Sim/Reco May 14th

I Cheong Hong

From last collaboration meeting:

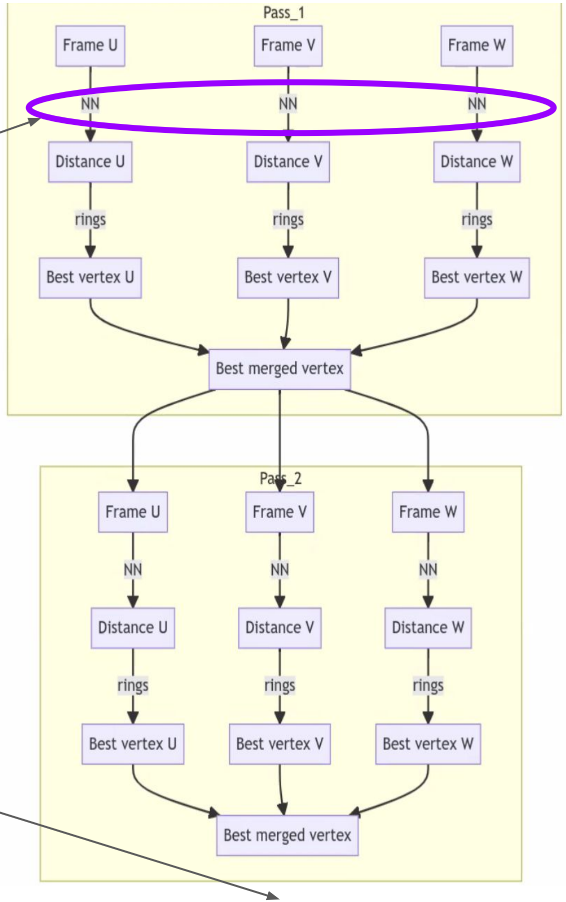
1. Modify current algo: Since **70%** of the failure sample already happens at Pass 1 NN, replace current NN

OR

2. Let current algo unchanged, and Filter&Fix after

A) Use a filter to find bad vertices.
   - ex: check if any track is flipped using en deposition pattern;

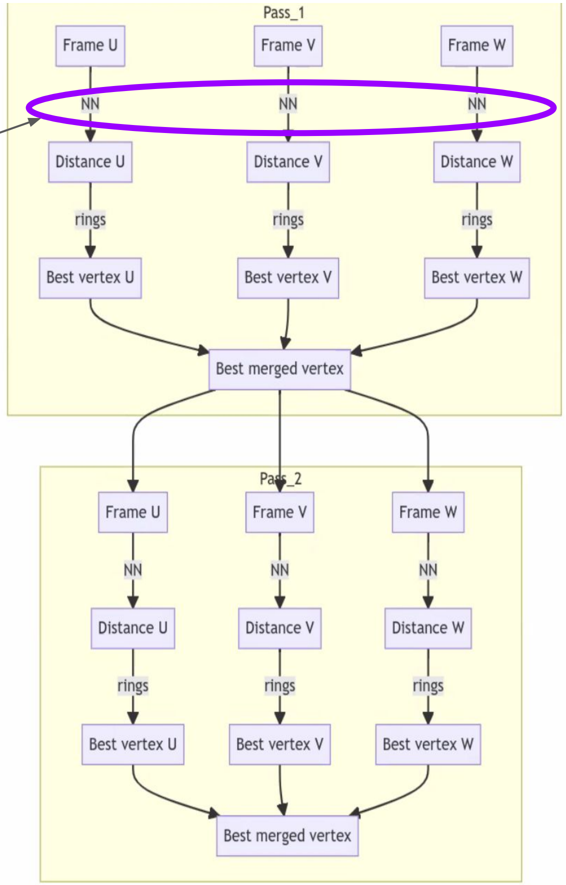
B) Create another algo (standard or ML) to find the right vtx position



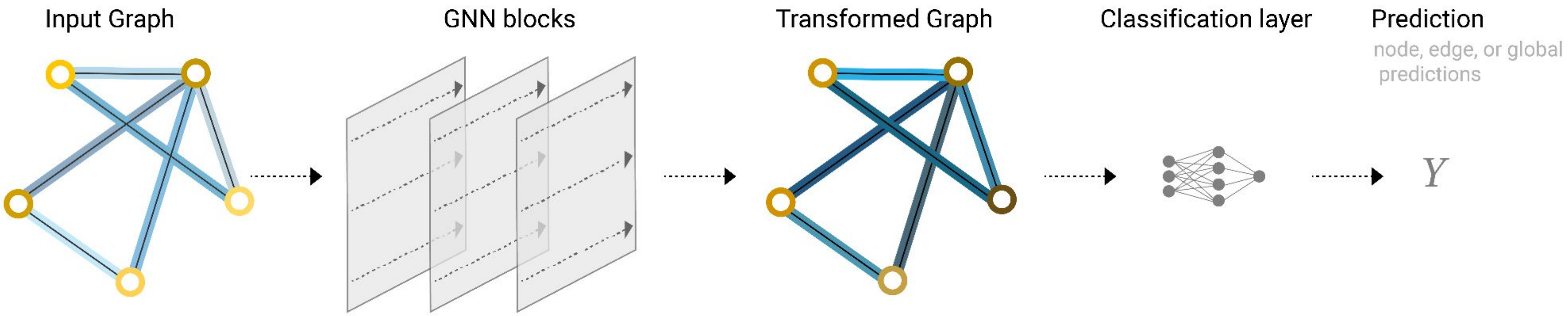
From last collaboration meeting:

1. Modify current algo: Since **70%** of the failure sample already happens at Pass 1 NN, replace current NN

Trying to use a Graph neural network (GNN) to replace the current algo



Just a little bit intro for GNN….



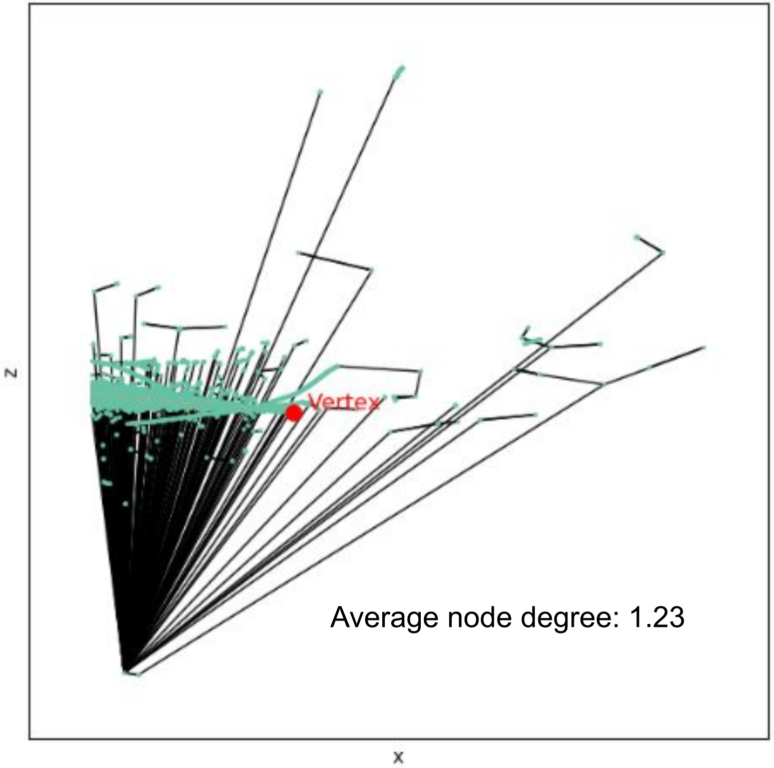
In order to use a GNN for vertex reco, we must also have a method to transform hits to graph (call it node-connection method in the slides afterwards)

What have done since last collab meeting:

