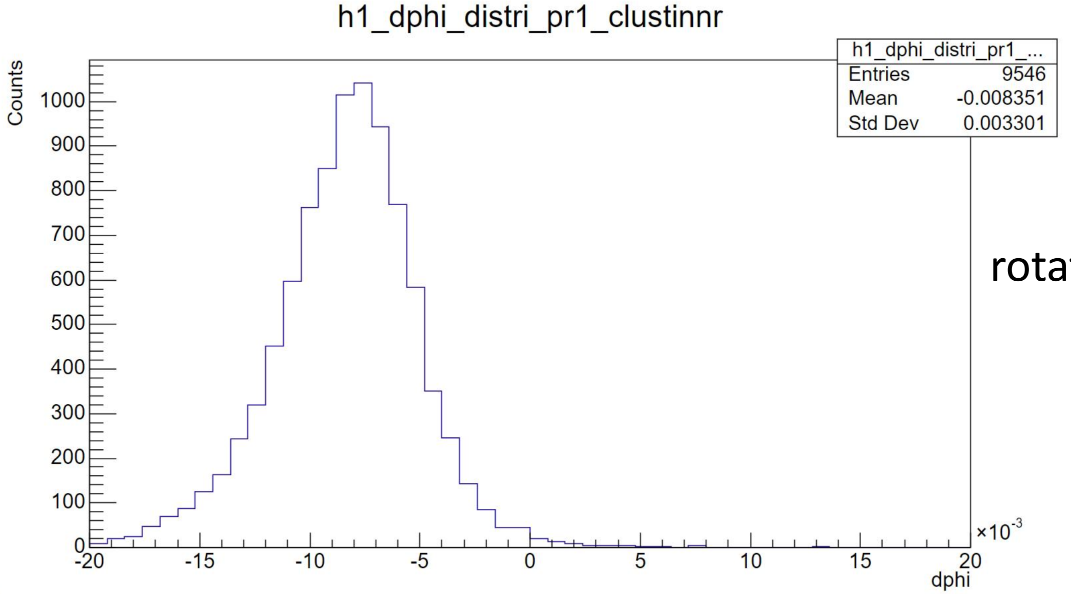
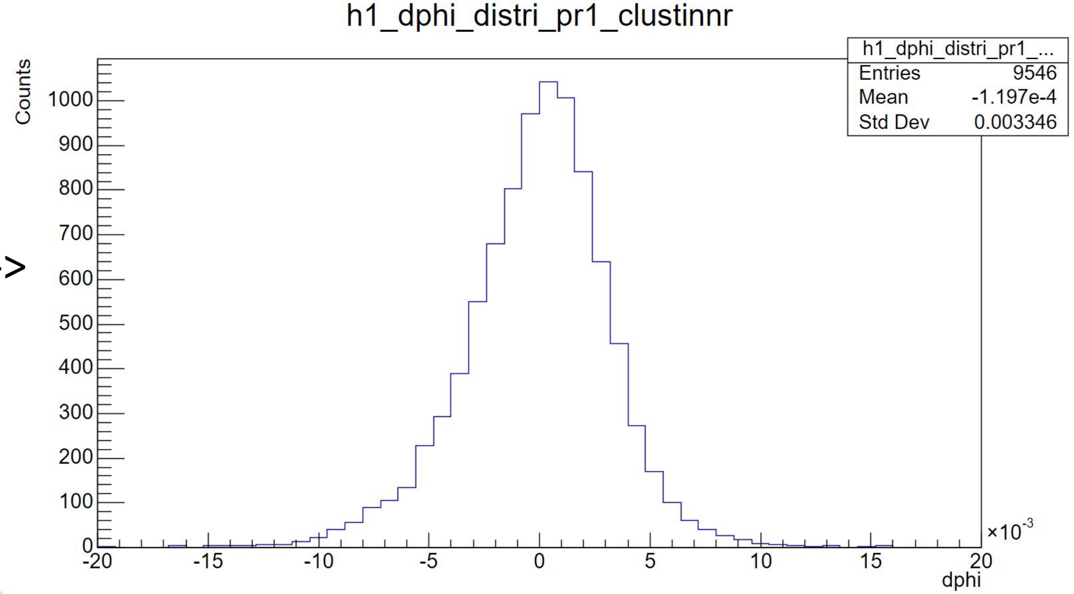
dphi(truth - reco) vs pT

truth_phi: electron hit on emcal surface phi
reco_phi: cluster inner face center phi



