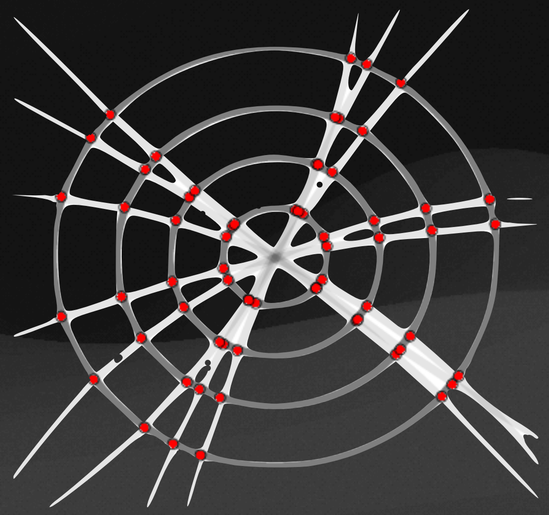
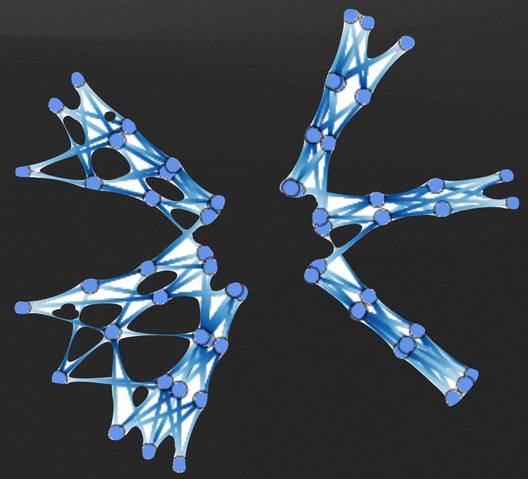
Particle Tracking with a Graph Neural Network

Group 3: Jason Weitz, Anthony Aportela, Dmitri Demler

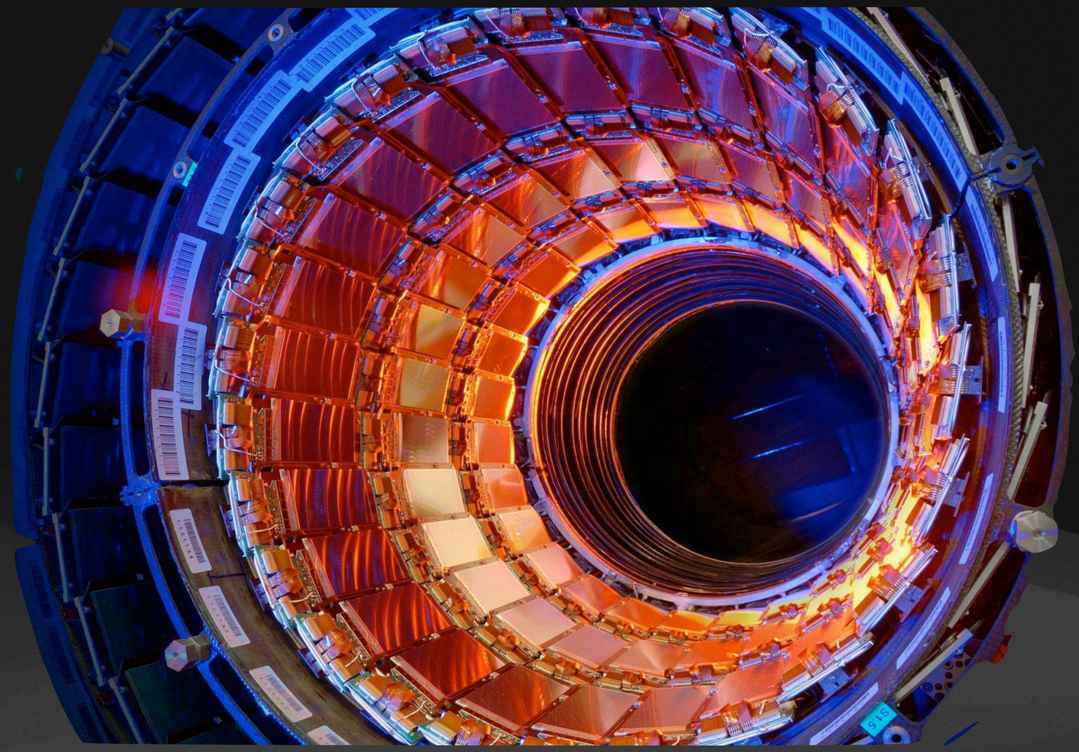
Overview

- Reconstruct particle paths (from LHC)
- Use of the TrackML Kaggle dataset
- Data Preprocessing
- Creating a Graph Neural Network
- Comparison to a baseline
- Results