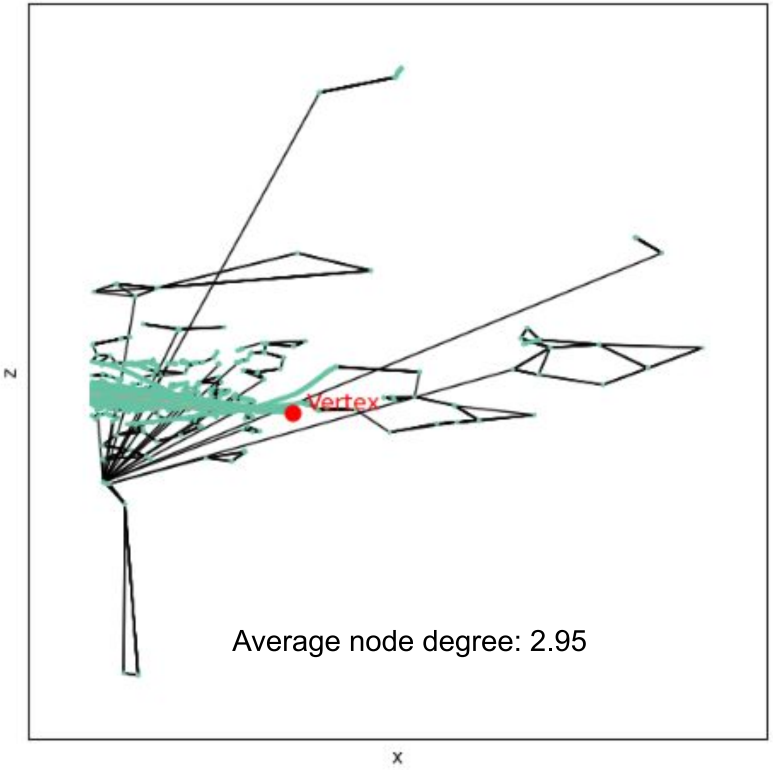
1. Testing different node-connection methods
2. Tried to optimse model

All the studies are done with 100k atm cc+nc samples (only U views for now)

1.Testing different node-connection method

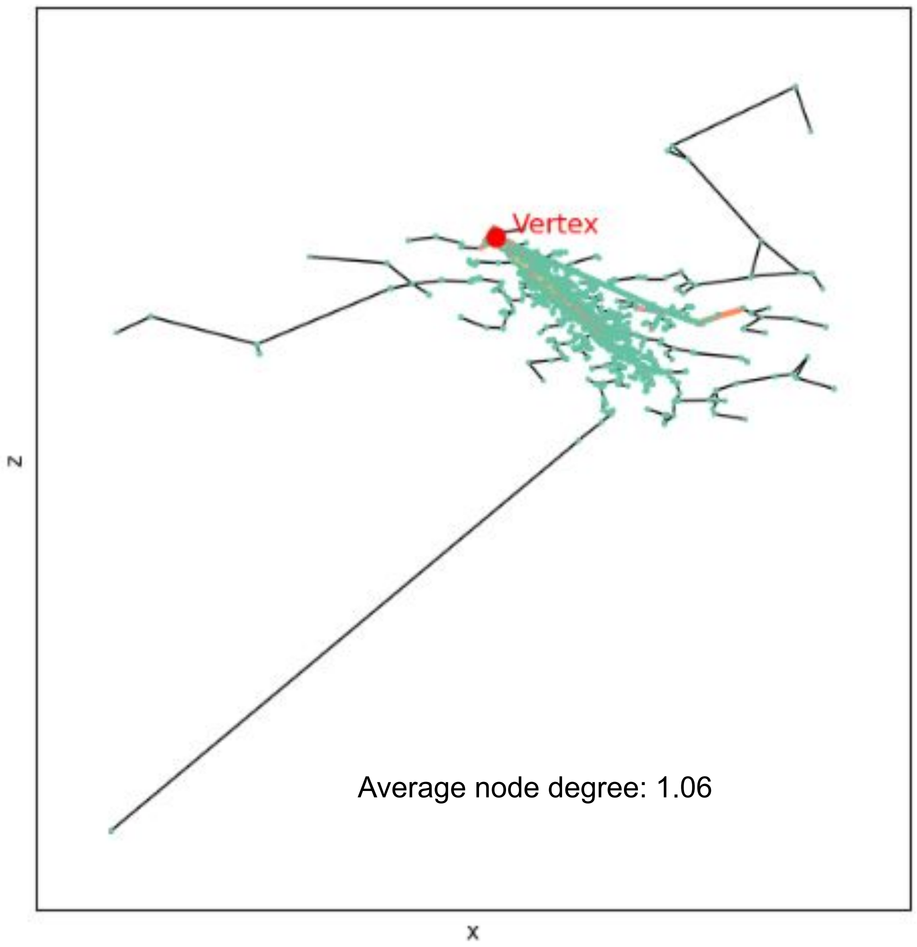


Fully connected 2
Nearest Neighboor



Fully connected
5 Nearest Neighboor

Node-connection method



Pandora’s Method

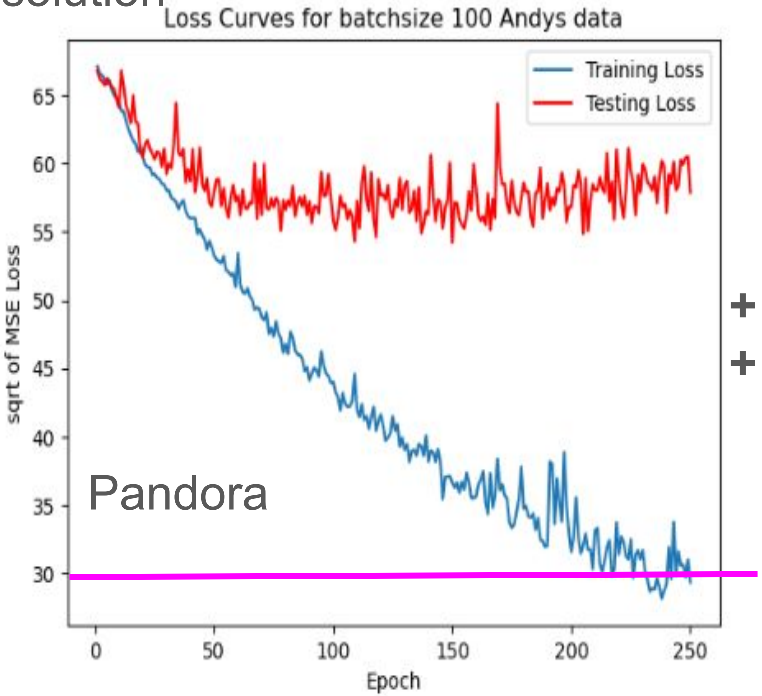
https://indico.fnal.gov/event/63824/contributions/286696/attachments/176241/239368/gnn_transformer_inputs.pdf

Prototype Model to test different graph

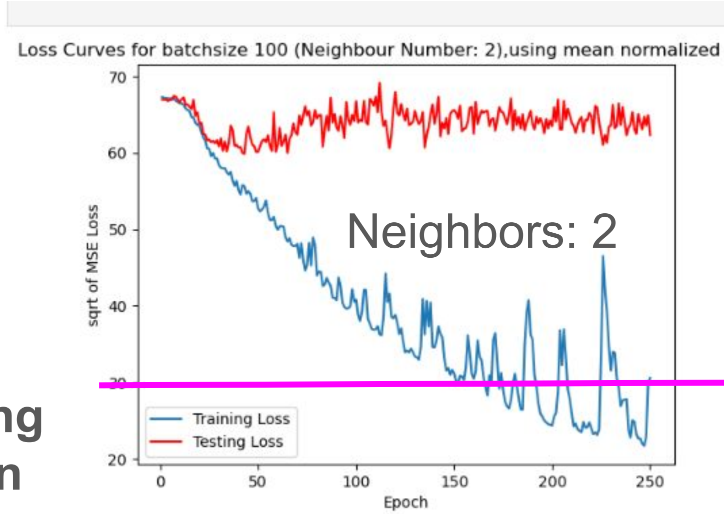
```
56]: GNN_DUNE(
       (convs): ModuleList(
         (0): SAGEConv(3, 64, aggr=mean)
         (1): SAGEConv(64, 64, aggr=mean)
         (2): SAGEConv(64, 64, aggr=mean)
         (3): SAGEConv(64, 64, aggr=mean)
         (4): SAGEConv(64, 64, aggr=mean)
         (5): SAGEConv(64, 64, aggr=mean)
         (6): SAGEConv(64, 64, aggr=mean)
         (7): SAGEConv(64, 64, aggr=mean)
         (8): SAGEConv(64, 64, aggr=mean)
         (9): SAGEConv(64, 64, aggr=mean)
       )
       (linears): ModuleList(
         (0): Linear(in_features=64, out_features=64, bias=True)
         (1): Linear(in_features=64, out_features=64, bias=True)
         (2): Linear(in_features=64, out_features=2, bias=True)
       )
     )
```