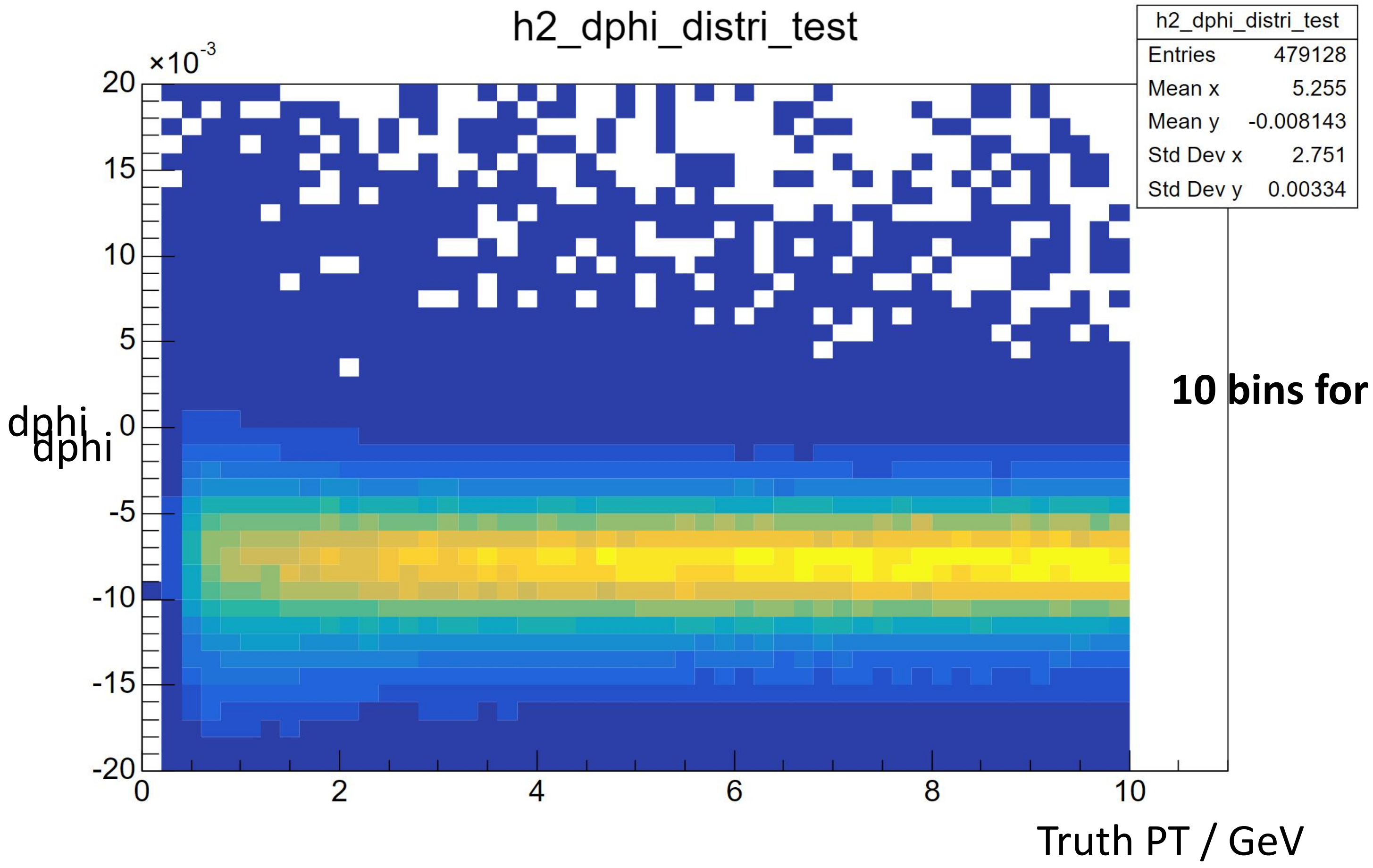
rotation -0.0083 rad -->



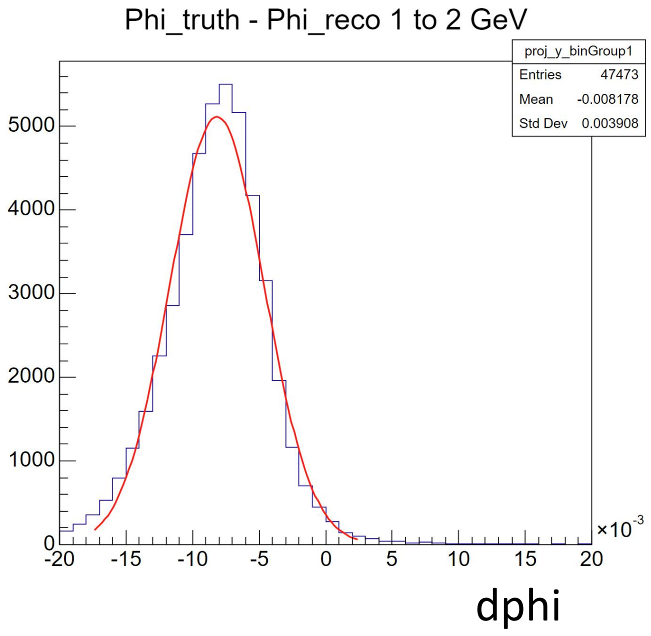
For the distributions without any corrections applied,
we studied the dphi performance in different pt range.

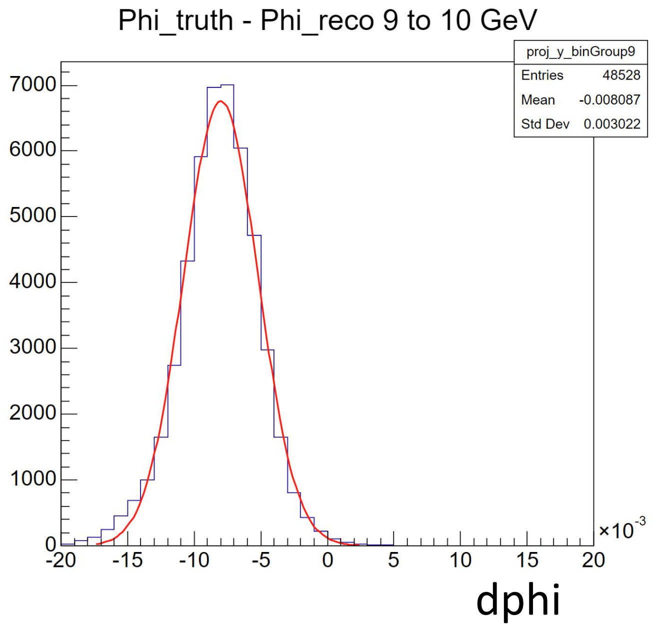
ps: There are many noisy towers in the EMCal, which take up a lot of storage. If we need to process a large number of events, we should apply an energy threshold when generating the pico DST; otherwise, we cannot be able to store many events.