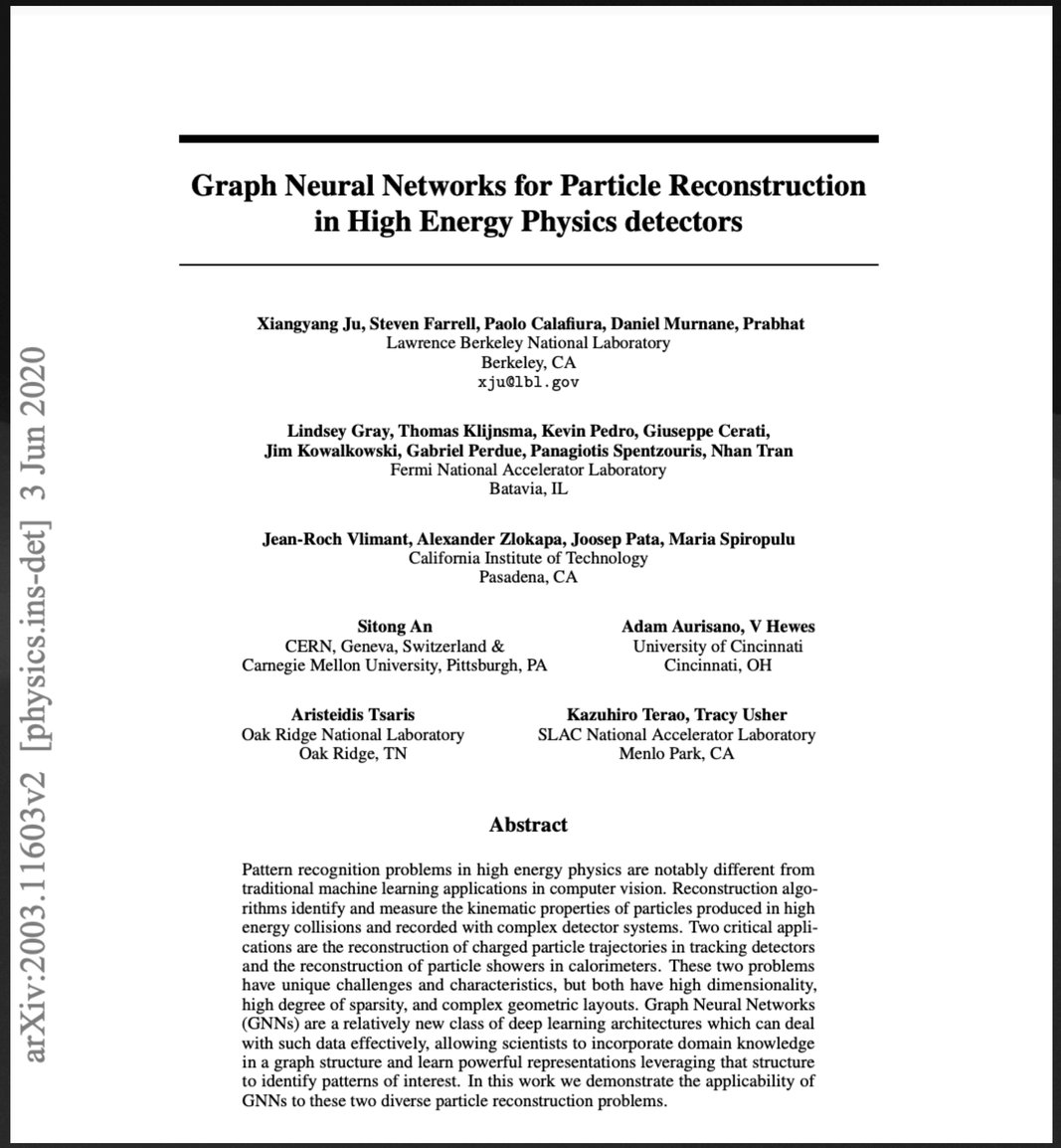
Our task



- Reconstruct Cern LHC collision events using graph neural networks
- Collection of too much info
- Need methods to discard what’s unnecessary
- Process of track reconstruction using ML

The Paper

- Challenges in determining particle path
- Traditional approach: Kalman Filter
  - Poor scaling
- Use of GNNs as a solution
  - Maintain accuracy, improve efficiency at inference



Graph Neural Networks for Particle Reconstruction in High Energy Physics detectors

Xiangyang Ju, Steven Farrell, Paolo Calafiura, Daniel Murnane, Prabhat
Lawrence Berkeley National Laboratory
Berkeley, CA
xju@lbl.gov

Lindsey Gray, Thomas Klijnsma, Kevin Pedro, Giuseppe Cerati, Jim Kowalkowski, Gabriel Perdue, Panagiotis Spentzouris, Nhan Tran
Fermi National Accelerator Laboratory
Batavia, IL

Jean-Roch Vlimant, Alexander Zlokapa, Joosep Pata, Maria Spiropulu
California Institute of Technology
Pasadena, CA

Sitong An
CERN, Geneva, Switzerland &
Carnegie Mellon University, Pittsburgh, PA

Adam Aurisano, V Hewes
University of Cincinnati
Cincinnati, OH

Aristeidis Tsaris
Oak Ridge National Laboratory
Oak Ridge, TN

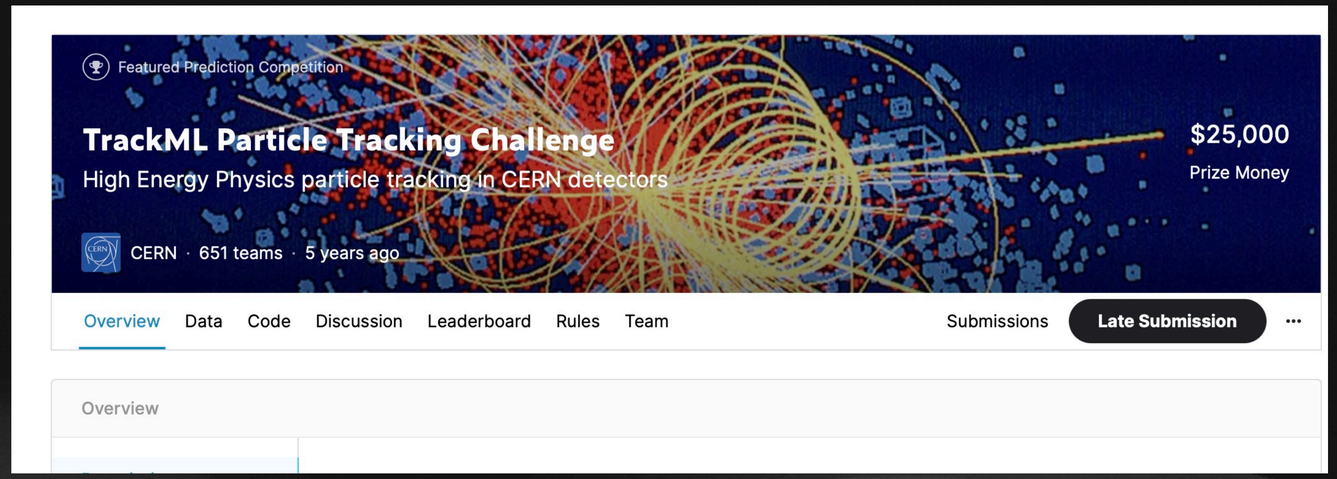
Kazuhiro Terao, Tracy Usher
SLAC National Accelerator Laboratory
Menlo Park, CA