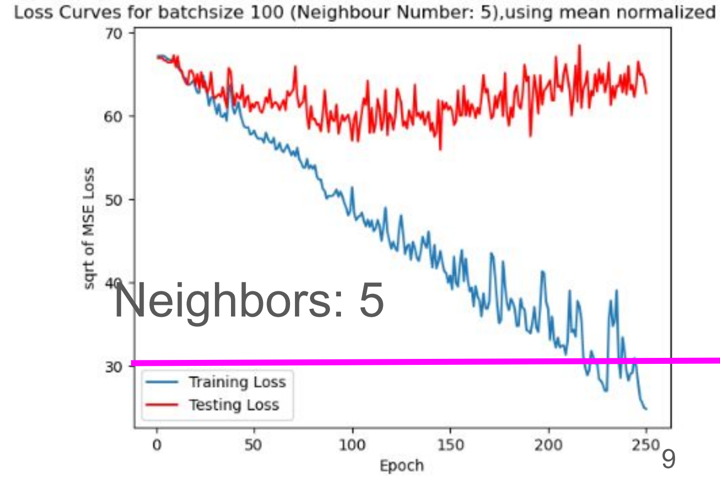
Activation: relu

Node connection method; y-axis: vtx resolution



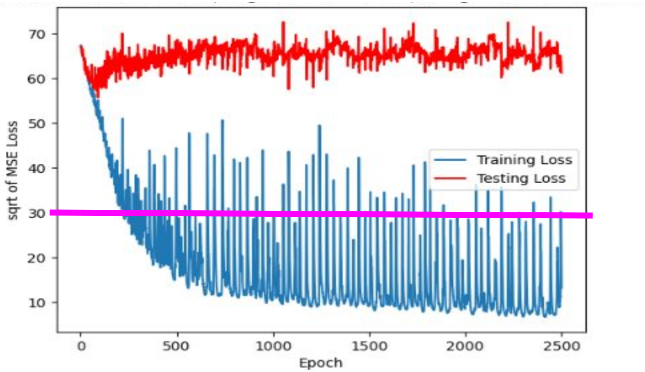
+ **Huge Overfitting**
+ **Poor resolution**

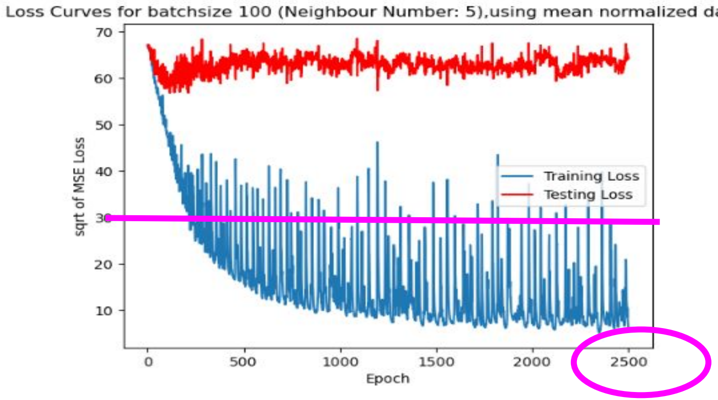


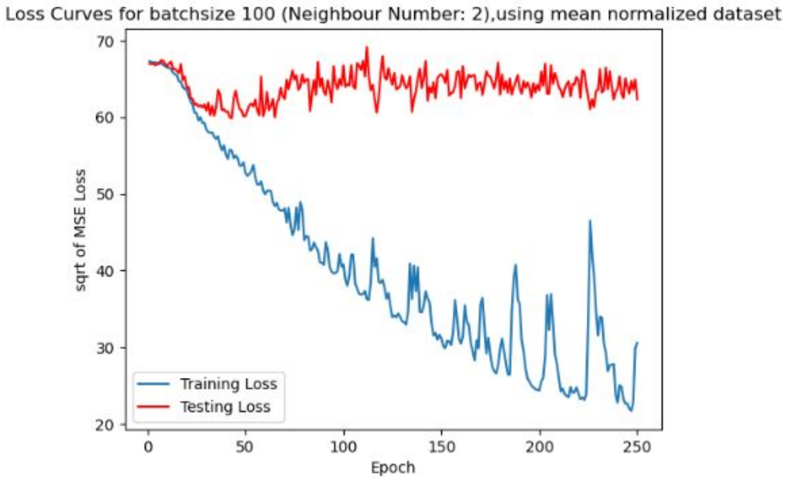


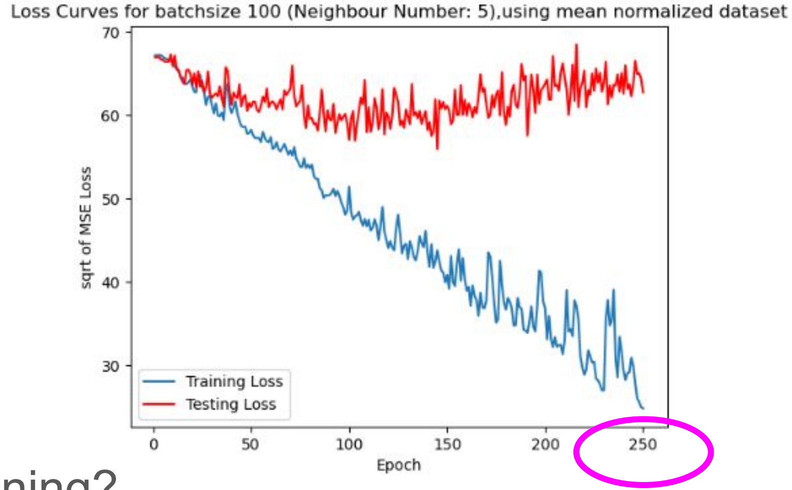
Seems the node-connection method doesn't matter much→ used Pandora method for further studies
Ptential problem: all methods don't contain any long edges

Node connection method; y-axis: vtx resolution





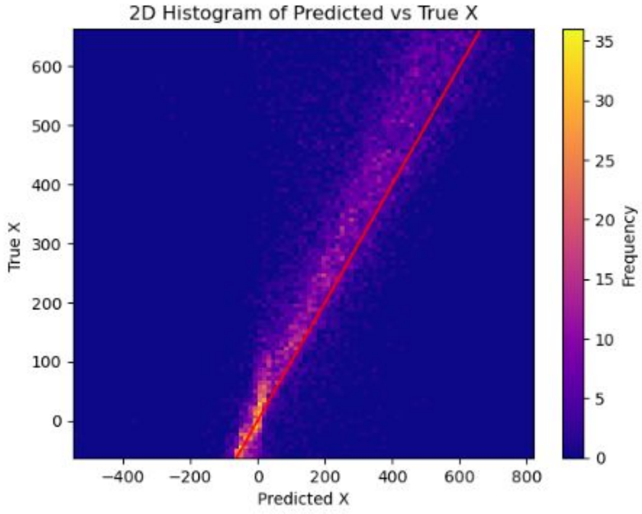




+ Huge Overfitting
+ Poor resolution → solved w/ more training?

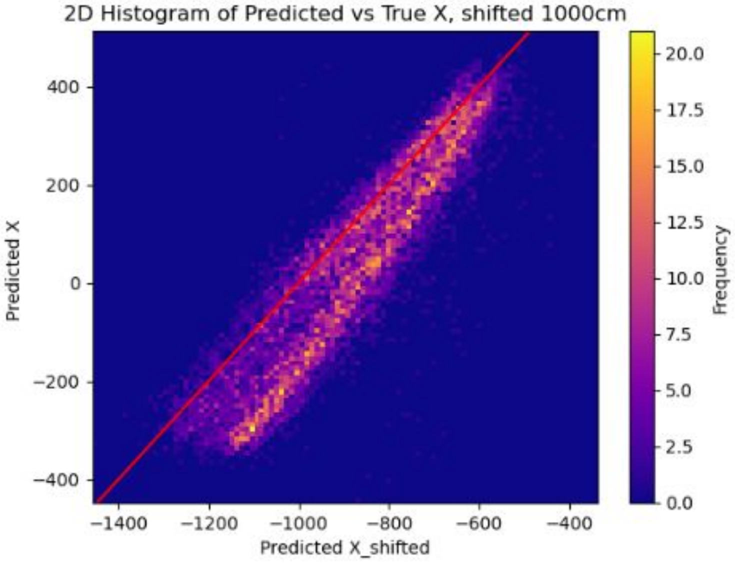
Pre-processing of data

We hope to produce a model that's translational invariant



True X

```
plt.title('2D Histogram of reco X against reco X shifted , shifted 1000cm')
plt.show()
```



However, the network is not very good at the original coordinate system

Pre-processing of data

```
graph_files = torch.load("/pbs/throng/lbno/ichong/Graph_pytensor/100k_data_U.pt")
train_data, test_data = split_train_test(
    graph_files, test_size=0.2, random_state=42
)
train_data = mean_normalization(fix_data(train_data))
test_data = mean_normalization(fix_data(test_data))
```