dphi(truth - reco) vs pT

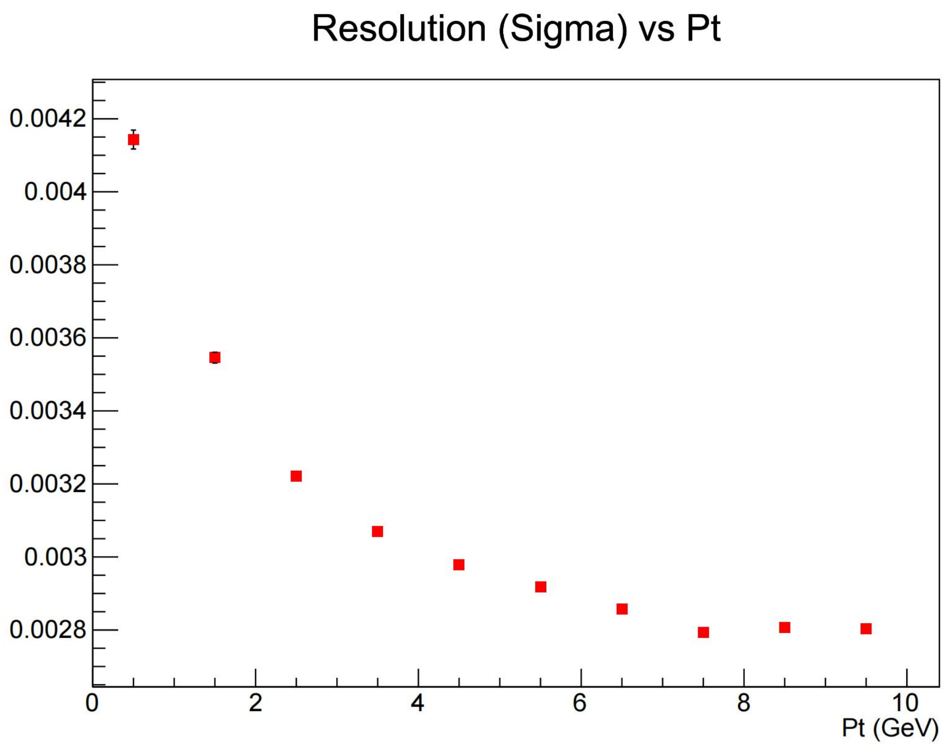


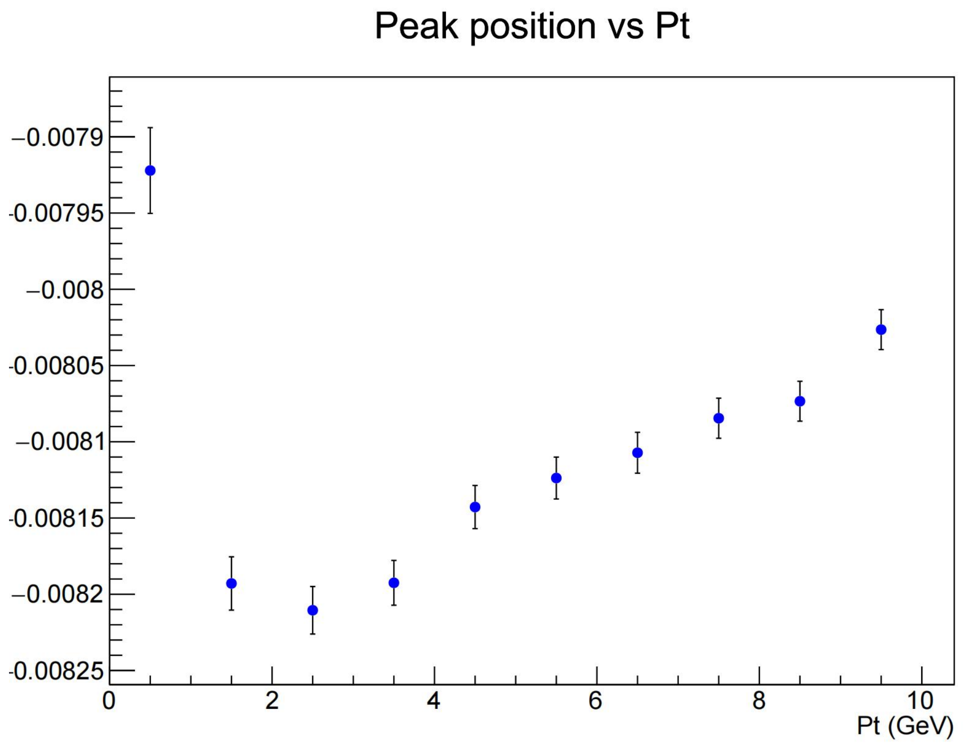
10 bins for 0-10 pt GeV





dphi(truth - reco) vs pT





ML4Reco

Jingyu

Event

1M events simulation Based on Fun4all on sPHENIX

INPUTGENERATOR::SimpleEventGenerator[0] -> add_particles(“e-”, 1);

INPUTGENERATOR::SimpleEventGenerator[0] -> set_vtx(0, 0, 0);

INPUTGENERATOR::SimpleEventGenerator[0] -> set_pt_range(0, 10); // GeV

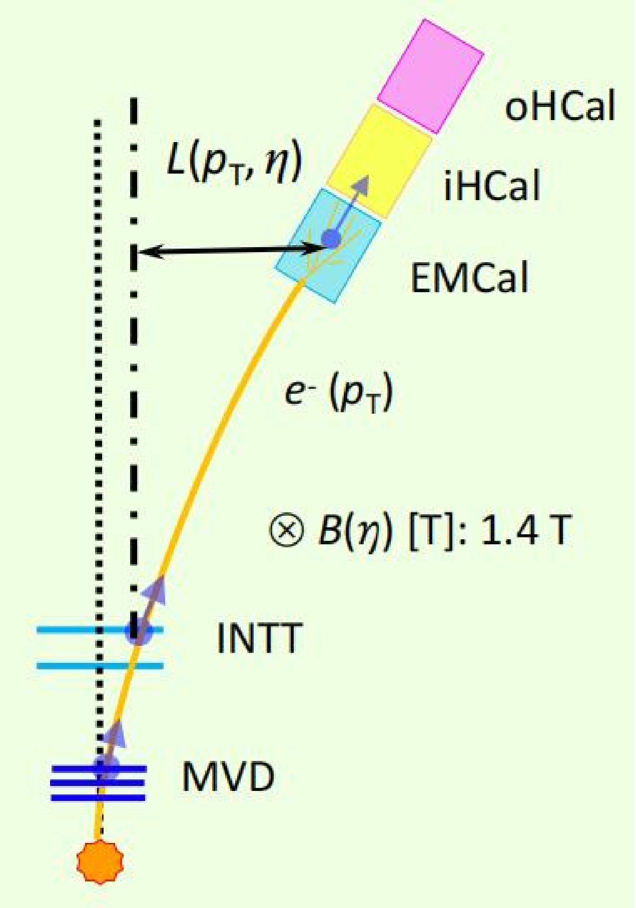
INPUTGENERATOR::SimpleEventGenerator[0] -> set_eta_range(-1.1, 1.1);

INPUTGENERATOR::SimpleEventGenerator[0] -> set_phi_range(-M_PI, M_PI);

0.5M for train and test, other 0.5M for showing performance

The data be used on showing performance is not same as train and test, no overfit in performance shown

Dataset



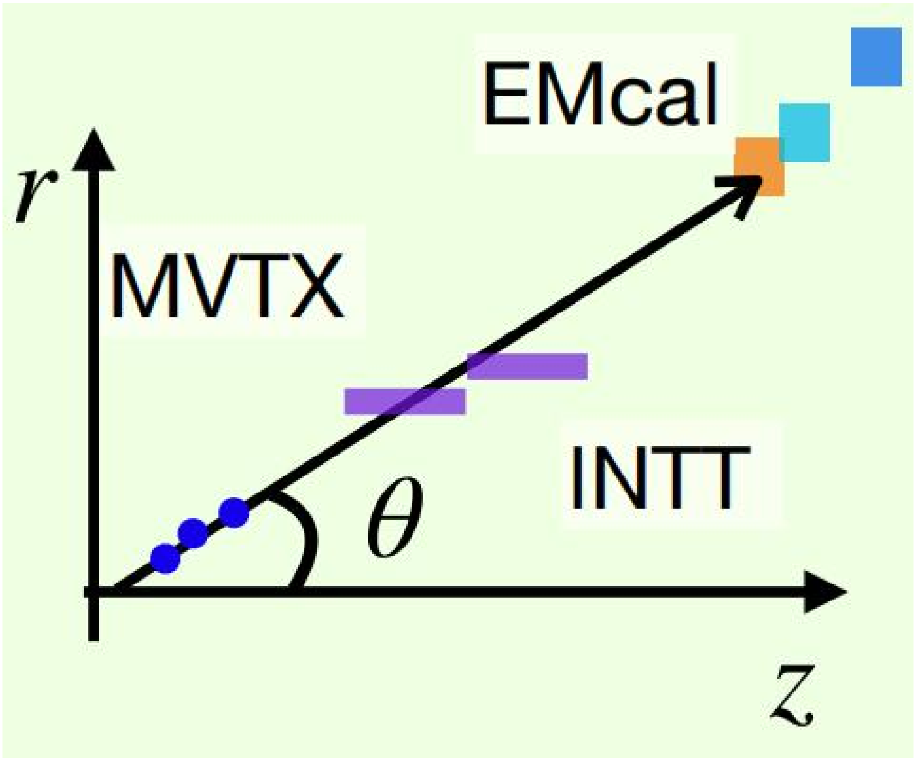
trk_feat: 5 hits position 5*3
inner/outer INTT only 1 clus
Select Closest Points of MVTX

calo_feat:
Calo cluster reco with inner face center,
Calo cluster reco with volume center,
9 towers that have max deposited energy (padding and cutting)