Abstract

Pattern recognition problems in high energy physics are notably different from traditional machine learning applications in computer vision. Reconstruction algorithms identify and measure the kinematic properties of particles produced in high energy collisions and recorded with complex detector systems. Two critical applications are the reconstruction of charged particle trajectories in tracking detectors and the reconstruction of particle showers in calorimeters. These two problems have unique challenges and characteristics, but both have high dimensionality, high degree of sparsity, and complex geometric layouts. Graph Neural Networks (GNNs) are a relatively new class of deep learning architectures which can deal with such data effectively, allowing scientists to incorporate domain knowledge in a graph structure and learn powerful representations leveraging that structure to identify patterns of interest. In this work we demonstrate the applicability of GNNs to these two diverse particle reconstruction problems.

Ju, Xiangyang, et al. "Graph neural networks for particle reconstruction in high energy physics detectors." arXiv preprint arXiv:2003.11603 (2020).

Dataset



- TrackML dataset
- X, Y, Z coordinates of hits in the calorimeter, Volume, etc.
- Steps taken for data preprocessing/ PCA
  - Conversion to η, ϕ, and Z
  - Partition detector space into segments

$$\eta = -\ln(\tan(\frac{\Theta}{2}))$$

$$\phi = \arctan(\frac{y}{x})$$

Ju, Xiangyang, et al. "Graph neural networks for particle reconstruction in high energy physics detectors." arXiv preprint arXiv:2003.11603 (2020).

Coordinate Transformation

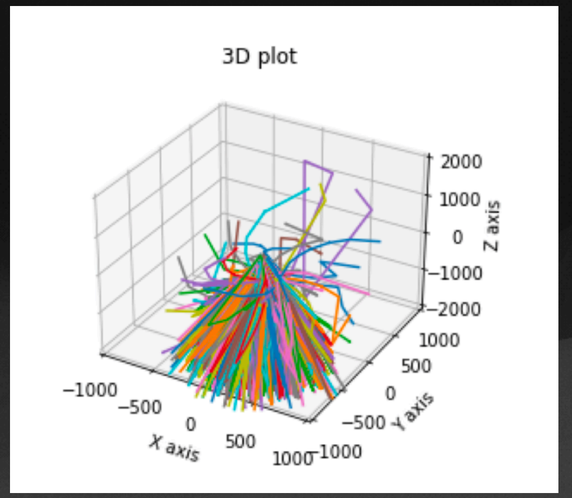
And Splitting Sectors

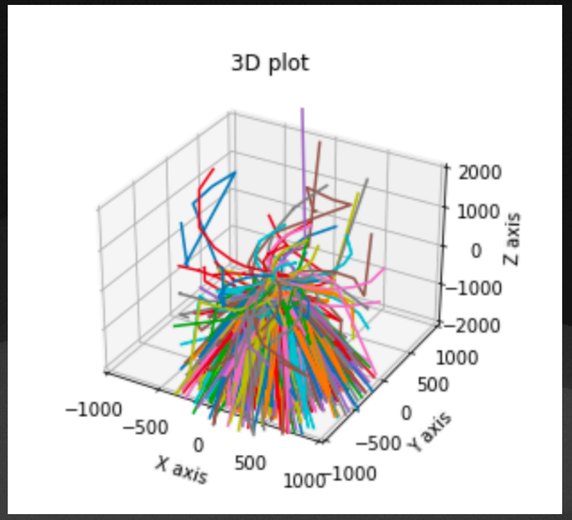
- Steps to transform coordinates
- Split into sectors in order to handle the large data
    - Biasing for higher energy particles

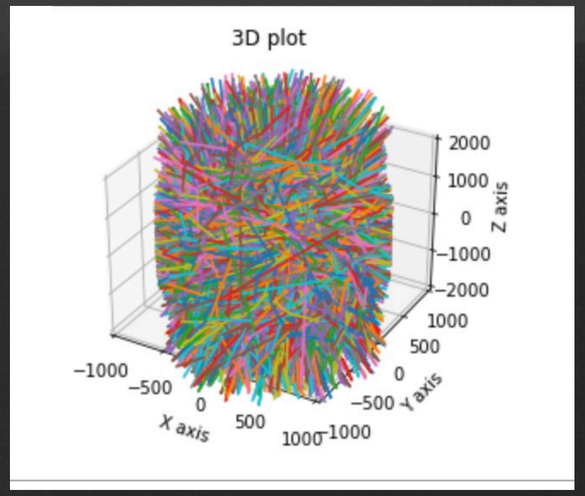
```python
def coord_convert(x,y,z):

    phi = np.arctan2(y, x)
    theta = np.arctan2(np.sqrt(x ** 2 + y ** 2), z)
    eta = -np.log(np.tan(theta/2))
    return phi, eta, z


def sector_splitter(df, n, m):

    phi_bins = np.linspace(-np.pi, np.pi, n+1)  # bins for phi angles
    eta_bins = np.linspace(-4.5, 4.5, m+1)  # bins for eta angles

    df_list = []
    for i in range(n):
        for j in range(m):

            phi_mask = (phi_bins[i] < df['phi']) & (df['phi'] < phi_bins[i+1])
            eta_mask = (eta_bins[j] < df['eta']) & (df['eta'] < eta_bins[j+1])

            df_list.append(df[(phi_mask & eta_mask)])

    return df_list
```