Mean normalization: shift the x,z information of graph by value of mean_x,mean_z → so the mean of x_new and z_new is 0 , which solve the translation variant problem

2.Tried to optimse model

Workflow

1. Produce graph from Pandora,save it to binary files
2. Read the binary files in python
3. Train the model using Optuna

Training

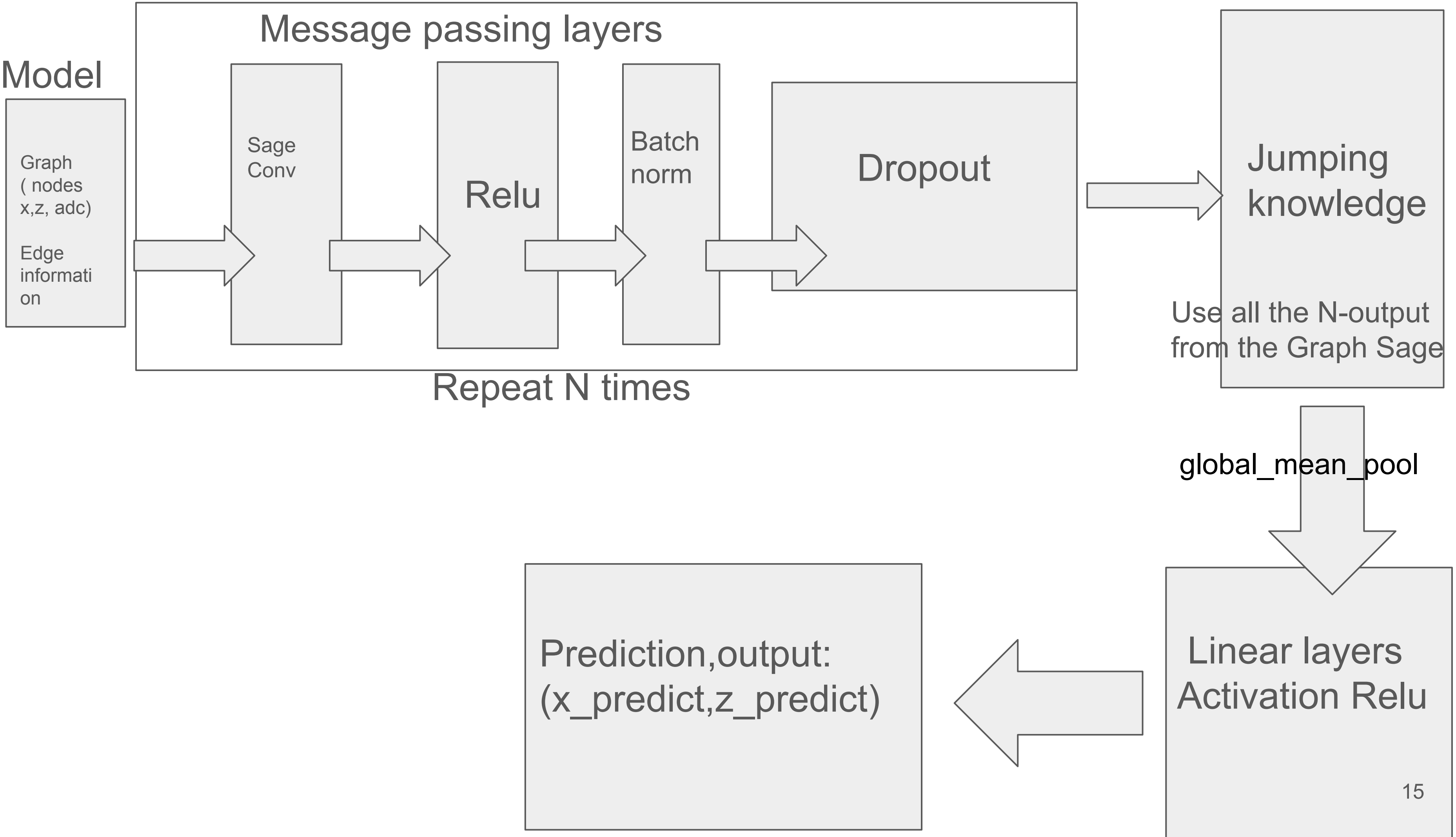
Loss function: Mean square error
True_vtx - prediction

```python
criterion = nn.MSELoss()
optimizer = torch.optim.Adam(
    model.parameters(), lr=trial.suggest_float("lr", 1e-5, 1e-1)
)

train_loader = DataLoader(train_data, batch_size, shuffle=True)
test_loader = DataLoader(test_data, batch_size, shuffle=True)

# Train and test the model
train_losses = []  # List to store training losses
test_losses = []  # List to store testing losses

for epoch in range(n_epochs):  # Iterate over epochs
    model.train()
    epoch_train_loss = 0.0
    epoch_test_loss = 0.0
    total_batches = len(train_loader)  # Total number of batches
    print(f"The current epoch is {epoch}", flush=True)
    # Training Phase
    for data in train_loader:  # Iterate in batches over the training dataset.
        data = data.to(device)# Move data to GPU
        transform = T.Compose([T.ToUndirected(), T.VirtualNode()])
        data = transform(data)

        out = model(
            data.x, data.edge_index, data.batch
        )  # Perform a single forward pass.
        loss = criterion(torch.stack(data.y, axis=1), out)
        # Compute the loss.
        # loss = loss/batch_size
        optimizer.zero_grad()  # Clear gradients.
        loss.backward()  # Derive gradients.
        optimizer.step()  # Update parameters based on gradients.
        epoch_train_loss += loss.item()

    epoch_train_loss /= total_batches  # Calculate average epoch training loss
    train_losses.append(
        epoch_train_loss
    )  # Append the average epoch training loss to the list

    # Testing Phase
    model.eval()
    with torch.no_grad():
        for data in test_loader:
            data = data.to(device)  # Move data to GPU
            out = model(data.x, data.edge_index, data.batch)
            loss = criterion(torch.stack(data.y, axis=1), out)
            epoch_test_loss += loss.item()

    epoch_test_loss /= len(test_loader)  # Calculate average epoch testing loss
    test_losses.append(
        epoch_test_loss
    )  # Append the average epoch testing loss to the list
    print('The prediction is ',out[0:5])
    print('The truth vtx is', torch.stack(data.y, axis=1)[0:5])
    # Report the test loss to Optuna
    trial.report(np.sqrt(epoch_test_loss), epoch)
    # Handle pruning based on the intermediate value.
    if trial.should_prune():
        raise optuna.exceptions.TrialPruned()
```

Def Loss function, using MSELoss

Transform it to undirected graph + master nodes

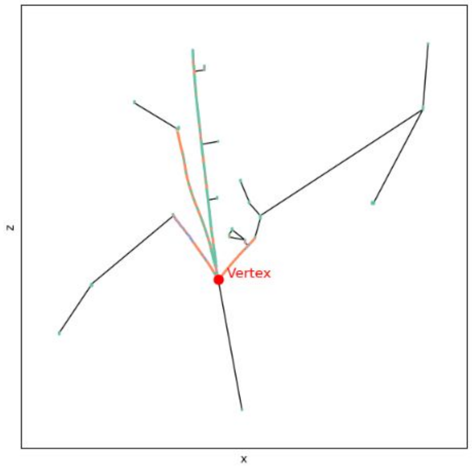
Get the prediction from the model, calculate the loss


Model
Message passing layers
Graph
(nodes
x,z, adc)
Edge
informati
on
Sage
Conv
Relu
Batch
norm
Dropout
Repeat N times
Jumping
knowledge
Use all the N-output
from the Graph Sage
global_mean_pool
Linear layers
Activation Relu
Prediction,output:
(x_predict,z_predict)

Sanity check:

1.Input data --- Plot them out to check
2.Prediction from model --- Print the prediction out
3.Model using -- print the model in the beginning
4.The truth label --- check them on the graph diagram

1&4



2.

```
The current epoch is 8
The prediction is  tensor([[  5.5826, -13.1422],
        [  3.8386, -10.5769],
        [ -5.9819,  23.4124],
        [ -7.1553,  20.8719],
        [  4.8695,  -6.7268]], device='cuda:0')
The truth vtx is tensor([[-40.3471, -13.6659],
        [ -6.4391,  -1.9618],
        [ -2.5167,  23.8845],
        [-10.1236,  15.9911],
        [-76.3425,  51.6819]], device='cuda:0')
```

3.

```
GNN_DUNE(
  (convs): ModuleList(
    (0): SAGEConv(3, 115, aggr=mean)
    (1-6): 6 x SAGEConv(115, 115, aggr=mean)
  )
  (batch_norms): ModuleList(
    (0-6): 7 x BatchNorm1d(115, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (dropouts): ModuleList(
    (0-10): 11 x Dropout(p=0.25377264690975404, inplace=False)
  )
  (jump): JumpingKnowledge(cat)
  (linears): ModuleList(
    (0): Linear(in_features=805, out_features=1, bias=True)
    (1-4): 4 x Linear(in_features=1, out_features=1, bias=True)
    (5): Linear(in_features=1, out_features=2, bias=True)
  )
)
The current epoch is 0
```

HyperPara optimization: Optuna!