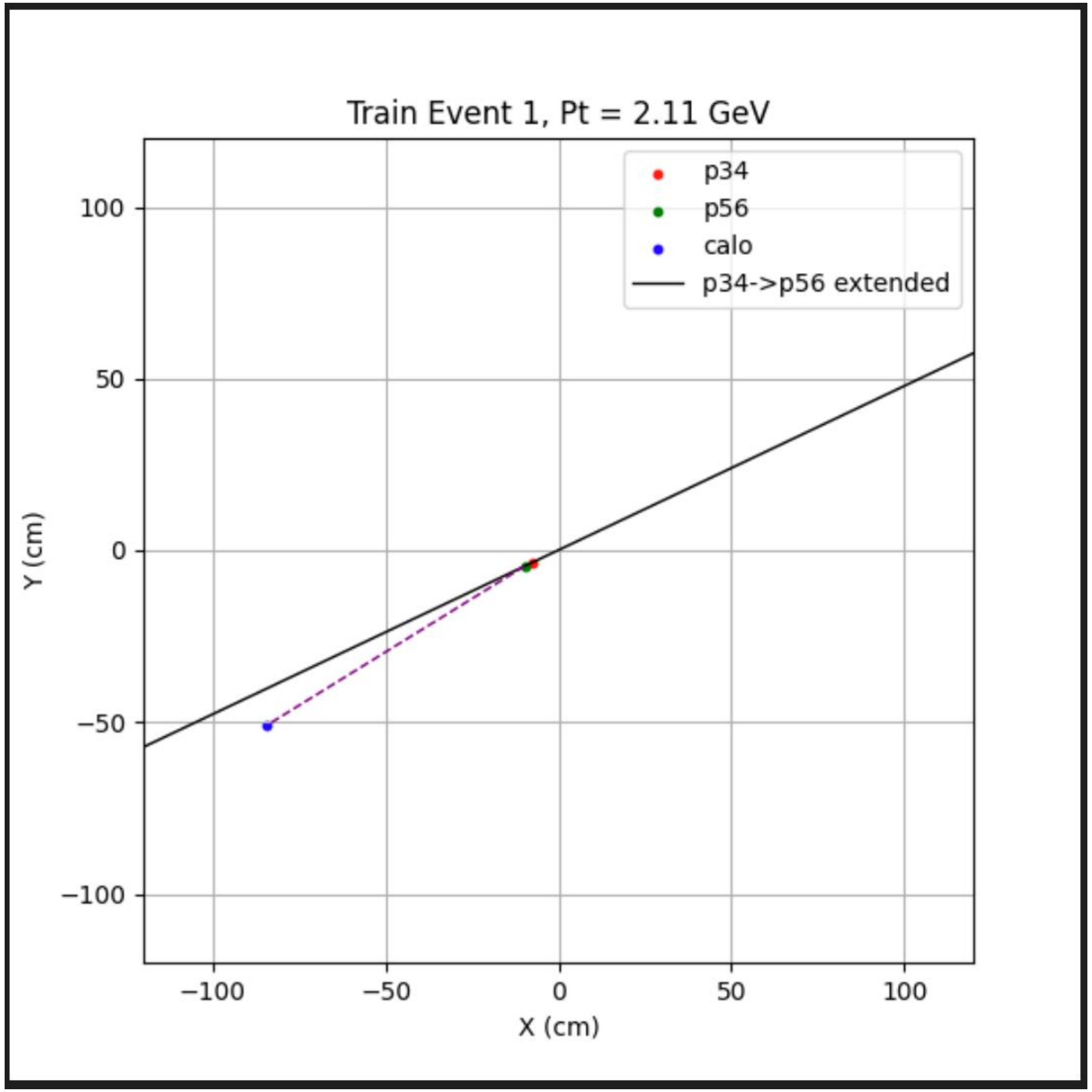
DATASETS:
X: Feat_select
Y: Truth_Pt

```
# data scaler
    if scaler is None:
        scaler = StandardScaler()
        data_X = scaler.fit_transform(data_X)
    else:
        data_X = scaler.transform(data_X)
```

Dataset



[INTT_R,Z + calo_R,Z,E]

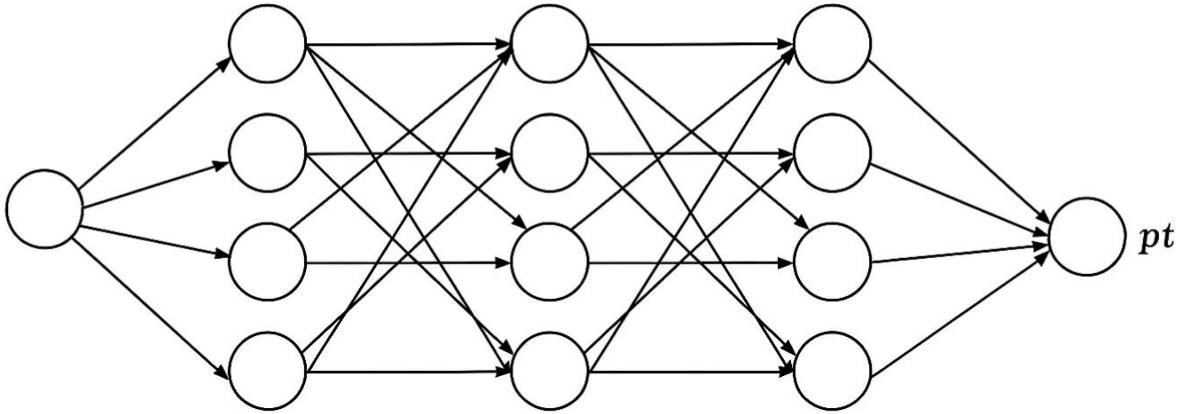


[1/ΔΦ]

Model

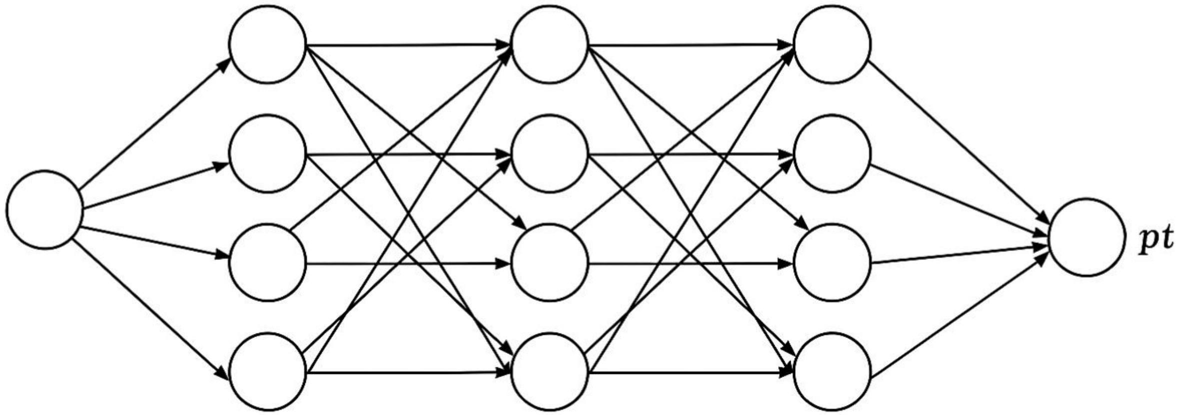
In the training from angle to pt, in addition to using resolution as the loss, some additional penalty mechanism are also needed.
Therefore, a simple approach is to train separately first, and then combine the results.

[INTT_R,Z + calo_R,Z,E]



MLP1: 3hidden layer, hidden_dim=256, nn.Dropout(0.2)

[1/ΔΦ]



MLP2: 3hidden layer, hidden_dim=256, nn.Dropout(0.2)

Train_model1 - [INTT_R,Z + calo_R,Z,E]

Hyper parameters:

- batch_size=1024, epochs=200, lr=5e-5, val_ratio=0.3

Loss function:

- pt_reso = (yb - pred) / (yb + 1e-6)

  Relative resolution,
  1e-6 to prevent division by zero.

- weights = (pt_reso.abs() < 0.2).float() * 2.0 + 1.0

  Increase the weight inside the peak,
  reduce the influence of outlier data points

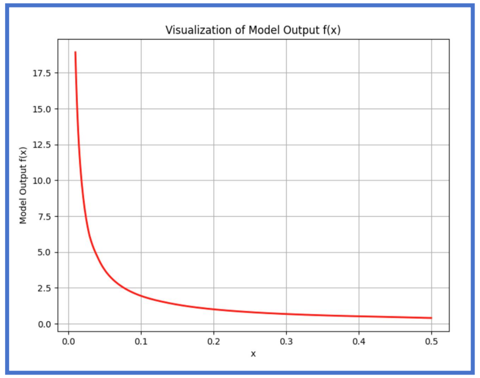
- loss = ((pt_reso) ** 2 * weights).mean()

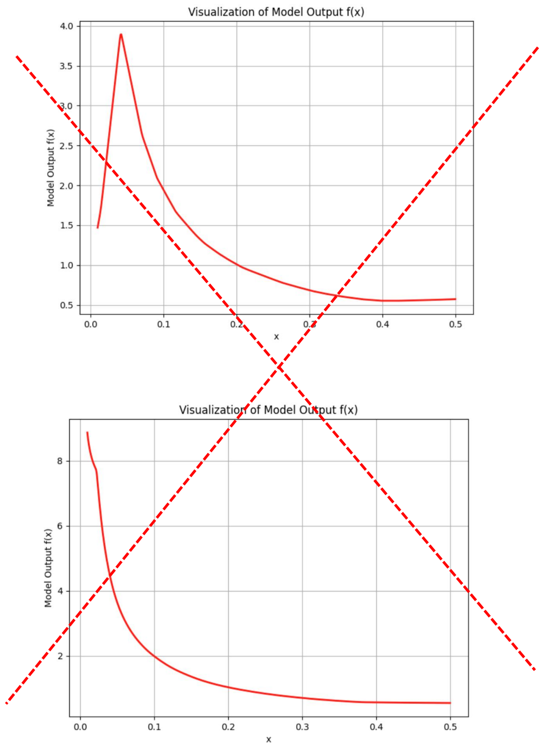
  Use squared values instead of absolute values to
  make the peak position closer to zero.

- if val_loss < best_val_loss: Saved best model

Train_model2 - [1/ΔΦ]

model visualization





epochs=500.

Loss function:

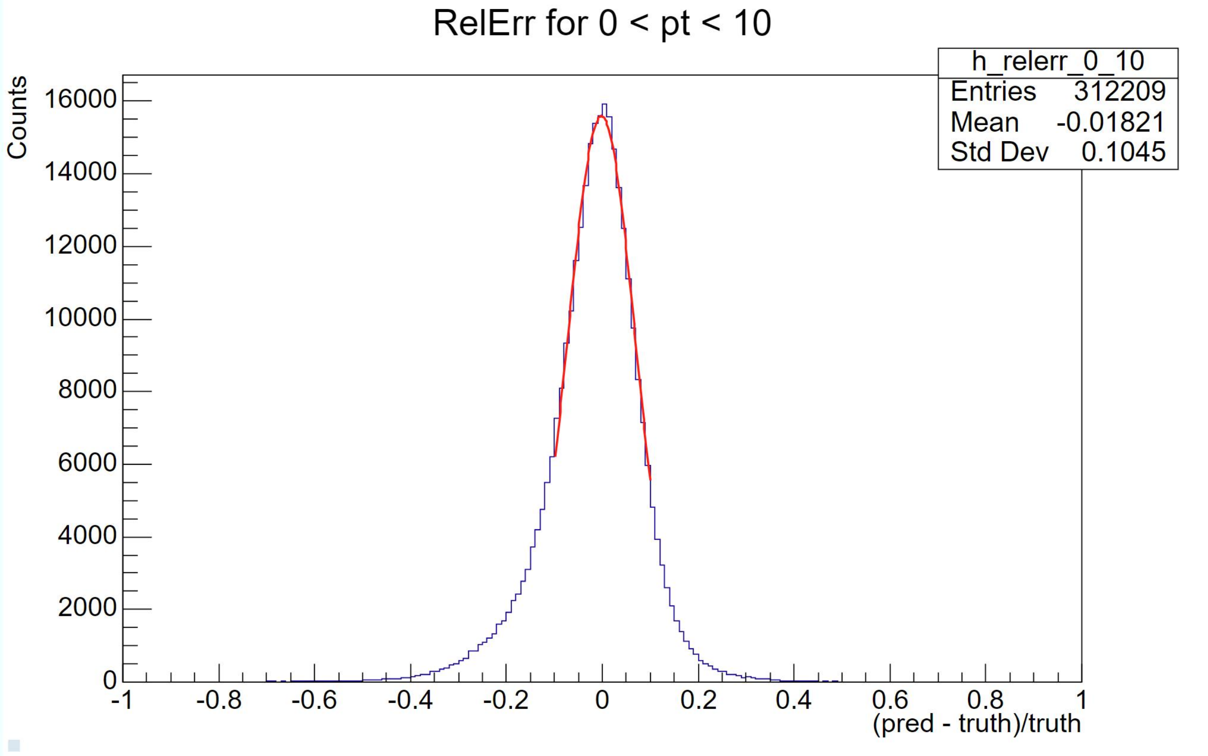
- pt_reso = (yb - pred) / (yb + 1e-6)
- weights = (pt_reso.abs() < 0.2).float() * 2.0 + 1.0
- main_loss = ((pt_reso) ** 2 * weights).mean()
- monotonic_loss
  - lambda_mono = 0.3

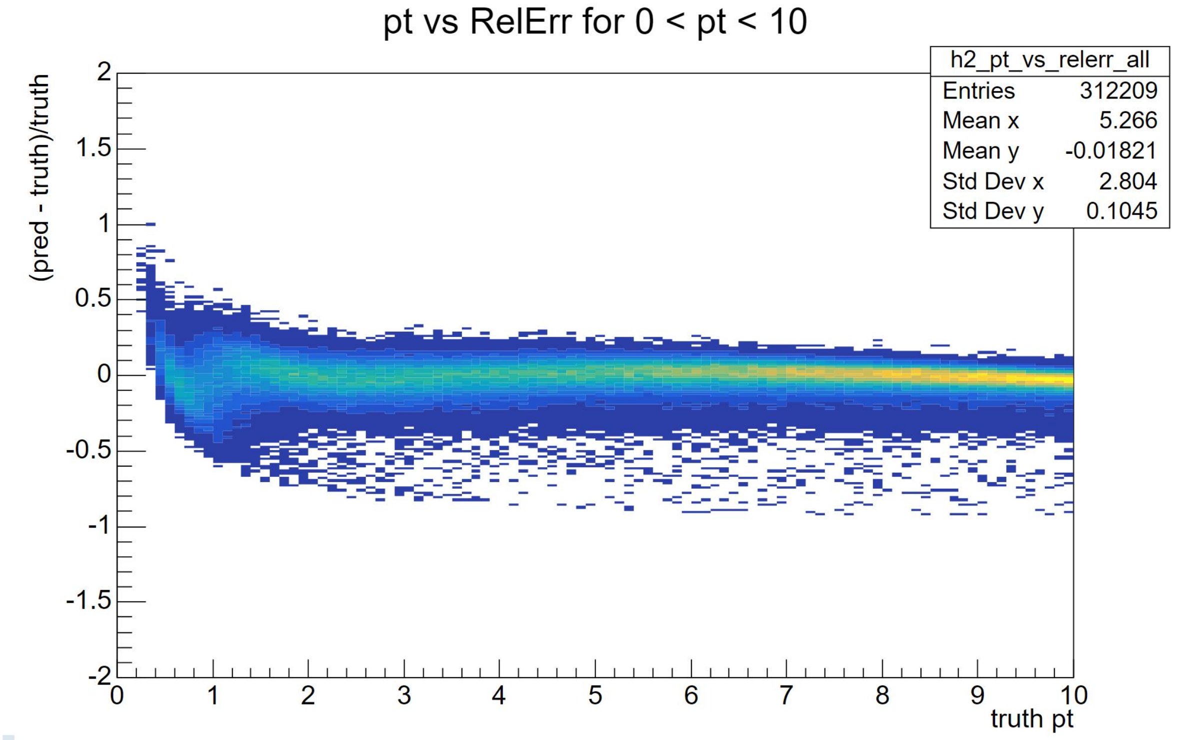
requires pt to decrease as the angle increases; otherwise, oscillations may occur.

- boundary_loss
  - [0.5, 1, 2, 10, 15, 25, 50, 100, 200] ->[0.0961, 0.1922, 0.3844, 1.922, 2.883, 4.805, 9.61, 19.22, 38.44] :
  - lambda_boundary = min(0.005 * epoch, 0.2)
- loss = main_loss
- + lambda_mono * monotonic_loss
- + lambda_boundary * boundary_loss

Add boundary conditions outside the data range to ensure that the pt estimate is sufficiently elevated at small angles.
Dynamic weighting to prevent affecting data learning in the early stages.

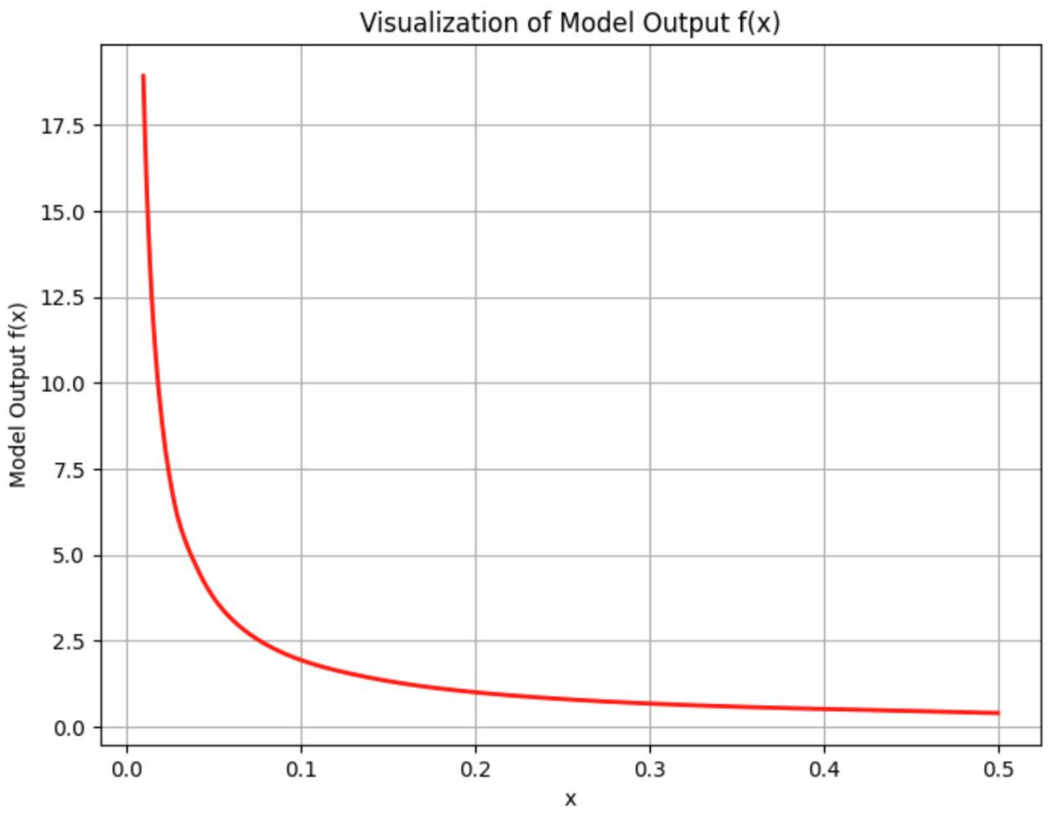
Model1 - [INTT_R,Z + calo_R,Z,E]

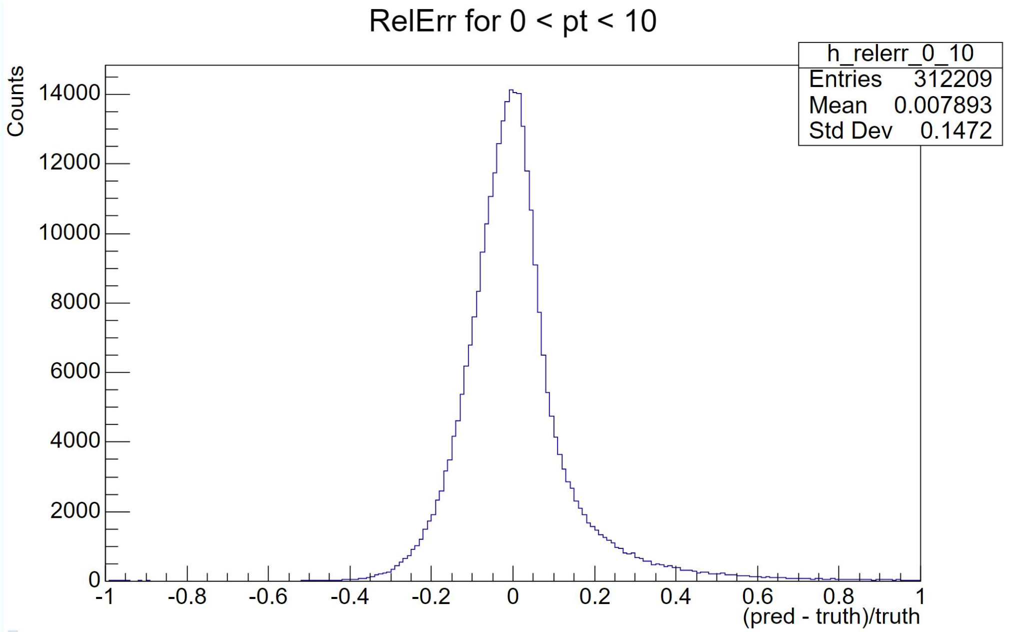


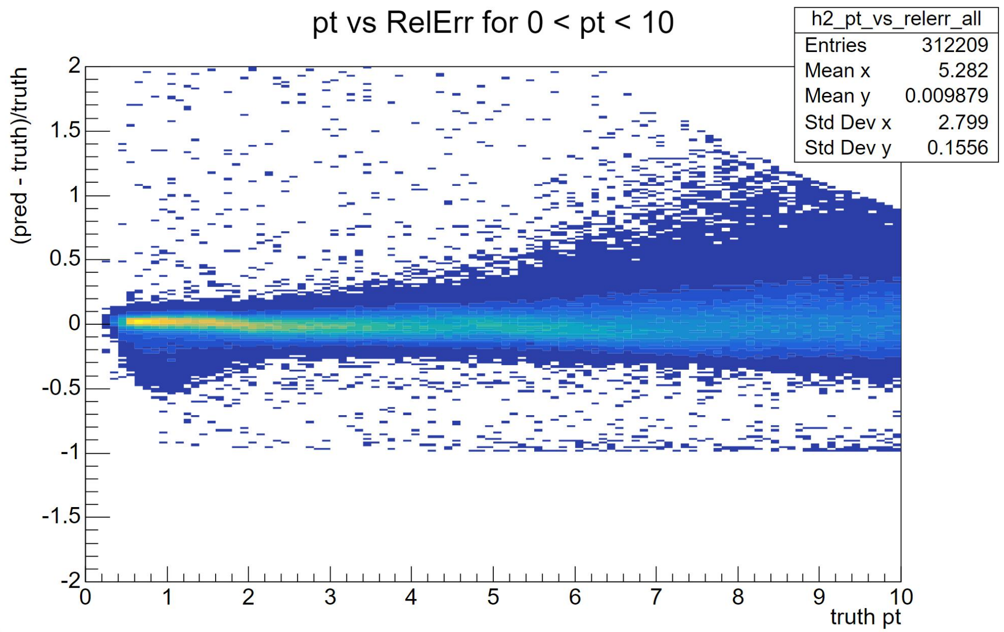


Model2 - [1/ΔΦ]

model visualization







Train_combined model 1&2

X: [dphi_i, pt_pred1_i, calo_edep_i, pt_pred2_i] + pt_bin_onehot

Y: Truth_Pt

pt_est = 0.5 * (pt_dphi + pt_energy)
# embed
pt_bin_edges = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
pt_bin_onehot = [0] * 10

MLP: 3hidden layer, hidden_dim=128

epochs=300

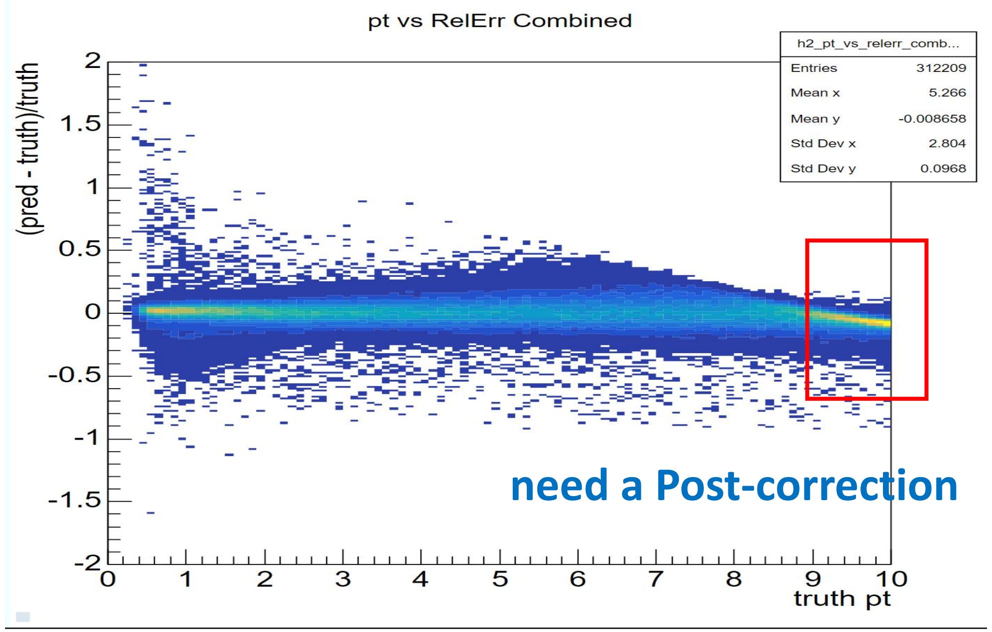
Loss:

pred = model(xb)

    pt_reso = (yb - pred) / (yb)

    weights = (pt_reso.abs() < 0.2).float() * 2.0 + 1.0

    loss = ((pt_reso) ** 2 * weights).mean()



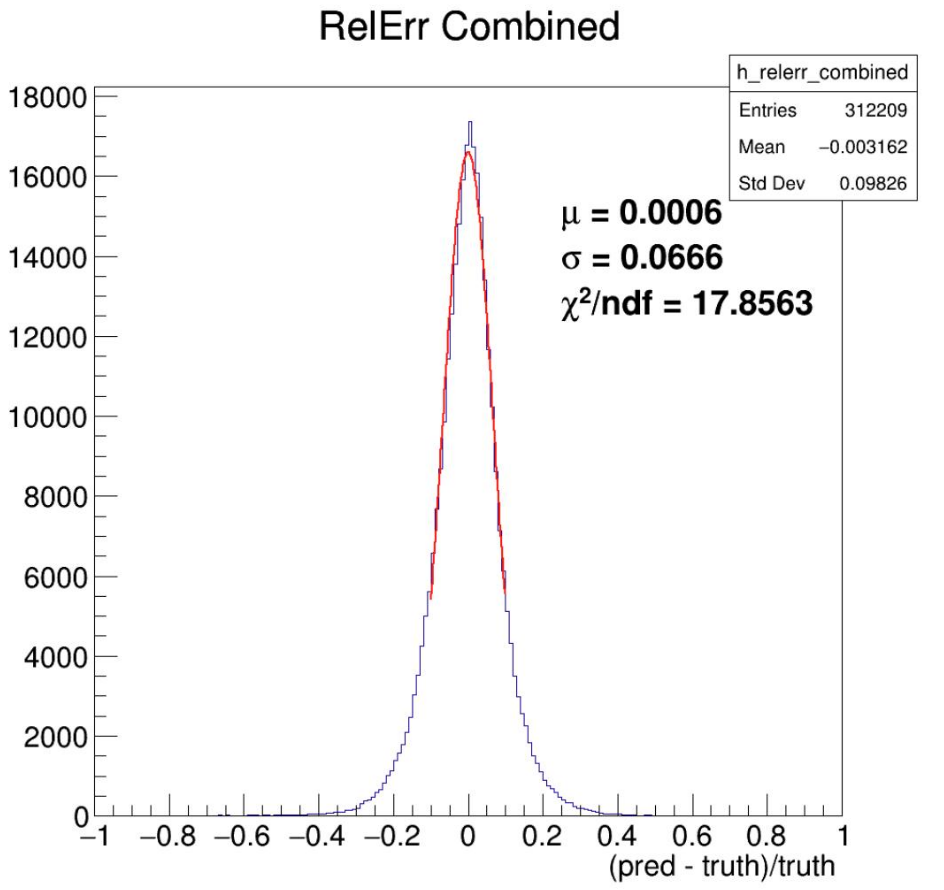
Performance

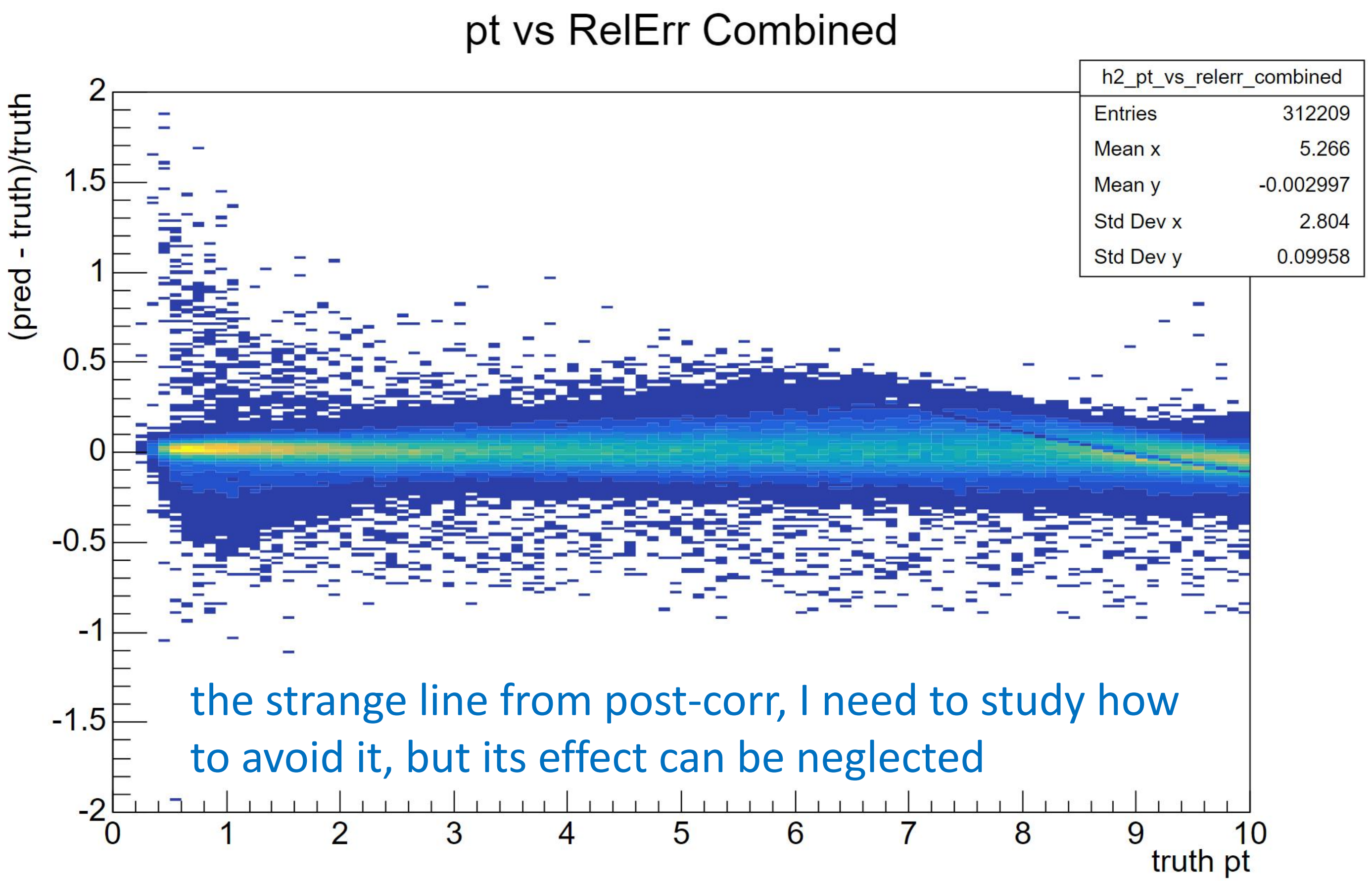
Post-correction

if reco_pt >= 8.8:

correction_factor = 0.02 + 0.08 * (reco_pt - 8.8)

pred_np[i] = reco_pt * (1.0 + correction_factor)





Better results: Better efficiency, Better bias, Better resolution

Summary

- Better efficiency: Only i-o INTT and calo have hits, with cluster deposited energy greater than 0.5 GeV.
- Better bias: The mean value and Gaussian peak are closer to zero compared to other reconstruction methods.
- Better resolution: The distribution is more symmetric, with the smallest width and the smallest standard deviation.

- The workflow, code, model configuration, hyper parameters, and post-corrections still need to be carefully checked.

Back up