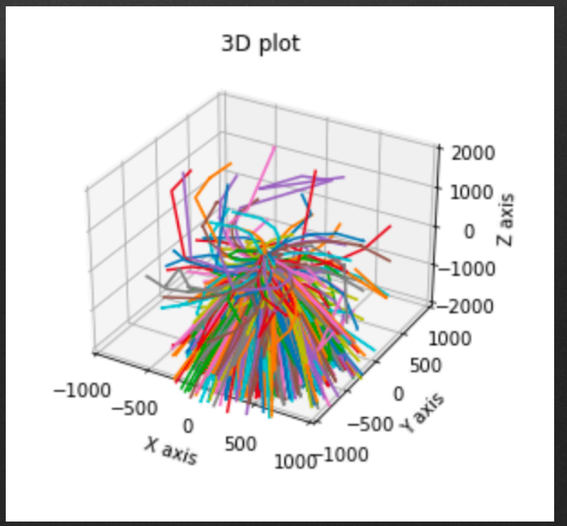
Data Visualization

- Reconstructed particle paths
- Known particle IDs indicated by color
- Each plot is a different event
  - Dataset is spatially coordinated
  - Need to randomize
- 1 event ~ 100,000 hits





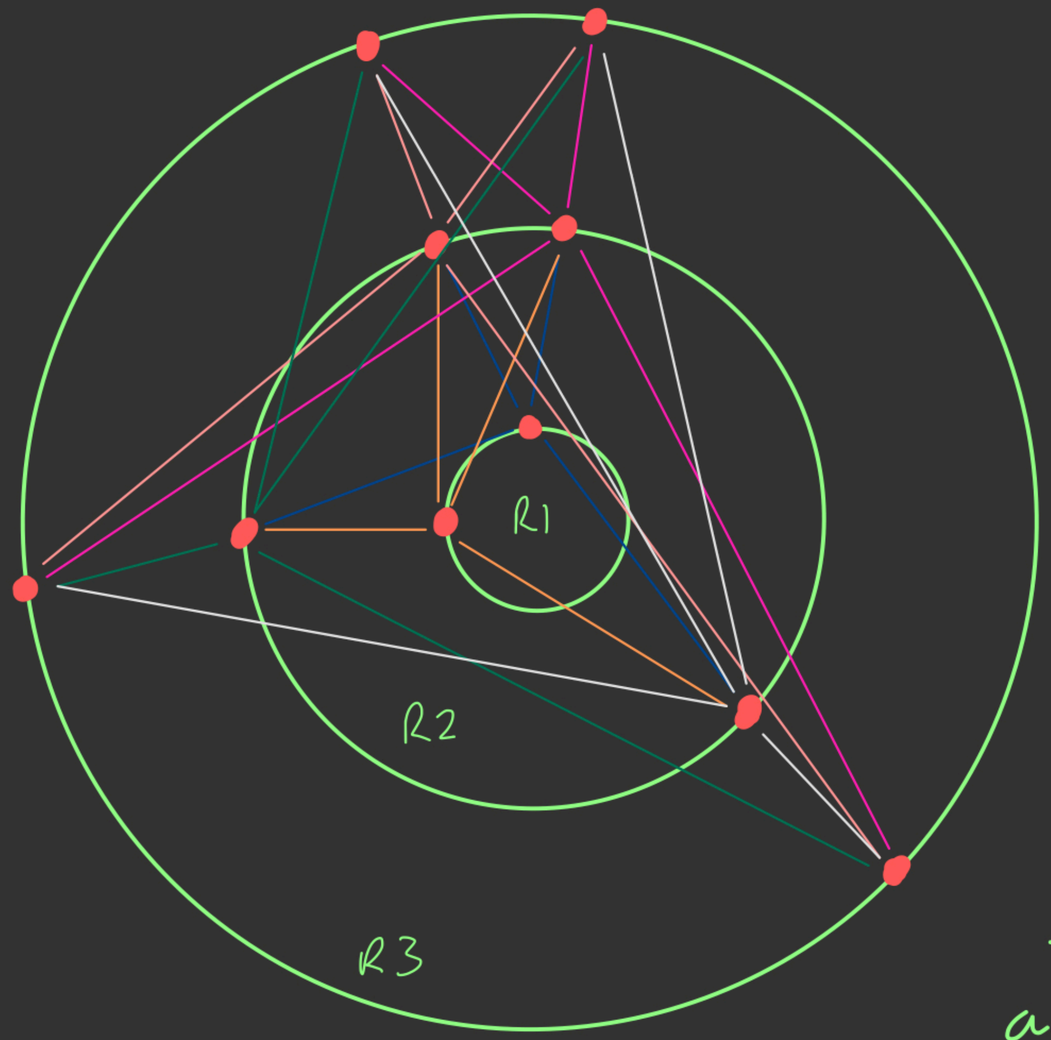


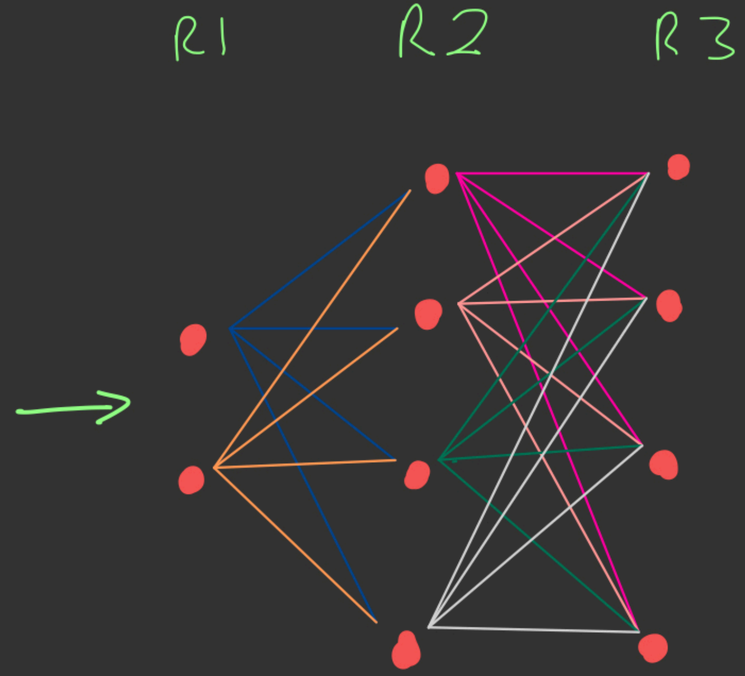


Data Preprocessing

- Create a point cloud object with all the data
- Create a PyTorch geometric graph object
  - Each hit must be connected to every other node $= \Theta(n^2)$
  - With 100,000 hits per event, number of edges is 10^{10}
  - To solve: remove the mathematically impossible edge connects (remove dR>1.7)

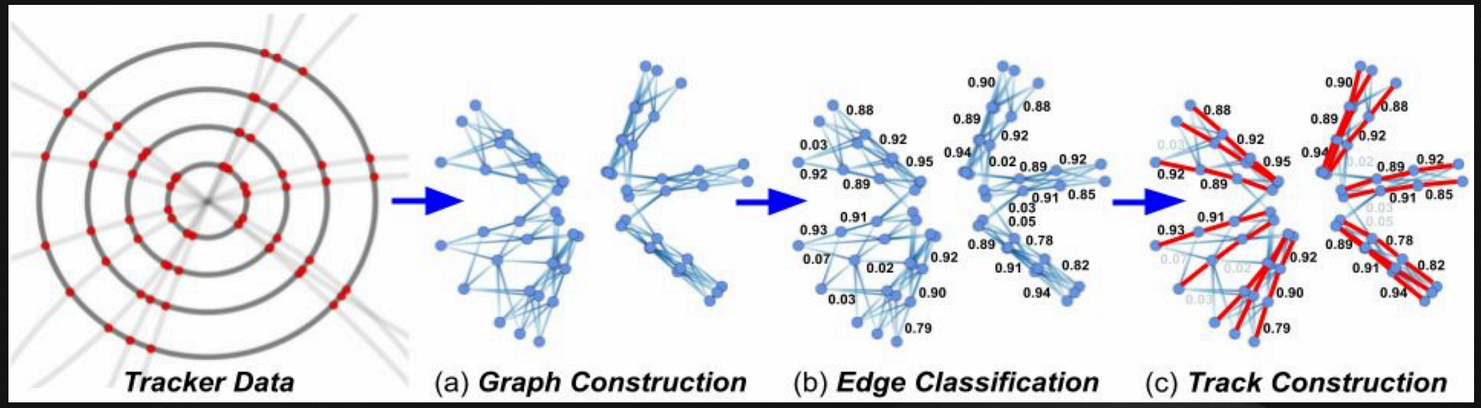
$$dR = \sqrt{d\phi^2 + d\eta^2}$$





The graph is a complete r-partite graph, where the the nodes are labeled by (r, z, ϕ) and the edges are labeled by $(\Delta\eta, \Delta\phi)$ and whether or not the two nodes are are from the same track.

Methods



- Implementation of a Graph Neural Network
  - Edge classifier
- Two attempts
  - Create a usable GNN from scratch
  - Implementation of similar method from paper
- Facing some challenges with our original method

Creating and Training

- Basic idea is the same for both
- import spacial information (x,y,z) and convert to (phi, eta, z)
- convert into graphs using a CustomDataset class
- define a model, loss, and optimizer
- iterate and reduce loss

```
dataframes = data_puller(1)

# Split into train and test sets
train_dfs, test_dfs = train_test_split(dataframes, test_size=0.2)

# Create custom datasets and dataloaders
train_dataset = CustomDataset(train_dfs)
test_dataset = CustomDataset(test_dfs)
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)
```

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")


# Define the parameters of the model
# number of features per node
input_dim = 3
#size of hidden layer
hidden_dim = 64
#size of output
output_dim = 1
# number hidden layers
num_layers = 1

# Create the PFN model
model = GNN(input_dim, hidden_dim, output_dim, num_layers)
# model = model.to(device)

# Define the loss function and optimizer
criterion = nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
```

```
# Train the model
for epoch in range(100):
    total_loss = 0.0
    for data in train_loader:
        optimizer.zero_grad()
        output = model(data)
        loss = criterion(output, data.y)
        loss.backward()
        optimizer.step()
        total_loss += loss.item()
    print("Epoch %d, Loss = %f" % (epoch+1, total_loss/len(dataloader)))
```

The model

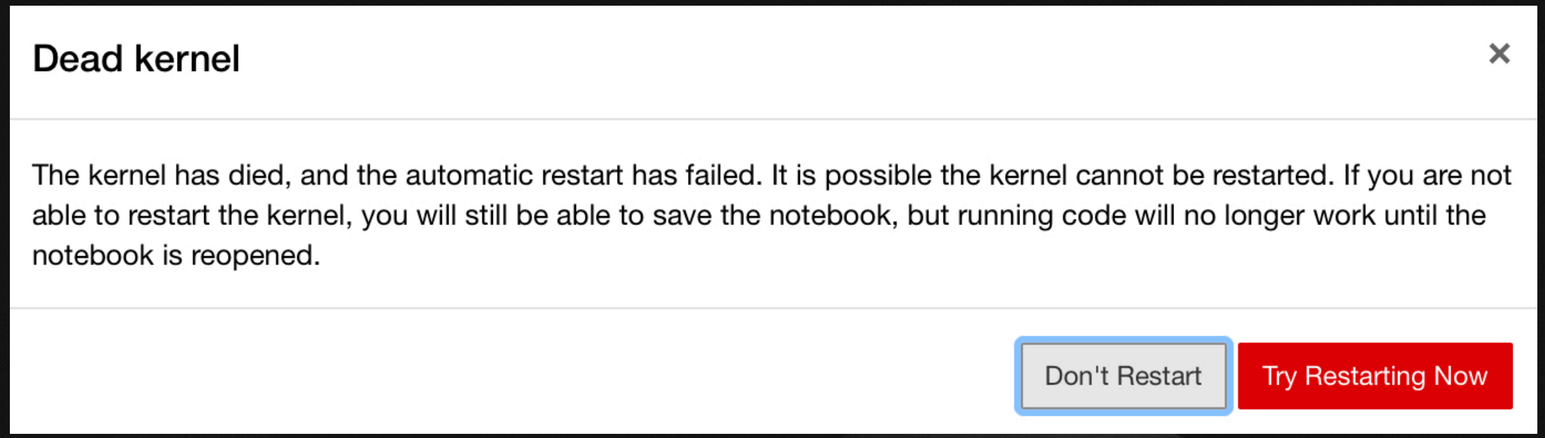
Two main methods (theirs)

- __init__() which defines the architecture
- forward() defines how operations between layers are preformed

- still trying to fully understand their architecture

```python
class EdgeClassifier(nn.Module):
    def __init__(
        self,
        node_indim,
        edge_indim,
        L=4,
        node_latentdim=8,
        edge_latentdim=12,
        r_hidden_size=32,
        o_hidden_size=32,
    ):
        super().__init__()
        self.node_encoder = MLP(node_indim, node_latentdim, 64, L=1)
        self.edge_encoder = MLP(edge_indim, edge_latentdim, 64, L=1)
        gnn_layers = []
        for _l in range(L):
            # fixme: Wrong parameters?
            gnn_layers.append(
                IN(
                    node_latentdim,
                    edge_latentdim,
                    node_outdim=node_latentdim,
                    edge_outdim=edge_latentdim,
                    edge_hidden_dim=r_hidden_size,
                    node_hidden_dim=o_hidden_size,
                )
            )
        self.gnn_layers = nn.ModuleList(gnn_layers)
        self.W = MLP(edge_latentdim, 1, 32, L=2)

    def forward(self, x: Tensor, edge_index: Tensor, edge_attr: Tensor) -> Tensor:
        node_latent = self.node_encoder(x)
        edge_latent = self.edge_encoder(edge_attr)
        for layer in self.gnn_layers:
            node_latent, edge_latent = layer(node_latent, edge_index, edge_latent)
        edge_weights = torch.sigmoid(self.W(edge_latent))
        return edge_weights
```

Our GNN



- Similar distilled version
- Running into memory issues and/or multiple device issues
- Probably because our graphs are ~25Mb a pop

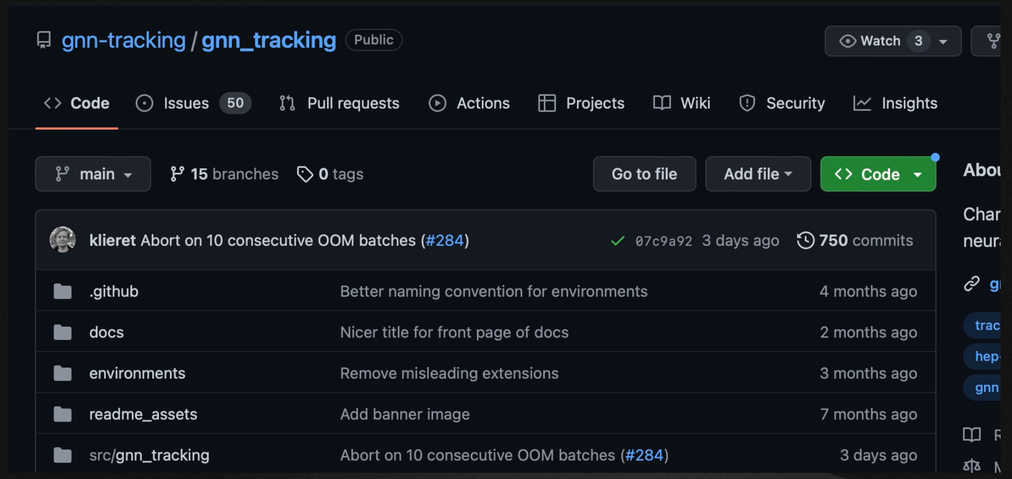
- Possible problem might be size of imported dataframes
- solution could be making it so graph is generated lazily from directory

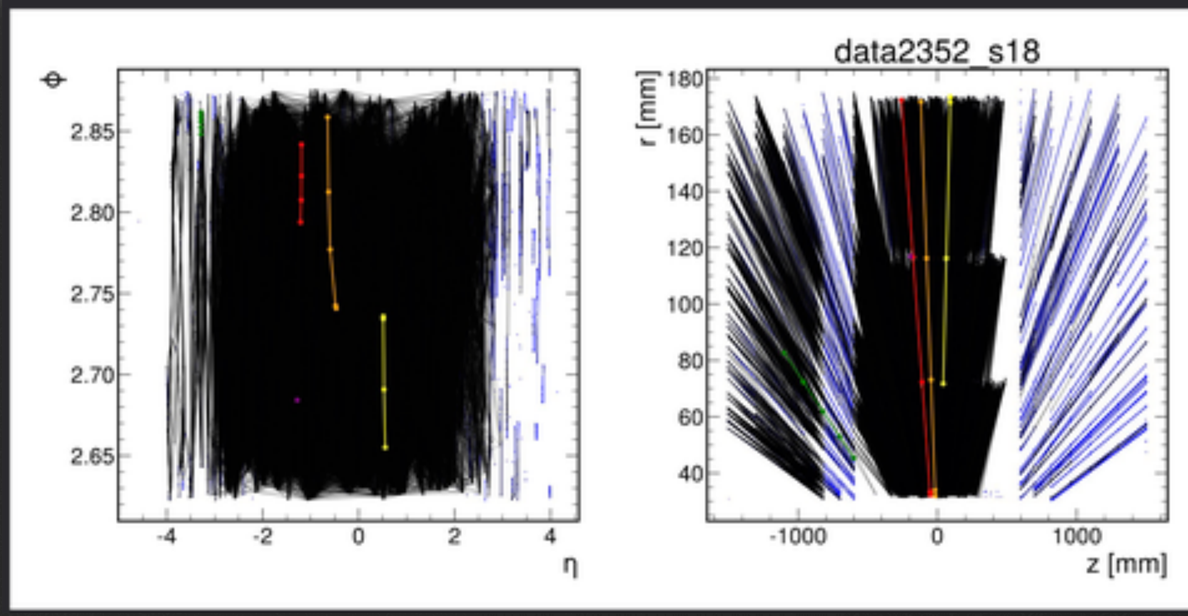
```
RuntimeError: CUDA out of memory. Tried to allocate 22.68 GiB (GPU 0; 10.92 GiB total capacity; 3.60 GiB alread
y allocated; 6.44 GiB free; 3.64 GiB reserved in total by PyTorch)
```

```
RuntimeError: Expected all tensors to be on the same device, but found at least two devices, cuda:0 and cpu! (w
hen checking arugment for argument mat2 in method wrapper_mm)
```

Other GNN

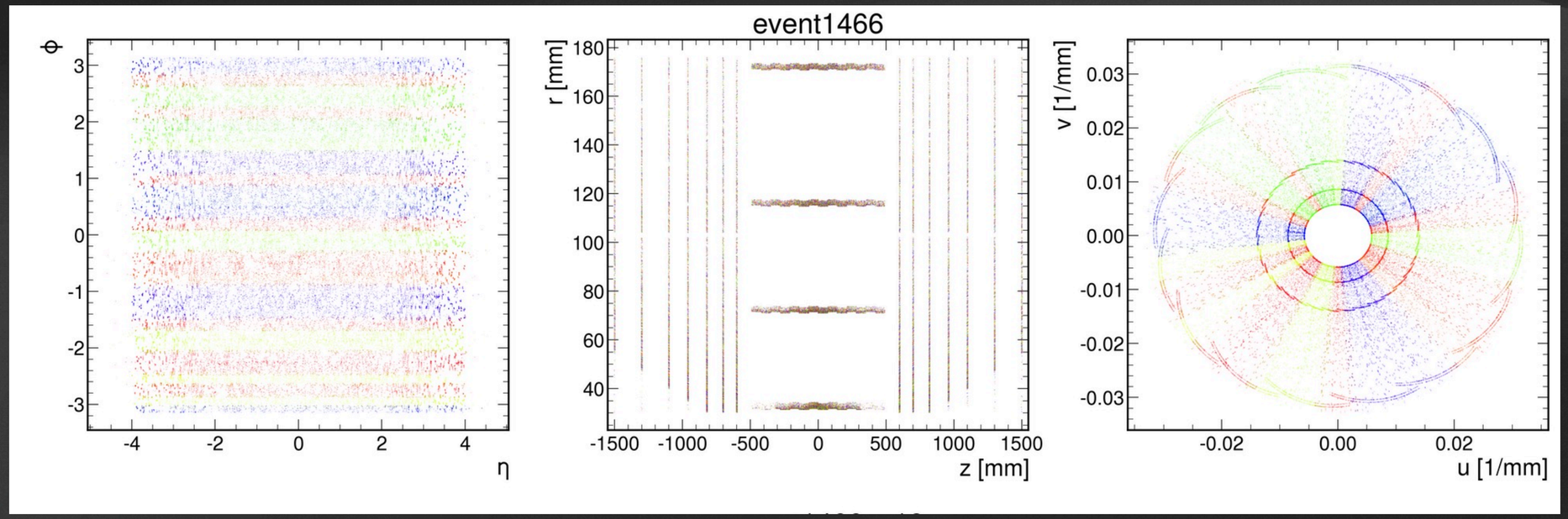
- 10-Layer Graph Temporal Convolutional Network (graphTCN)
- 10 Layers containing:
    - Node encoder
    - Edge encoder
    - Interaction Network
- More efficient graph creation
    - 100 kb per graph (instead of 25mb)





Other GNN

- Sector split different from our implementation

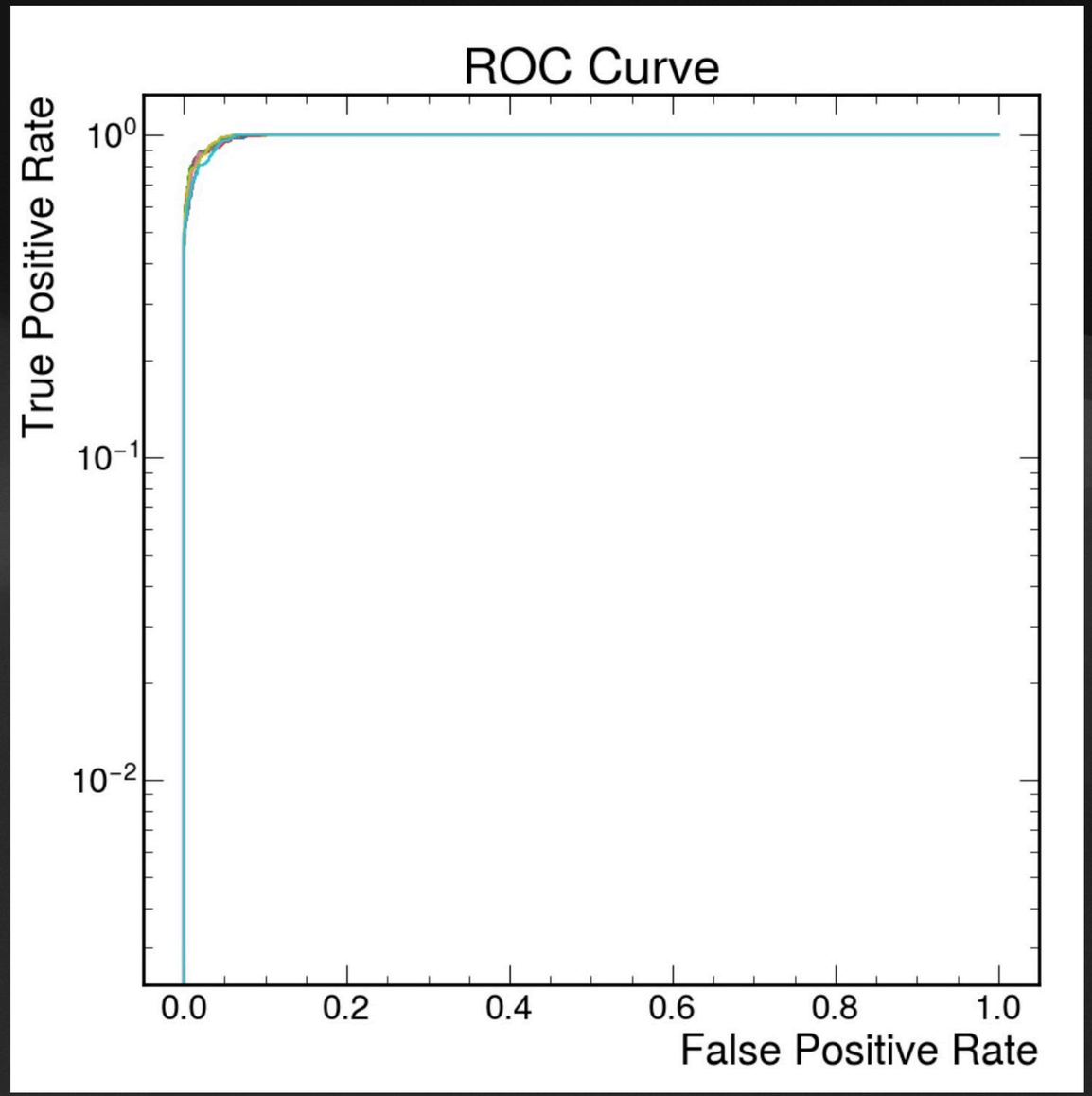


Point Cloud transformation

Results
Other GNN

- Trained on 70 Events in ~40 mins
- 96% accuracy
- 0.97 ROC-AUC
- After training, 0.5 seconds / event

ROC taken every epoch



Baseline Method Comparison

- Combinatoric geometric algorithm
  - Creation of a candidate list
  - Truncate the list
  - Extends candidates
- Not much ML implementation
  - Training data to calculate stats
- Accuracy ~ 79%
- 30 seconds / event

```
event_test: event 1000: open, #hits=120939, #particles=12263, #truth=120939
event 1000:
    round 0 (120939 hits):
    done round 0 (120939 hits): 37.824s
done event 1000: 37.825s

event 1000 score: 0.7876
event_test: event 1001: open, #hits=93680, #particles=8915, #truth=93680
event 1001:
    round 0 ( 93680 hits):
    done round 0 ( 93680 hits): 21.027s
done event 1001: 21.027s

event 1001 score: 0.7941
event_test: event 1002: open, #hits=125504, #particles=12763, #truth=125504
event 1002:
    round 0 (125504 hits):
    done round 0 (125504 hits): 42.315s
done event 1002: 42.315s

event 1002 score: 0.7881
event_test: event 1003: open, #hits=104451, #particles=10261, #truth=104451
event 1003:
    round 0 (104451 hits):
    done round 0 (104451 hits): 25.342s
done event 1003: 25.343s

event 1003 score: 0.7912
score mean 0.7903 (std 0.0026)
Time taken: 128.972s
```

Next Steps

- Our method in progress
- Currently facing some challenges
- Future of GNNs for particle track reconstruction