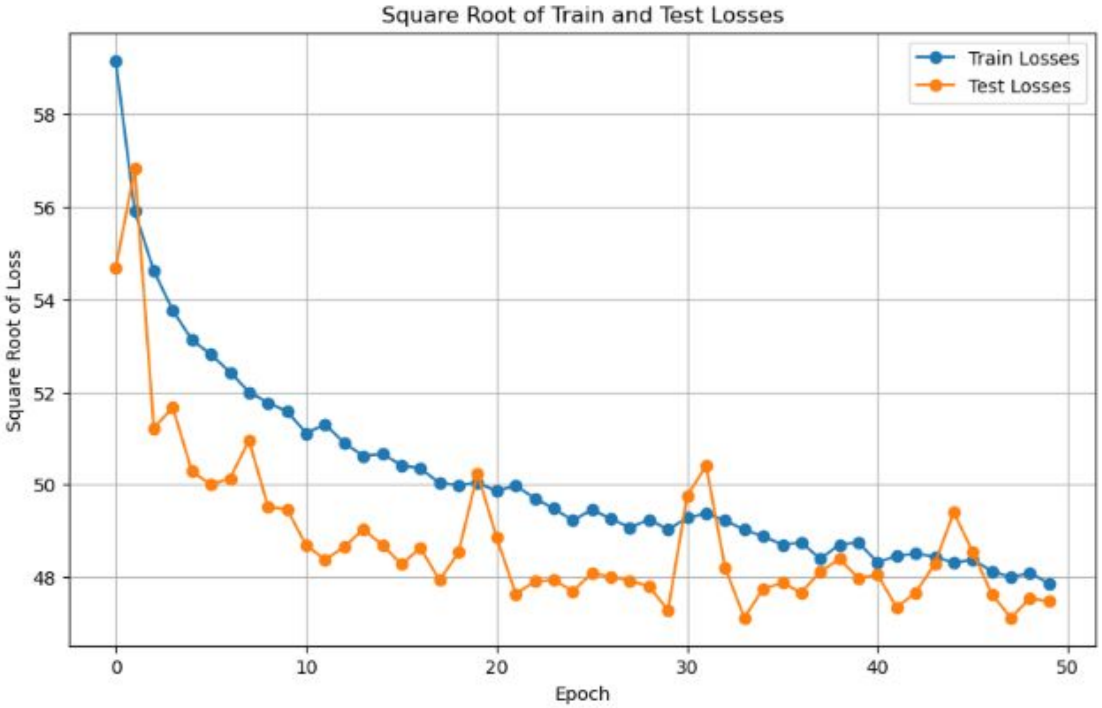
1. param list

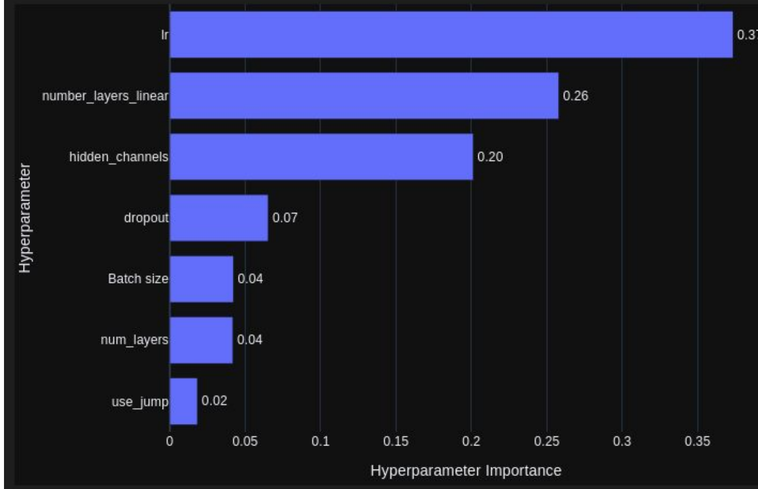
2. param ranges

```python
num_layers = trial.suggest_int("num_layers", 1, 10)
number_layers_linear = trial.suggest_int("number_layers_linear",1,20)
hidden_channels = trial.suggest_int("hidden_channels", 1, 700)
jk_hidden_channel = trial.suggest_int("jk_hidden_channel", 1, 700)
linear_hidden_channel = trial.suggest_int("linear_hidden_channel", 1, 700)

dropout = trial.suggest_float("dropout", 0,0.5)
use_jump = trial.suggest_categorical("use_jump", [True,False])
batch_size = trial.suggest_int("Batch size", 1, 200)
```

```python
optimizer = torch.optim.Adam(
    model.parameters(), lr=trial.suggest_float("lr", 1e-5, 1e-1)
)
```

Result from Hyperparamater optimsation





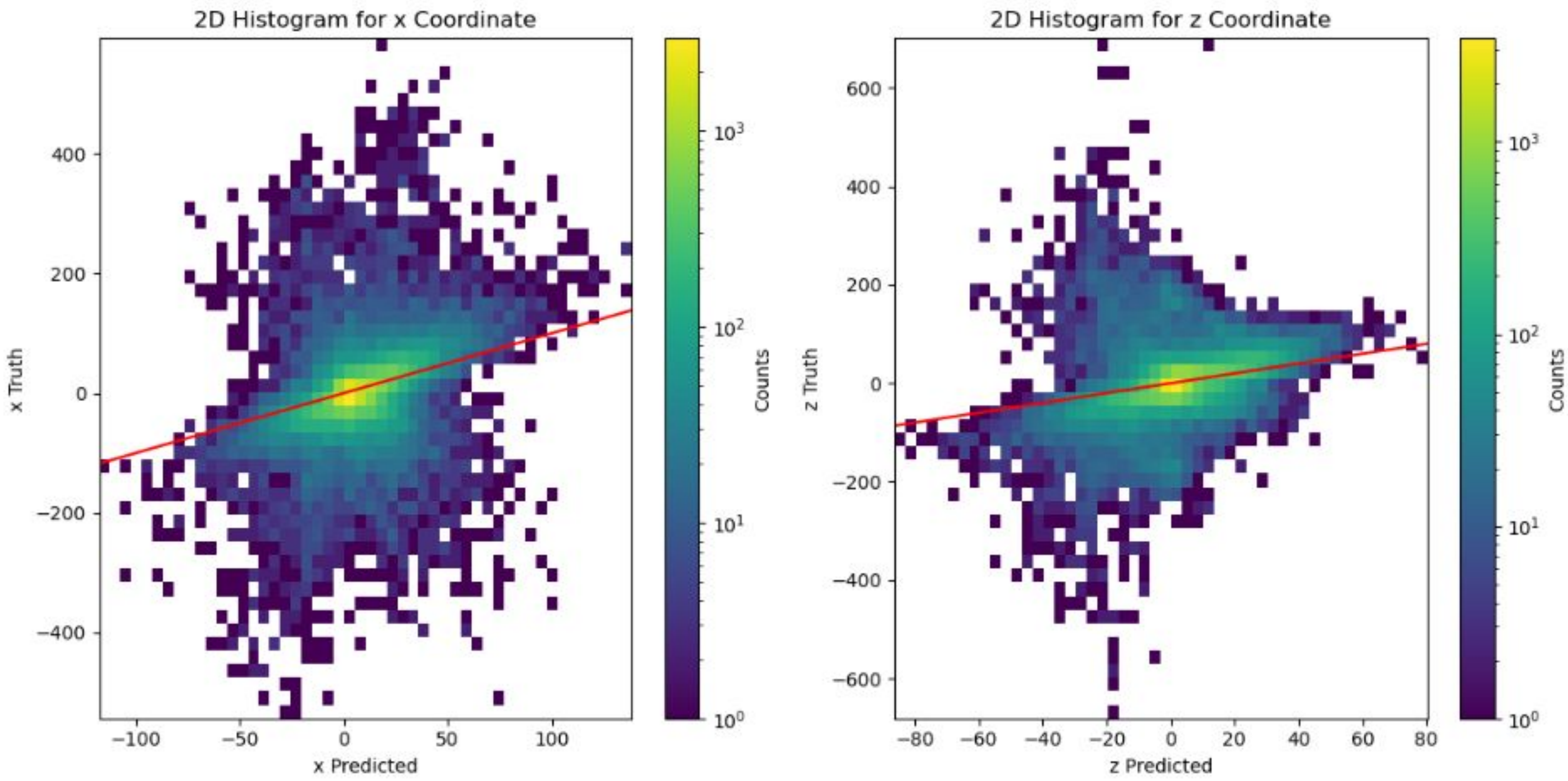
Best Trial (number=2)

47.534751811591065

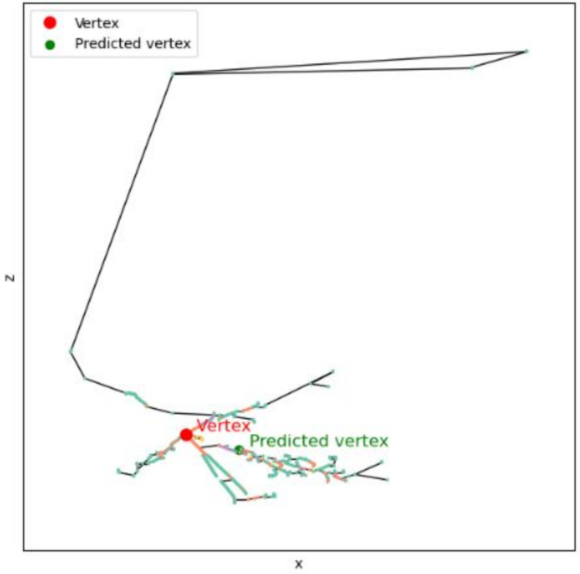
Params = [num_layers: 6, number_layers_linear: 2, hidden_channels: 239, dropout: 0.20688802612037932, use_jump: False, Batch size: 80, lr: 0.008976107995591361]

DETAILS

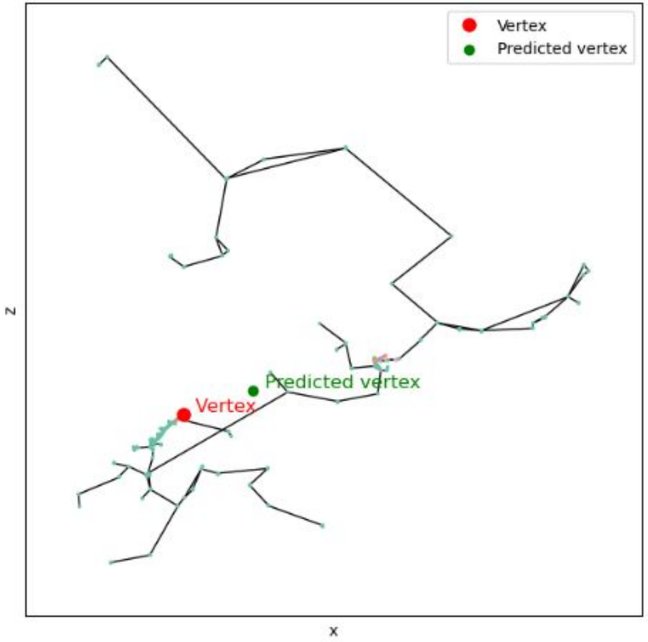
Trained with 100k atm cc+nc event



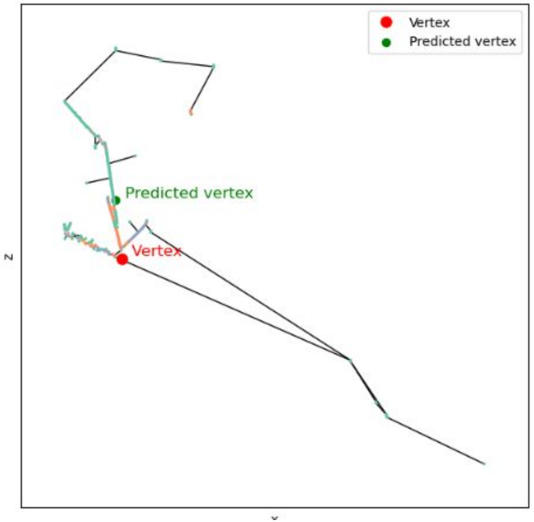
Many events with vertex at the center



Vertex
Predicted vertex
z
x



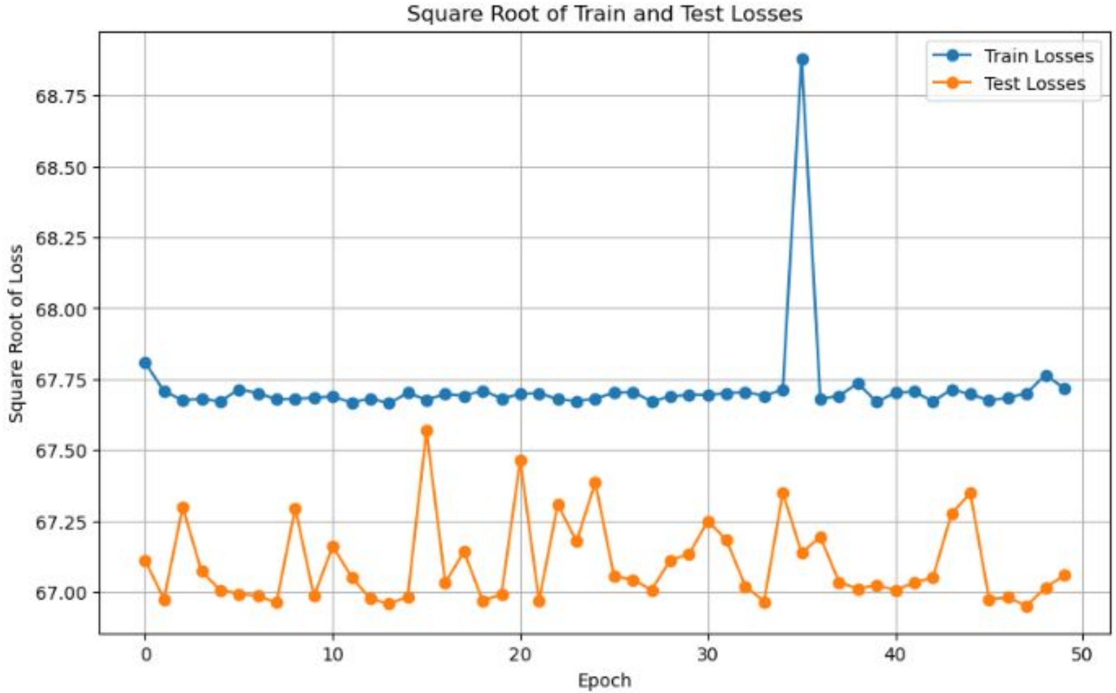
Vertex
Predicted vertex
z
x



Vertex
Predicted vertex
z

Oversmoothing?

num_layers 4
number_layers_linear 4
hidden_channels 70
jk_hidden_channel 66
linear_hidden_channel 67
dropout 0.012672886467014212

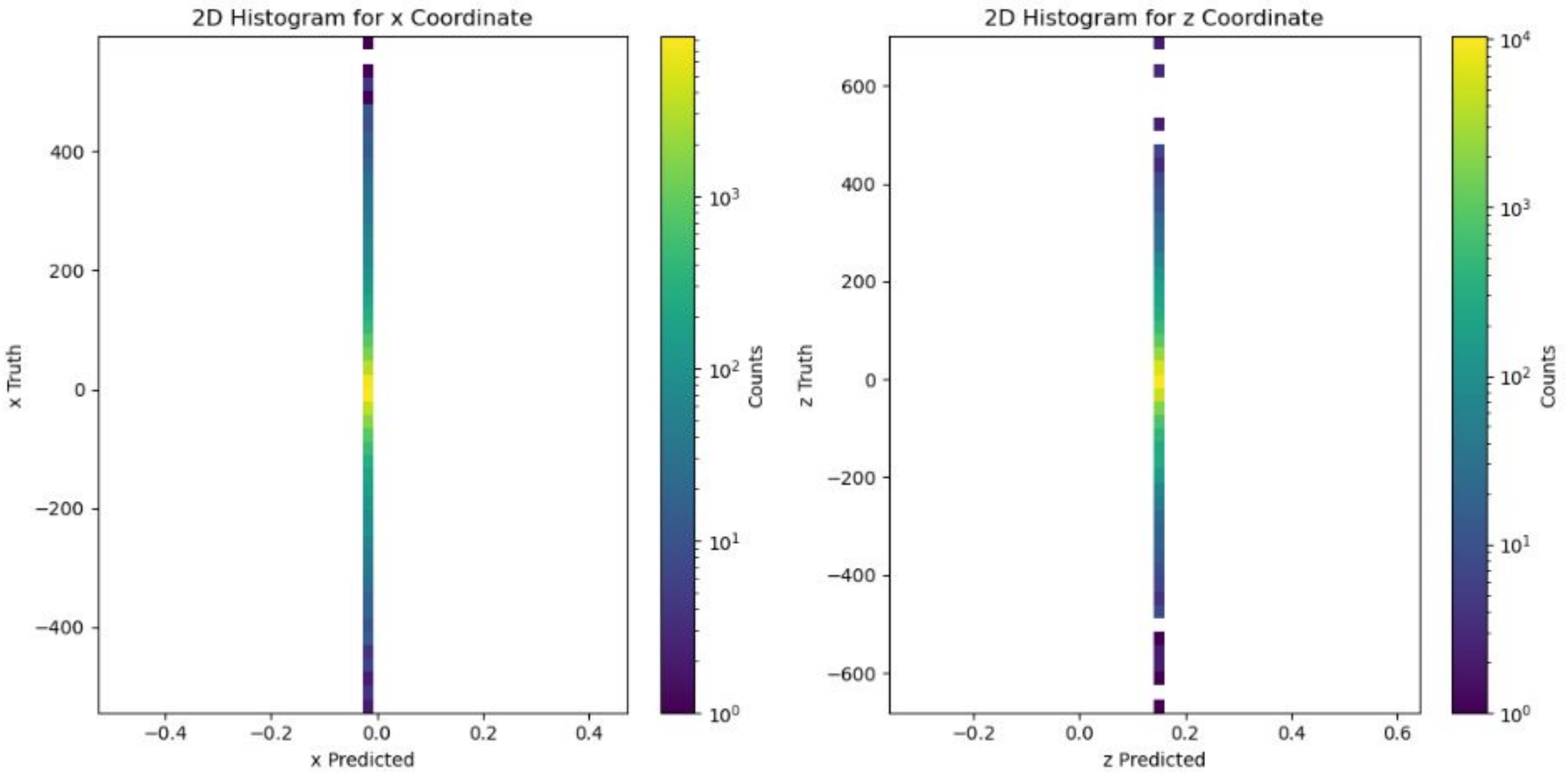



```
The current epoch is 0
The prediction is  tensor([[-0.2695,  0.3886],
        [-0.2695,  0.3886],
        [-0.2695,  0.3886],
        [-0.2695,  0.3886],
        [-0.2695,  0.3886]], device='cuda:0')
The truth vtx is tensor([[ 3.0200e+01, -4.3819e+01],
        [-5.2322e-02,  2.6323e+01],
        [-4.6409e+01, -7.1436e+01],
        [-5.3210e+00,  1.9788e+01],
        [ 1.2981e+01,  2.4321e+02]], device='cuda:0')
The current epoch is 1
The prediction is  tensor([[0.2485, 0.2952],
        [0.2485, 0.2952],
        [0.2485, 0.2952],
        [0.2485, 0.2952],
        [0.2485, 0.2952]], device='cuda:0')
The truth vtx is tensor([[ -7.1847, 134.2458],
        [ 58.7638,  29.5841],
        [-20.7795, -18.8505],
        [-12.5556,  19.4468],
        [  4.3935, -16.1088]], device='cuda:0')
The current epoch is 2
The prediction is  tensor([[0.2259, 0.3906],
        [0.2259, 0.3906],
        [0.2259, 0.3906],
        [0.2259, 0.3906],
        [0.2259, 0.3906]], device='cuda:0')
The truth vtx is tensor([[ 20.4032, -16.2834],
        [ 18.8467,  20.0847],
        [ 46.9897, -24.2996],
        [ 10.5652,   7.9223],
        [-30.8546,  29.1590]], device='cuda:0')
The current epoch is 3
```

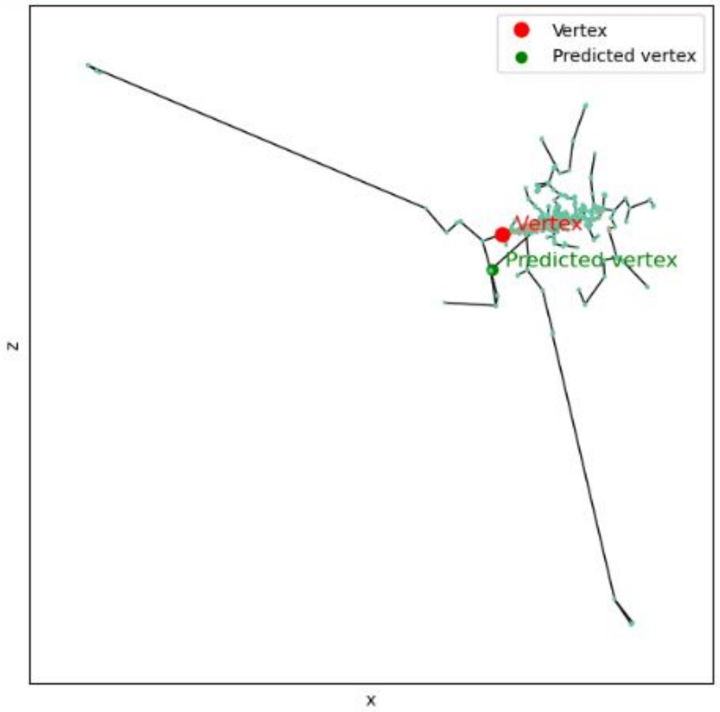
The prediction of the model is same for different input

The model not learning and treat every graph as the same

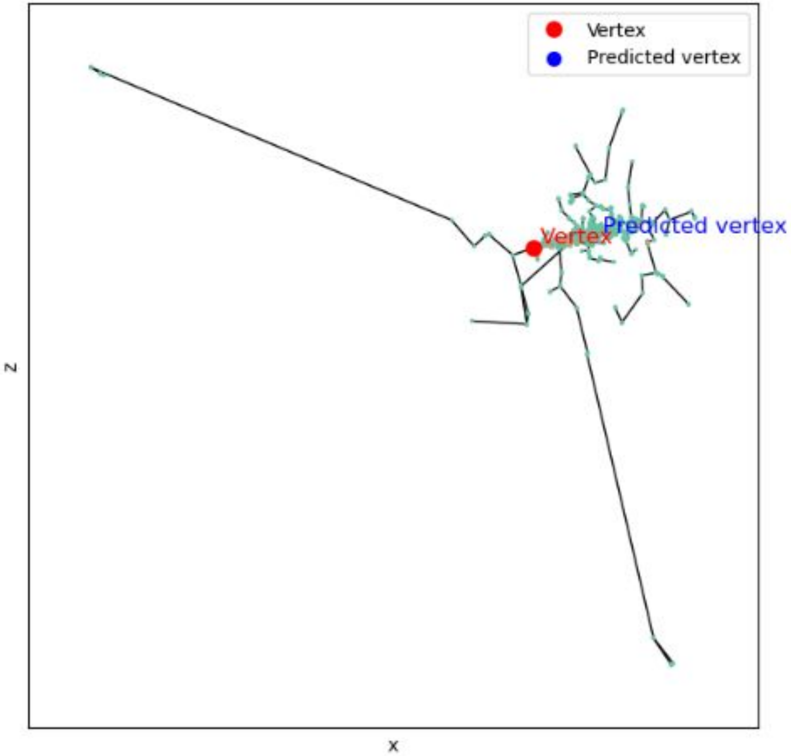
It happens a lot (80 out of 100)



2D Histogram for x Coordinate
x Truth
x Predicted
Counts
2D Histogram for z Coordinate
z Truth
z Predicted
Counts



From best model



From over-smoothing model

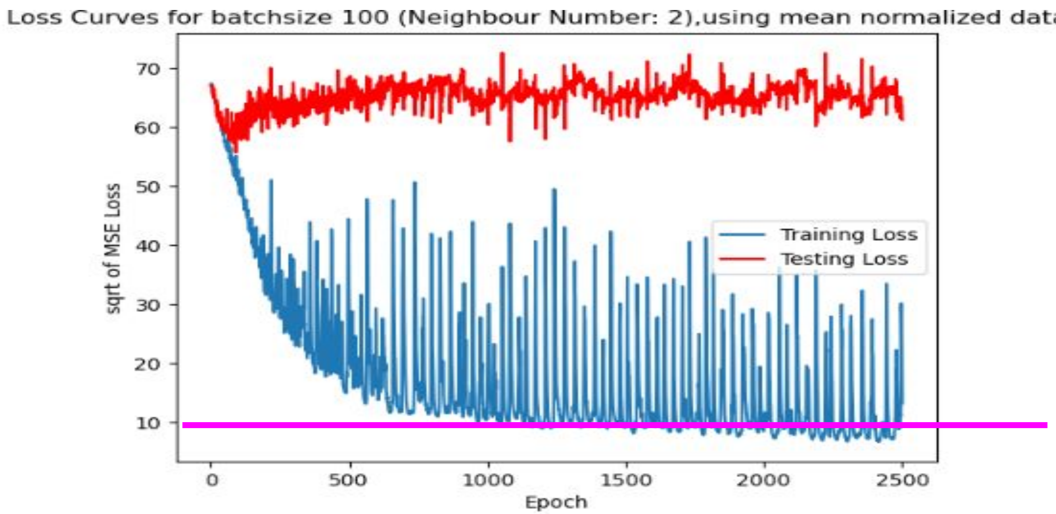
Questions:

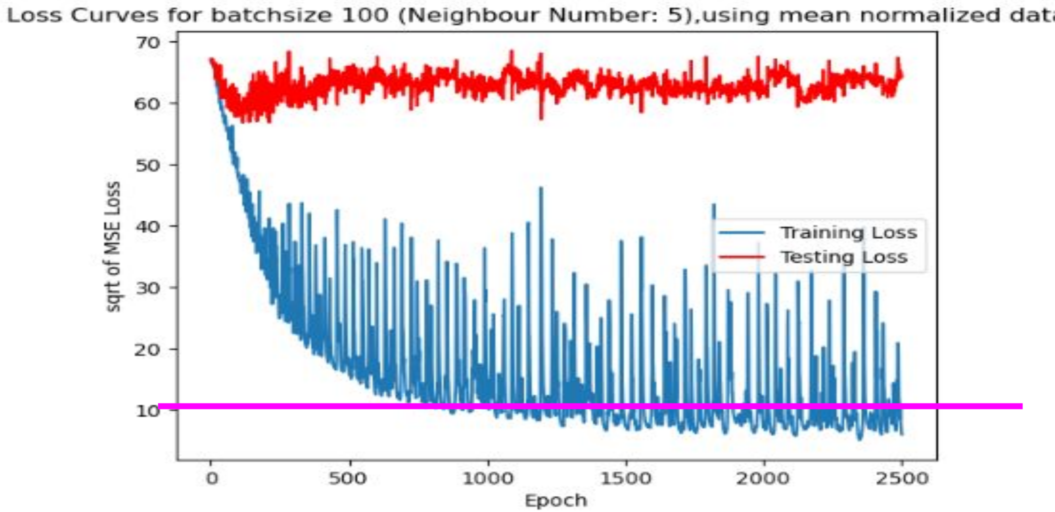
1.What can I do to improve the model?

Few things going to try:

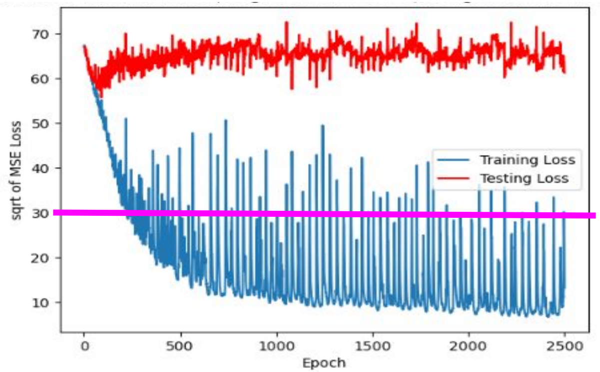
- Use different agg method for graphSage?
- Use different readout layer after Message Passing Layers?
- Try a simpler sample?
- Instead of training a regression model, try to train the model to learn distance class from truth vtx (like CVN vertexing)

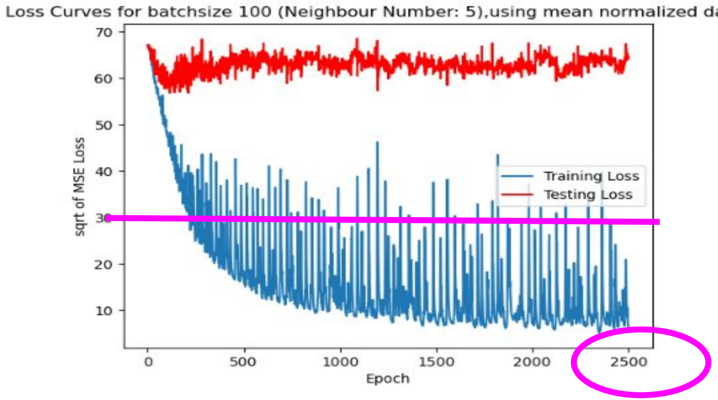
BackUp

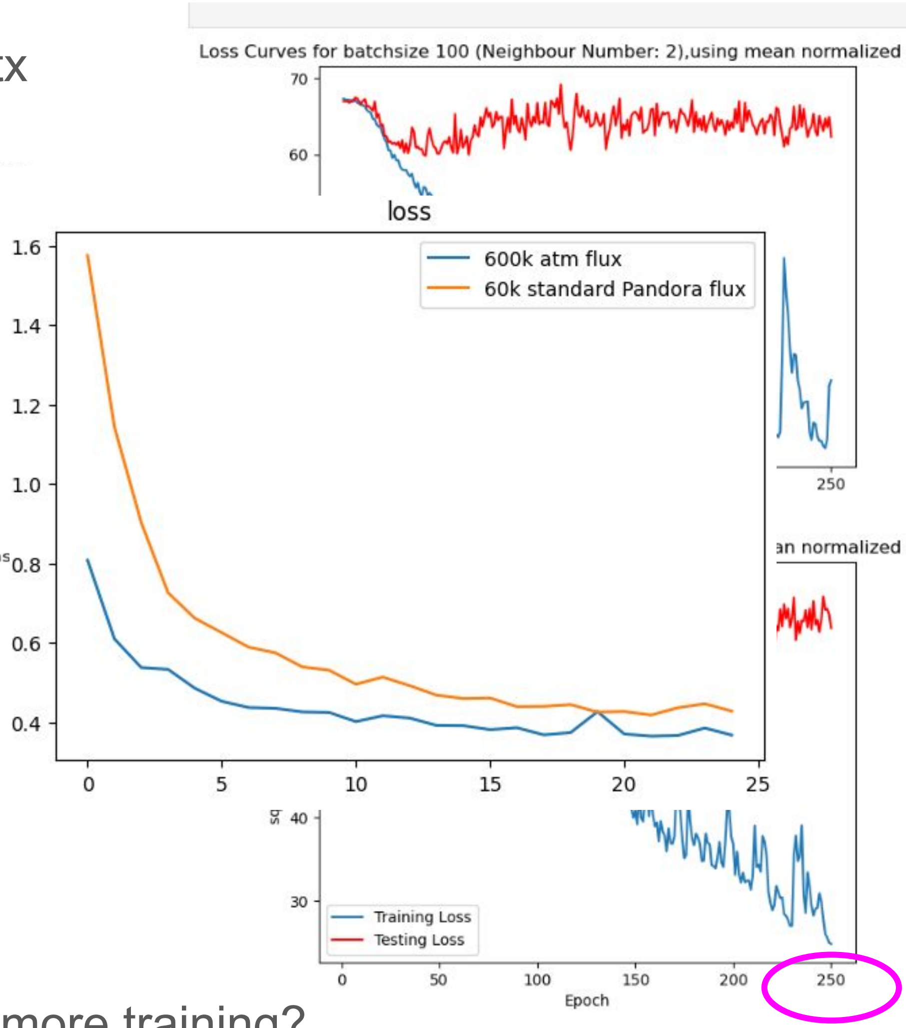

Loss Curves for batchsize 100 (Neighbour Number: 2),using mean normalized dataset
sqrt of MSE Loss
70
60
50
40
30
20
10
0
500
1000
1500
2000
2500
Epoch
Training Loss
Testing Loss


Loss Curves for batchsize 100 (Neighbour Number: 5),using mean normalized dataset
sqrt of MSE Loss
70
60
50
40
30
20
10
0
500
1000
1500
2000
2500
Epoch
Training Loss
Testing Loss

Node connection method; y-axis: vtx resolution







+ Huge Overfitting
+ Poor resolution → solved w/ more training?

Reproducibility: