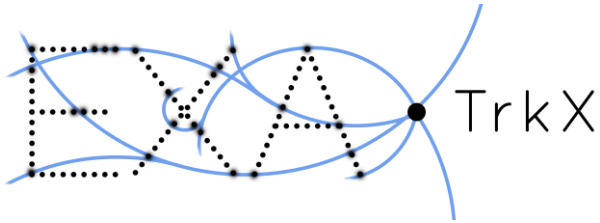
Graph Neural Networks for Track Finding

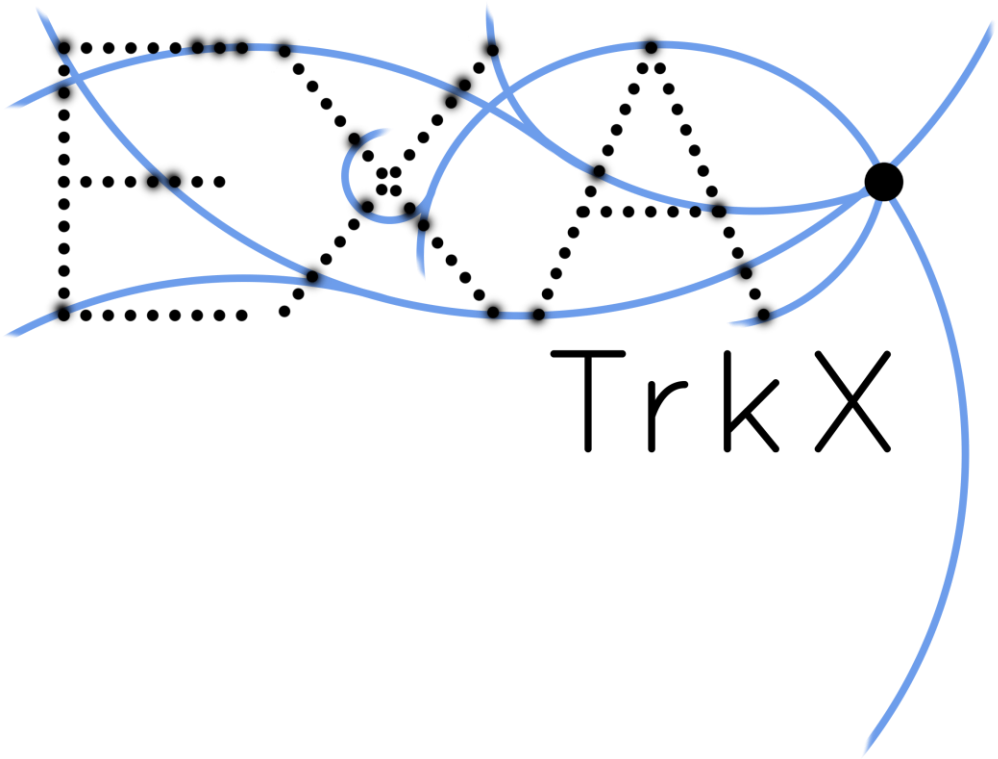
Daniel Murnane

Exa.TrkX @ Berkeley Lab



Connecting The Dots 2020
Princeton University

The Exa.TrkX Project



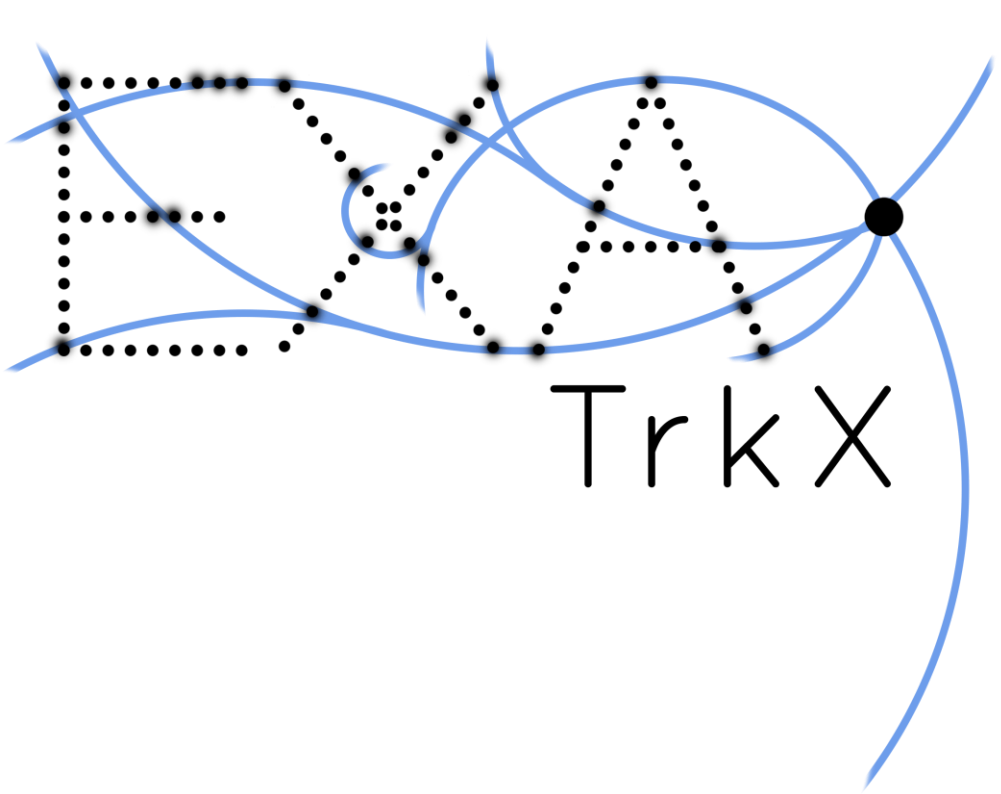
MISSION

Optimization, performance and validation studies of ML approaches to the Exascale tracking problem, to enable production-level tracking on next-generation detector systems.

PEOPLE

- *Caltech*: Joosep Pata, Maria Spiropulu, Jean-Roch Vlimant, Alexander Zlokapa
- *Cincinnati*: Adam Aurisano, Jeremy Hewes
- *FNAL*: Giuseppe Cerati, Lindsey Gray, Thomas Klijnsma, Jim Kowalkowski, Gabriel Perdue, Panagiotis Spentzouris
- *LBNL*: Paolo Calafiura (PI), Nicholas Choma, Sean Conlon, Steve Farrell, Xiangyang Ju, Daniel Murnane, Prabhat
- *ORNL*: Aristeidis Tsaris
- *Princeton*: Isobel Ojalvo, Savannah Thais
- *SLAC*: Pierre Cote De Soux, Francois Drielsma, Kasuhiro Terao, Tracy Usher

The Exa.TrkX Project



TRACK FINDING GOAL

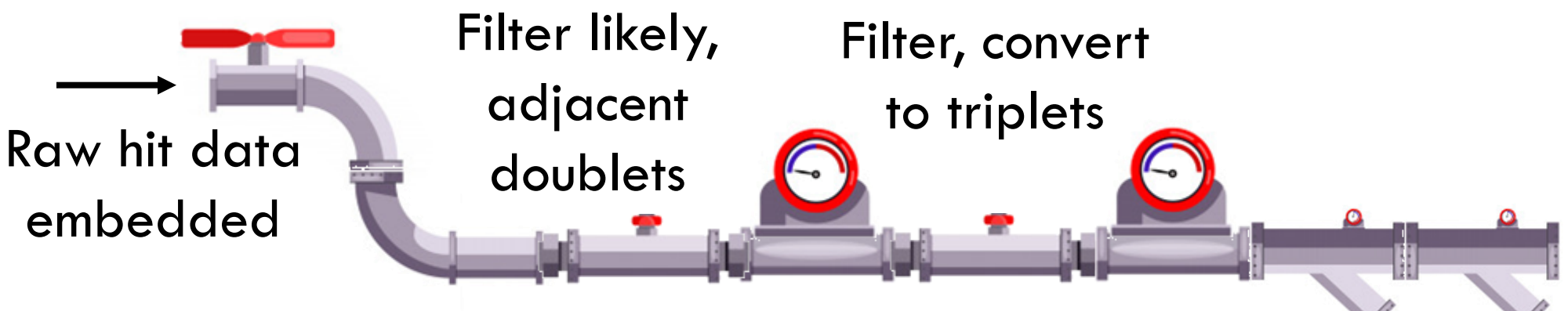
Given that graphs are a natural way to represent tracks, use learned embeddings and graph neural networks for sub-second processing of HL-LHC event data into:

Seeds (i.e. triplets) for further processing with traditional techniques,

AND/OR

Tracks, where each hit is assigned to exactly one track

Track Finding Pipeline



Raw hit data embedded

Filter likely, adjacent doublets

Filter, convert to triplets

Train/classify doublets in GNN

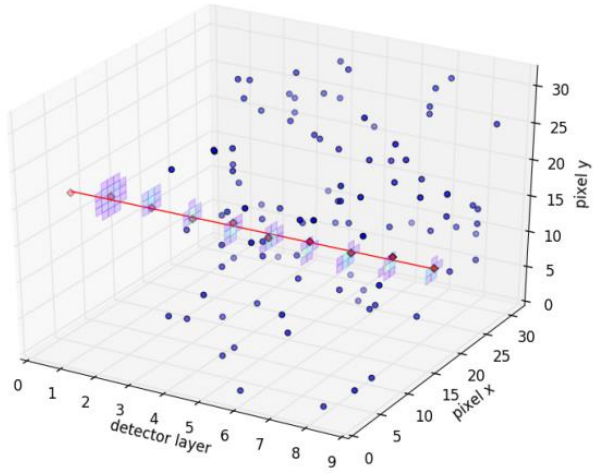
Train/classify triplets in GNN

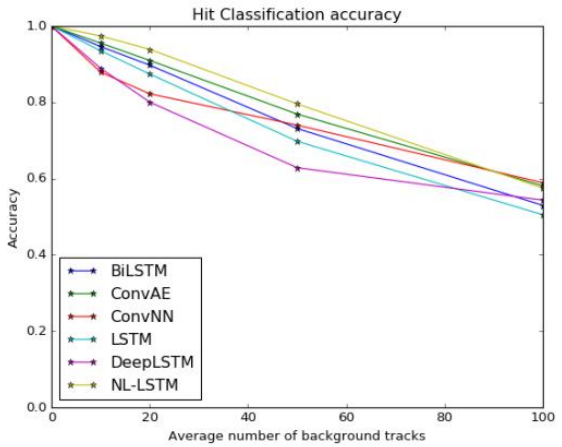
Apply cut for seeds

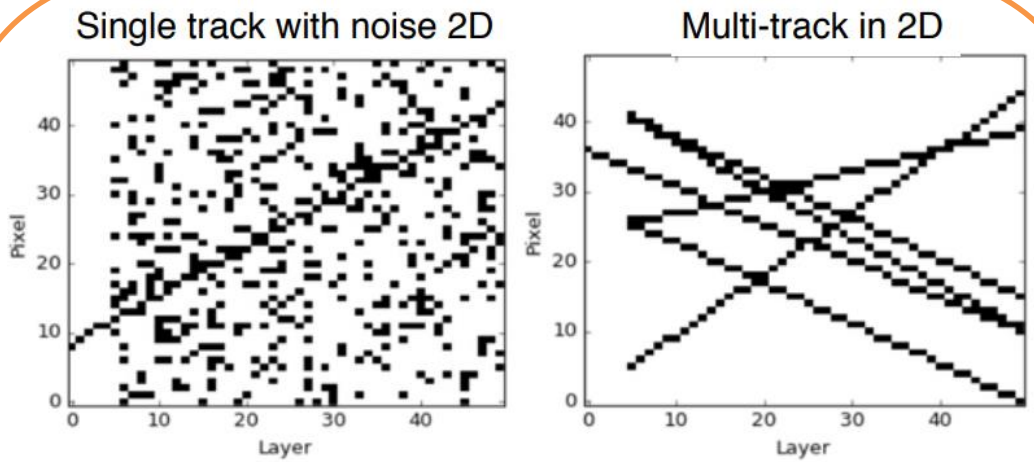
DBSCAN for track labels

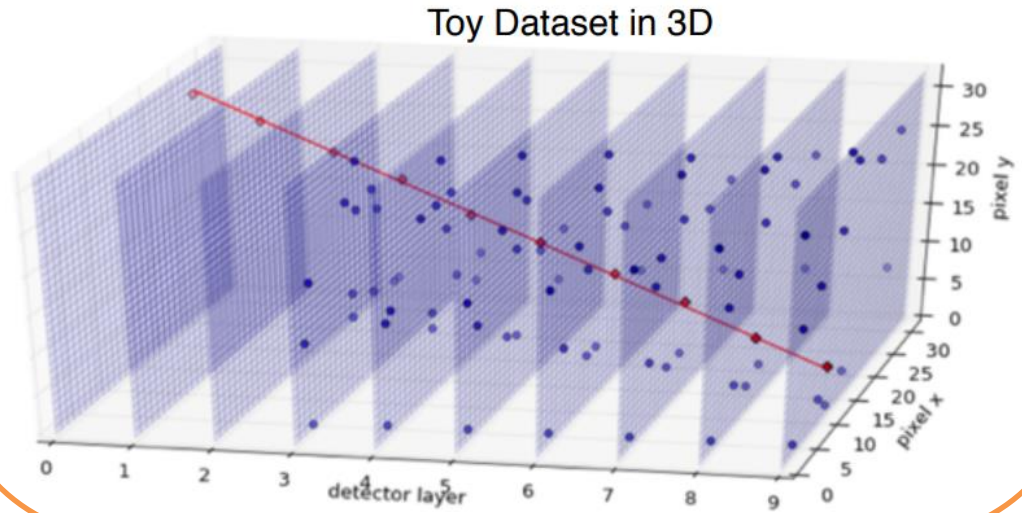
Previous ML Approaches

- Tracks as **images (CNN)**
- Tracks as **sequences of points (LSTM)**



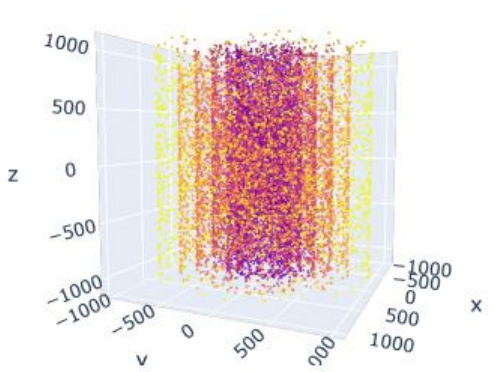




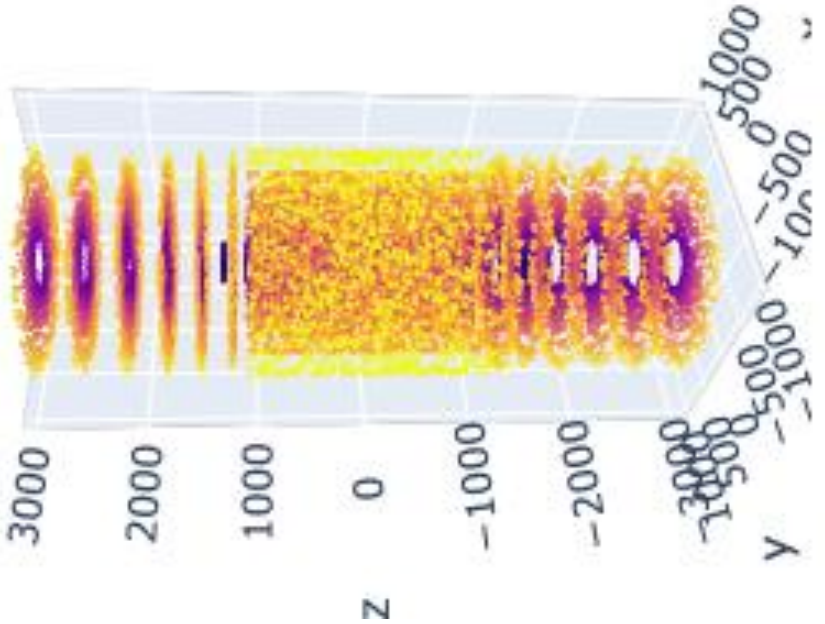


Dataset

- “TrackML Kaggle Competition” dataset
- Generated by simulation
- 9000 events to train on
- Each event has up to 100,000 layer hits from around 10,000 particles
- Layers can be hit multiple times by same particle (“duplicates”)
- Non-particle hits present (“noise”)

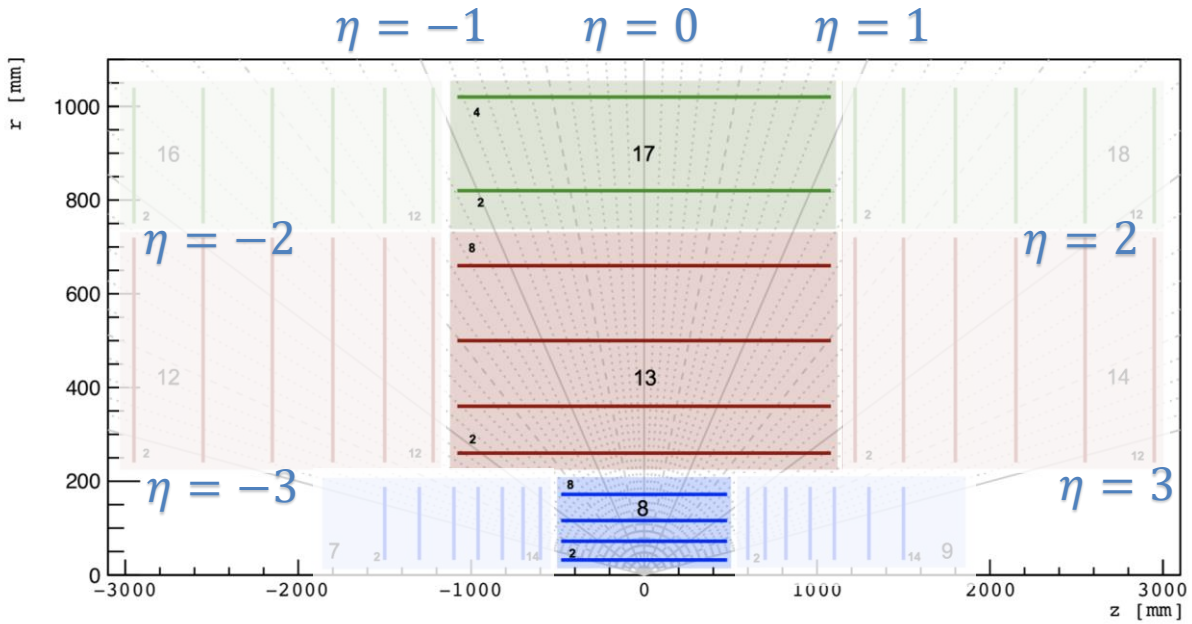


Barrel



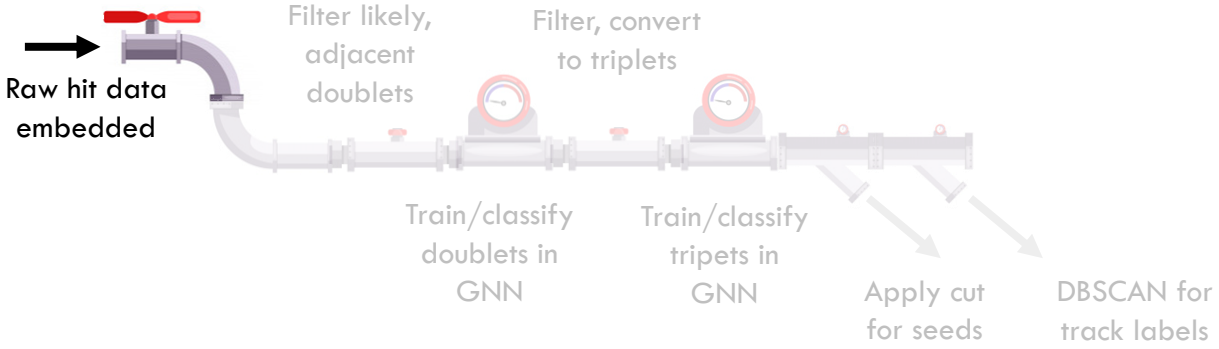
Full Detector

Dataset



- Ideal final result is a "TrackML score"
$$S \in [0,1]$$
- All hits belonging to same track labelled with same unique label $\Rightarrow S = 1$
- We use the **barrel** as a test case, and **ignore noise**

- Need to construct hit data into graph data, i.e. nodes and edges
- Can use geometric heuristics (have used in past: ~45% efficiency, 5% purity)
- To improve performance, use learned embedding construction

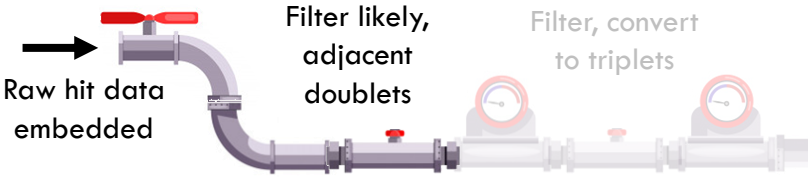


Embedding + MLP Filter Construction

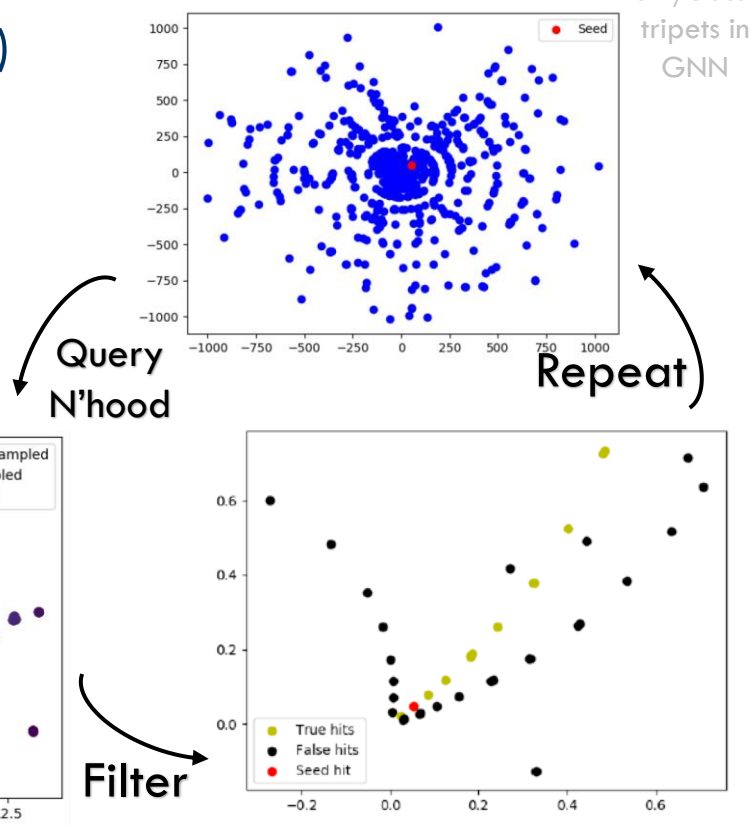
Won't give much detail: Nick Choma's (Berkeley Lab) talk covers embedding techniques

Generally:

1. For all hits in event, **embed** features (co-ordinates, cell direction data, etc.) into N-dimensional space
2. **Associate** hits from same tracks as close in N-dimensional distance
3. **Score** each "target" hit *within embedding neighbourhood* against the "seed" hit at centre
4. **Filter** by score, to create a set of seed-to-target doublets for the neighbourhood
5. All doublets in event **generate** a graph, converted to a directed graph *by ordering layers and applying a layer-adjacency condition.*



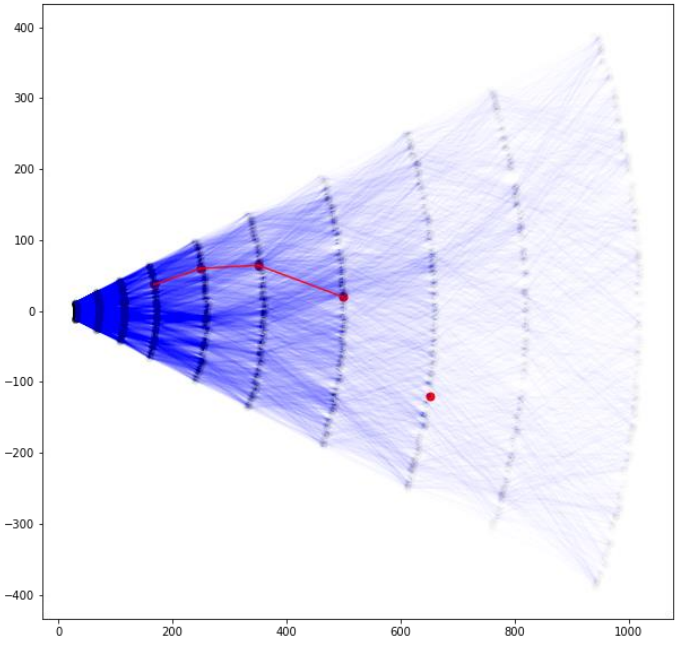
Train/classify tripets in GNN



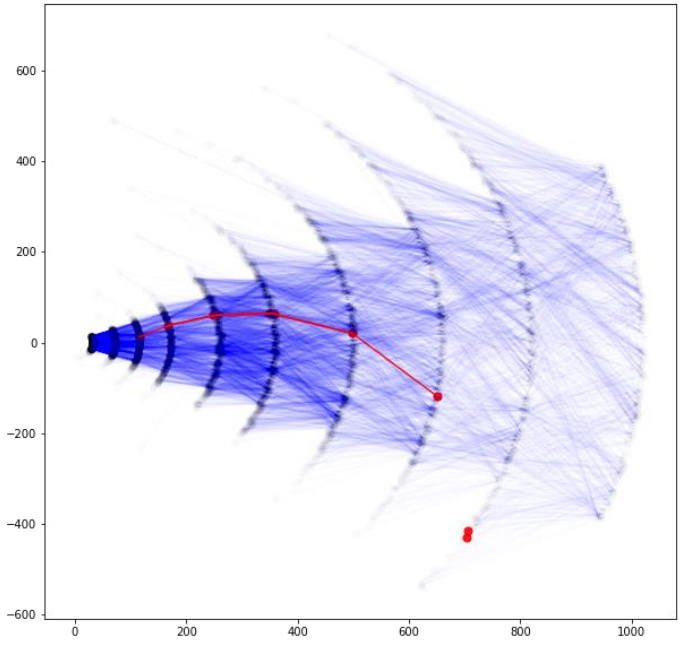
Segmentation

- A full graph from the embedding does not fit on a single GPU
- Therefore the event graphs are segmented, according to how large the GNN model is expected to be
- Different cut strategies interpolate between high efficiency and low memory use

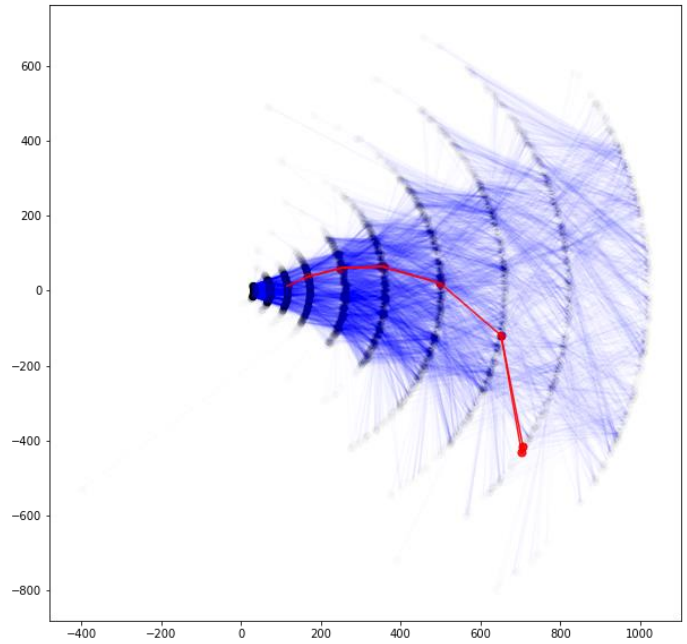
Hard cut



One-directional soft cut



Bi-directional soft cut



Faster, low memory

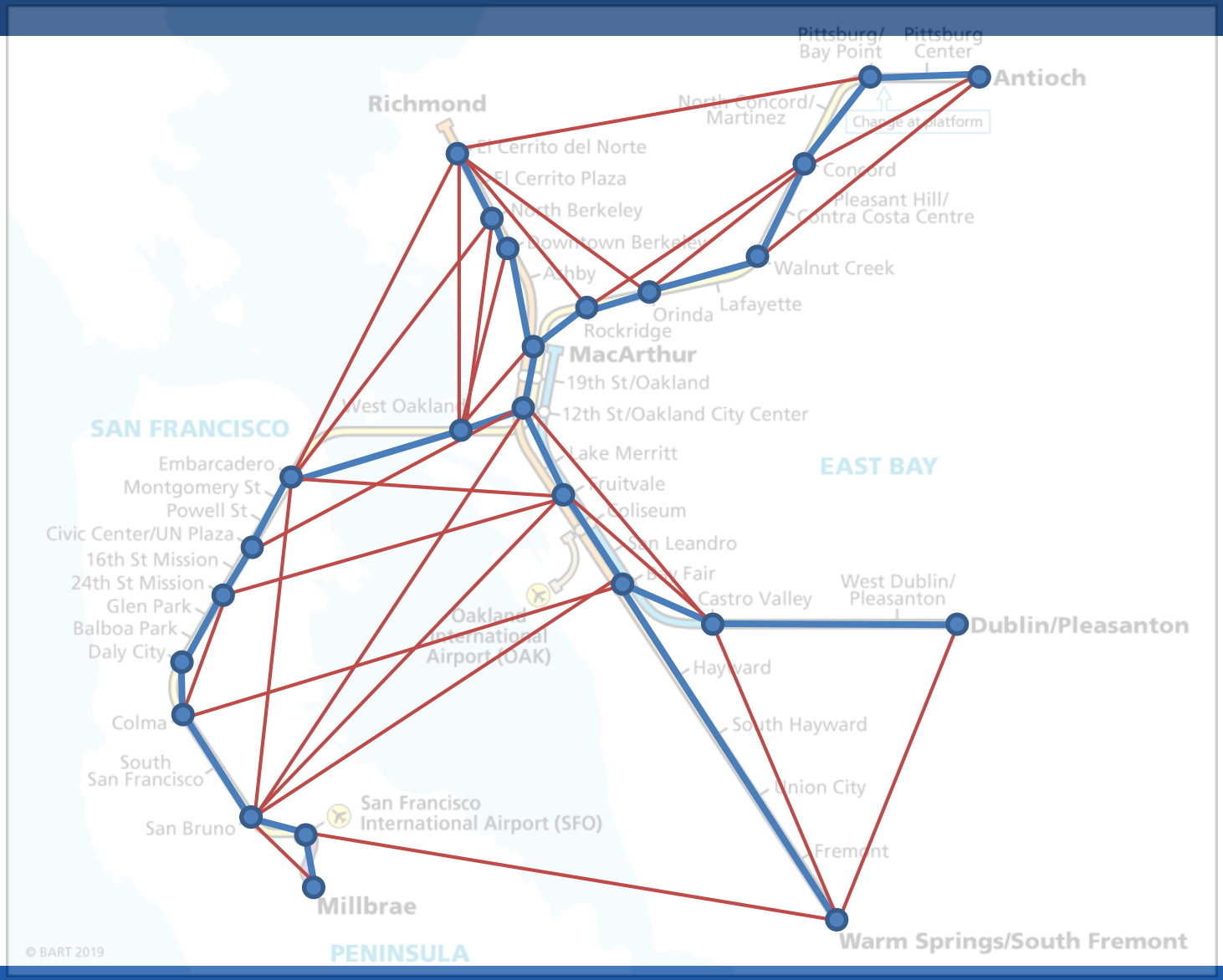
Better accuracy, efficiency

Graph Neural Network for Edge Classification

Classify edges with score between [0,1]

score > cut: real

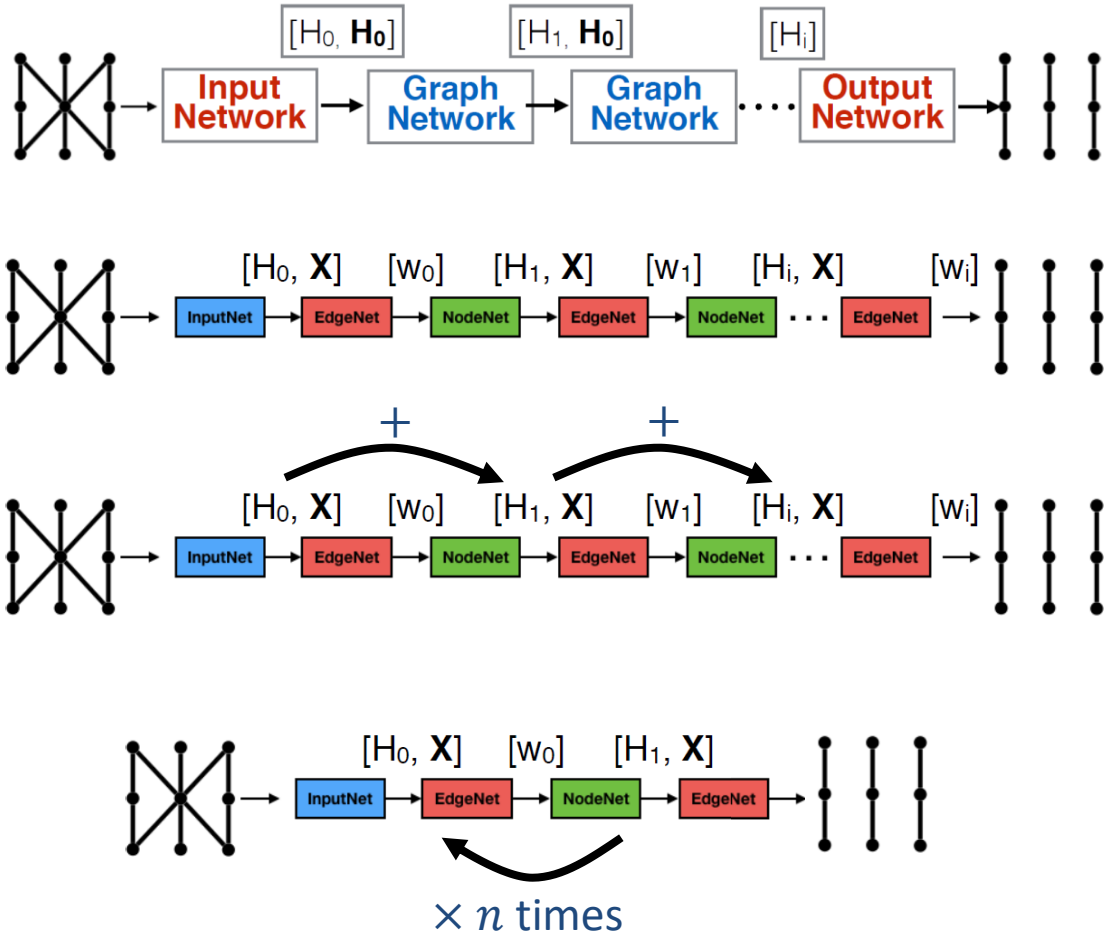
score < cut: fake



Passing information around the graph gives it learning power

- Can make a node "aware" of its neighbours by concatenating the neighbouring hidden features
- Iterating this neighbourhood learning passes information around the graph
- Can be considered a generalisation of a flat CNN convolution

GNN Edge prediction architecture



- Message Passing

Gilmer, Justin, et al. "Neural message passing for quantum chemistry." *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017.

- Attention Message Passing

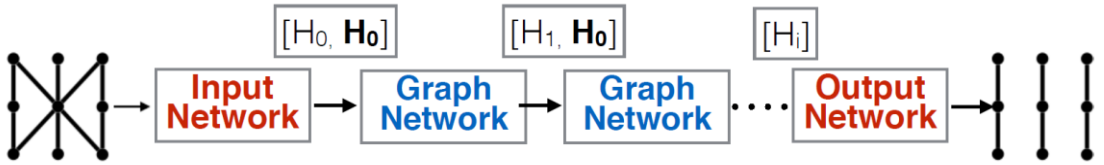
Veličković, Petar, et al. "Graph attention networks." *arXiv preprint arXiv:1710.10903* (2017).

- Attention Message Passing with Residuals

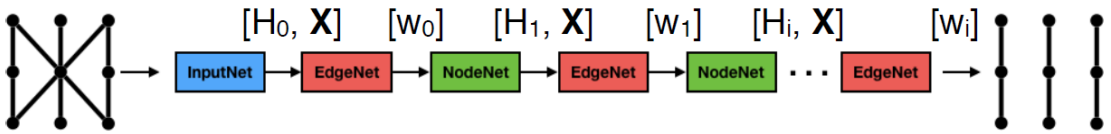
Kipf, Thomas N., and Max Welling. "Semi-supervised classification with graph convolutional networks." *arXiv preprint arXiv:1609.02907* (2016).

- Attention Message Passing with Recursion

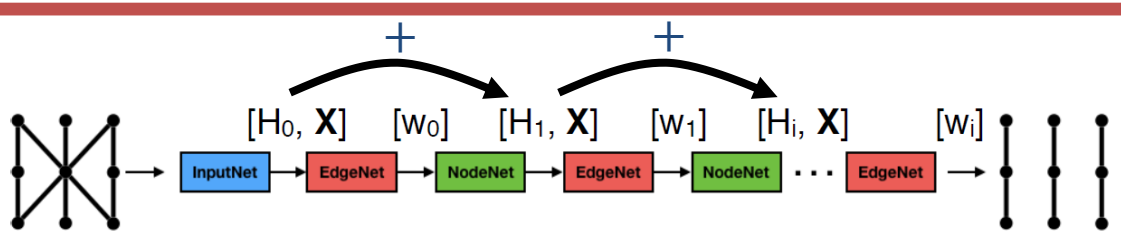
GNN Edge prediction architecture



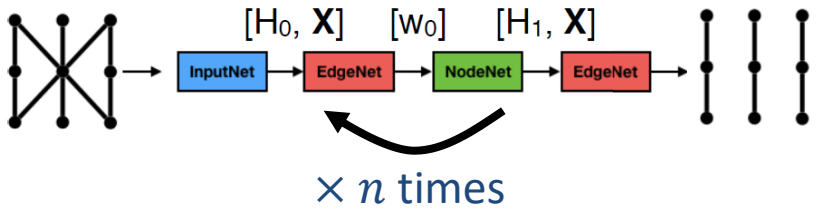
- Message Passing



- Attention Message Passing



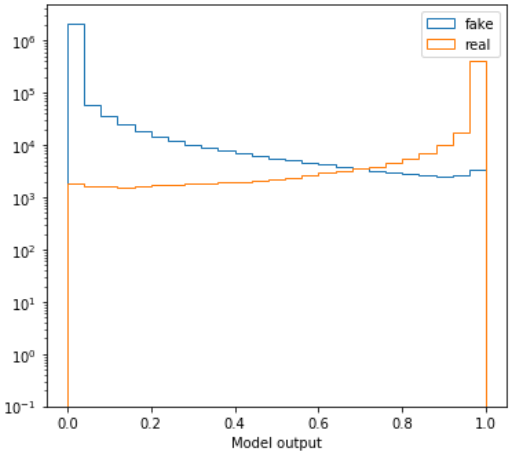
- Attention Message Passing with Residuals

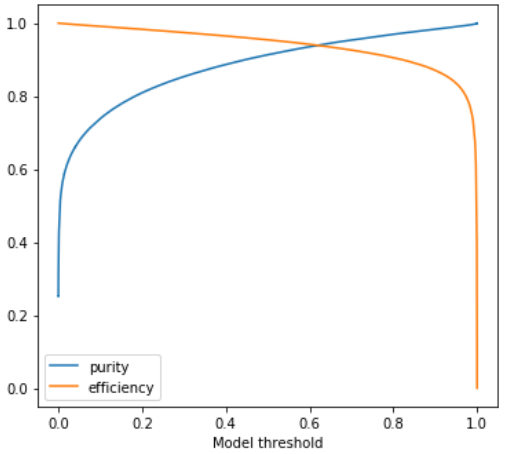


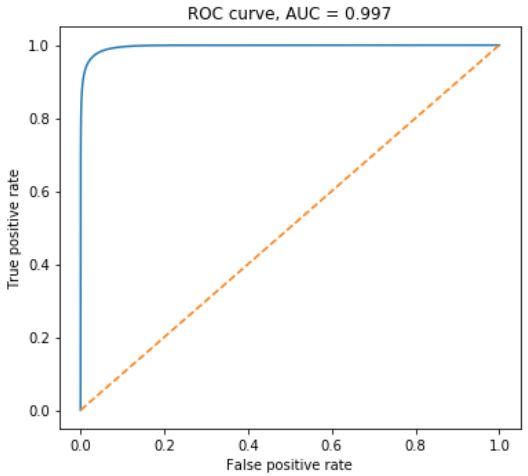
- Attention Message Passing with Recursion

Have found best efficiency & purity performance.

Doublet GNN Performance



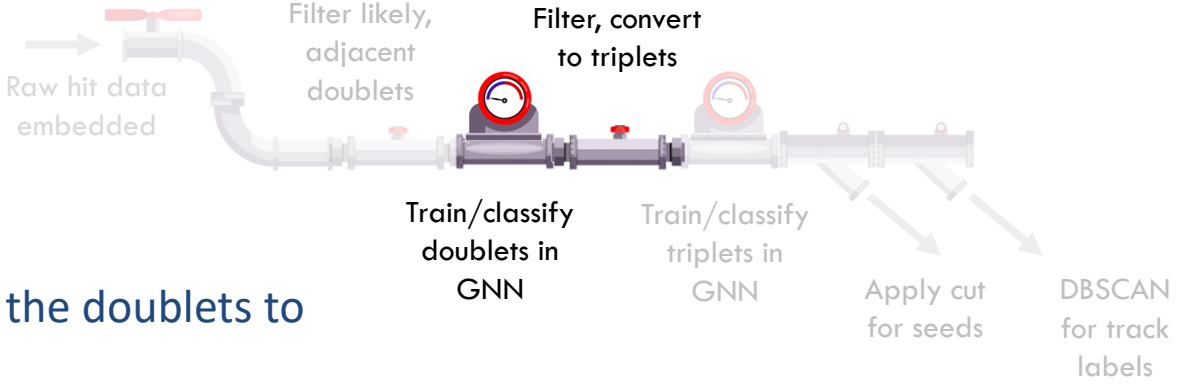




Threshold	0.5	0.8
Accuracy	0.9761	0.9784
Purity	0.9133	0.9694
Efficiency	0.9542	0.9052

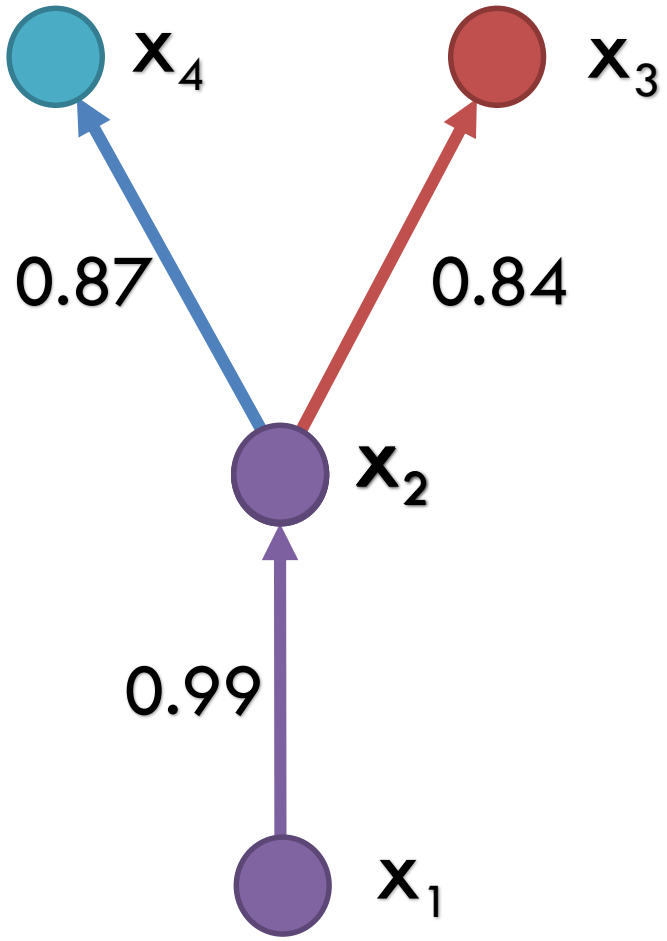
Two points to keep in mind:

- In the past, graphs have been constructed with a heuristic procedure that had much lower efficiency than the learned embedding. This GNN is classifying an already ~ 96% efficient doublet dataset
- These metrics are not the end product: we use the scores of the doublets to create triplets without losing efficiency



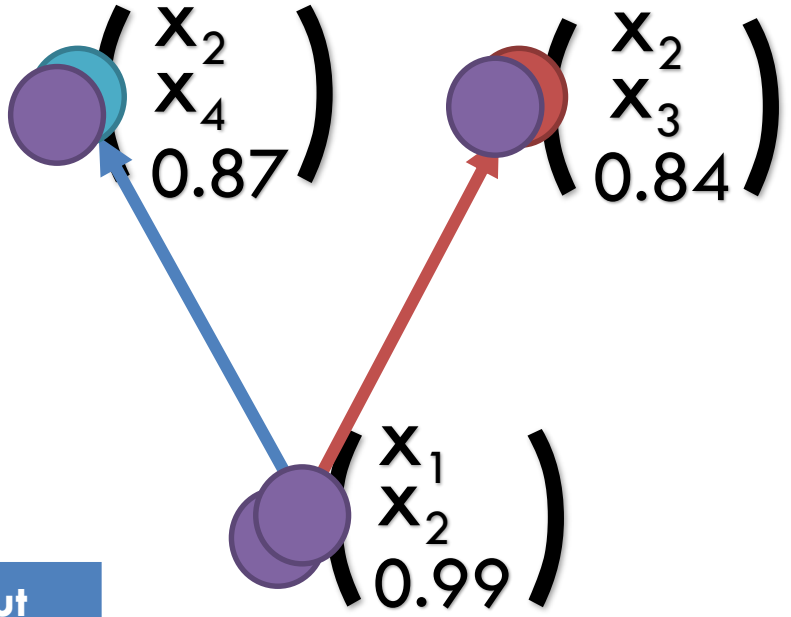
Triplet Construction

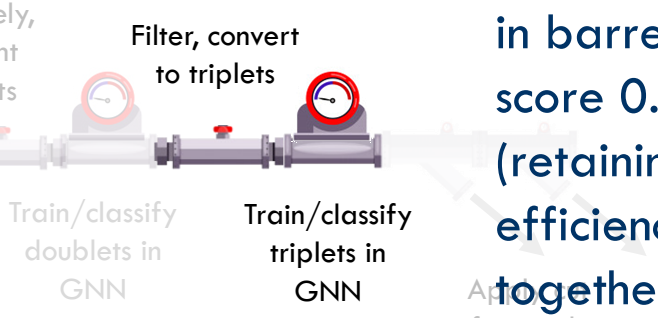
- We convert from a doublet graph to triplet graph: triplet edges have direct access to curvature information, therefore we hypothesise the accuracy should be even better
- Doublets are associated to nodes, triplets are associated to edges



Triplet Construction

- We convert from a doublet graph to triplet graph: triplet edges have direct access to curvature information, therefore we hypothesise the accuracy should be even better
- Doublets are associated to nodes, triplets are associated to edges





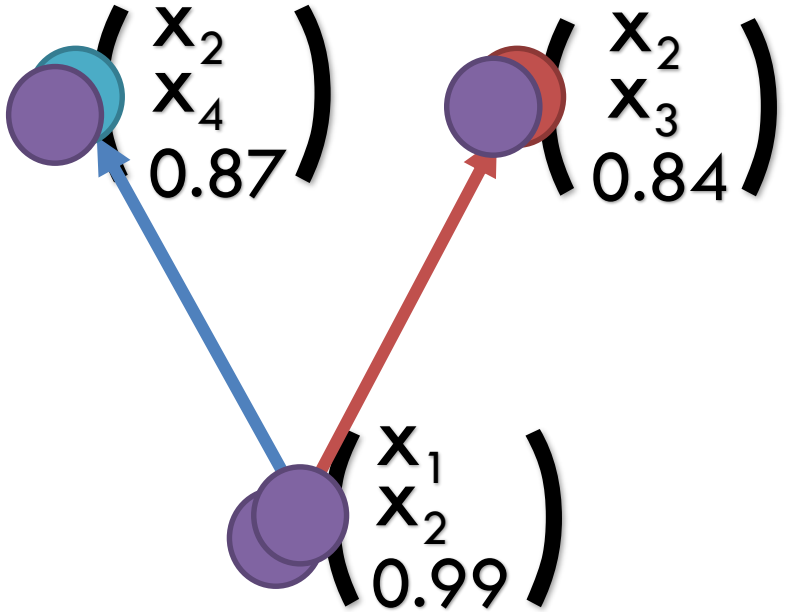
- For a TrackML event in barrel, using a score 0.1 cut (retaining 99.12% efficiency), and stitch together graph segments:

Magnitude	Before cut	After cut
Tracks	$O(6{,}000)$	
Hits	$O(40{,}000)$	
Doublets	$O(140{,}000)$	$O(40{,}000)$
Triplets	$O(400{,}000)$	$O(60{,}000)$

- We thus have a sustainable process to N-plet GNN

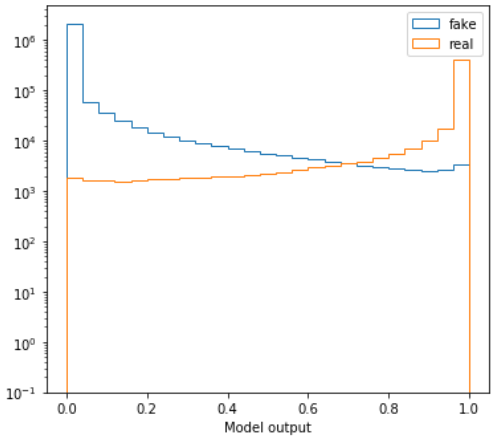
Triplet Construction

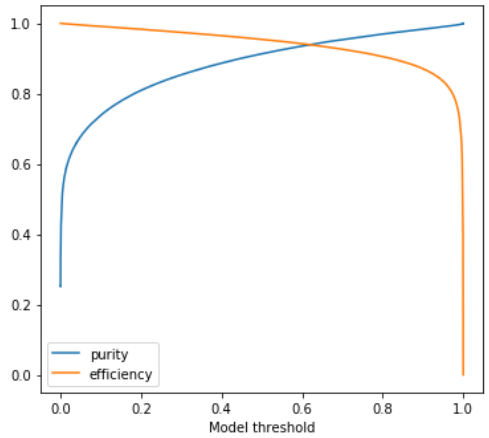
- We convert from a doublet graph to triplet graph: triplet edges have direct access to curvature information, therefore we hypothesise the accuracy should be even better
- Doublets are associated to nodes, triplets are associated to edges
- Use an extremely fast sparse doublet-to-triplet conversion method, with sparse adjacency matrices on GPU with CuPy @ ~ 90ms per event:

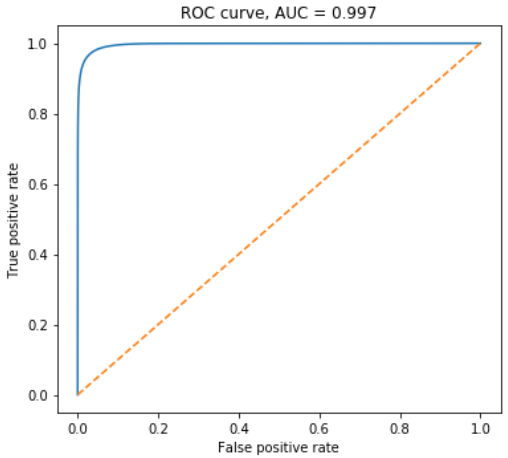


$$e_{triplet} = {}^{n_{doublets}}\begin{pmatrix} \\ \\ \end{pmatrix}^{n_{hits}} \times {}^{n_{doublets}}\begin{pmatrix} \\ \\ \end{pmatrix}^{n_{hits}} {}^{T}$$

Triplet GNN Performance

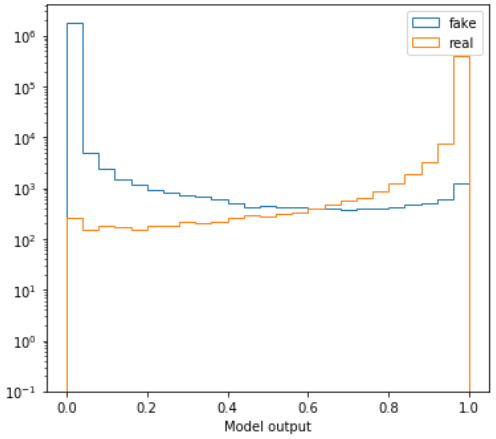


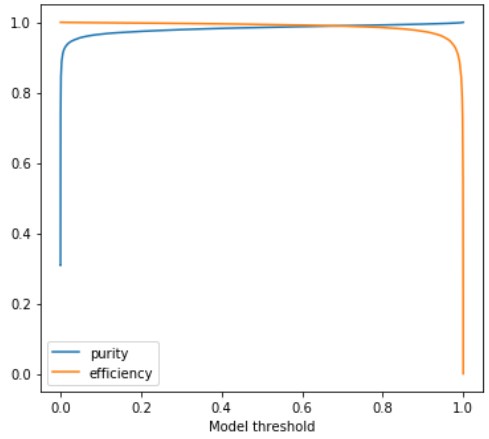


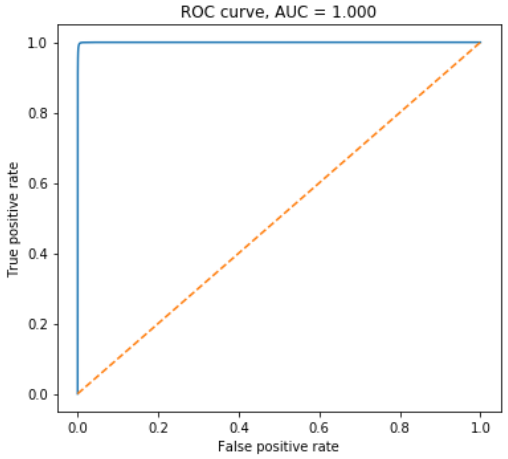


Threshold	0.5	0.8
Accuracy	0.9761	0.9784
Purity	0.9133	0.9694
Efficiency*	0.9542	0.9052

Doublet GNN







Triplet GNN

Threshold	0.5	0.8
Accuracy	0.9960	0.9957
Purity	0.9854	0.9923
Efficiency*	0.9939	0.9850

*efficiencies are relative to graph data fed into each GNN stage, *not* "truth graph"

Triplet GNN Performance

Gold: Unambiguously correct triplet or quadruplet

Other colours: False positive/negative

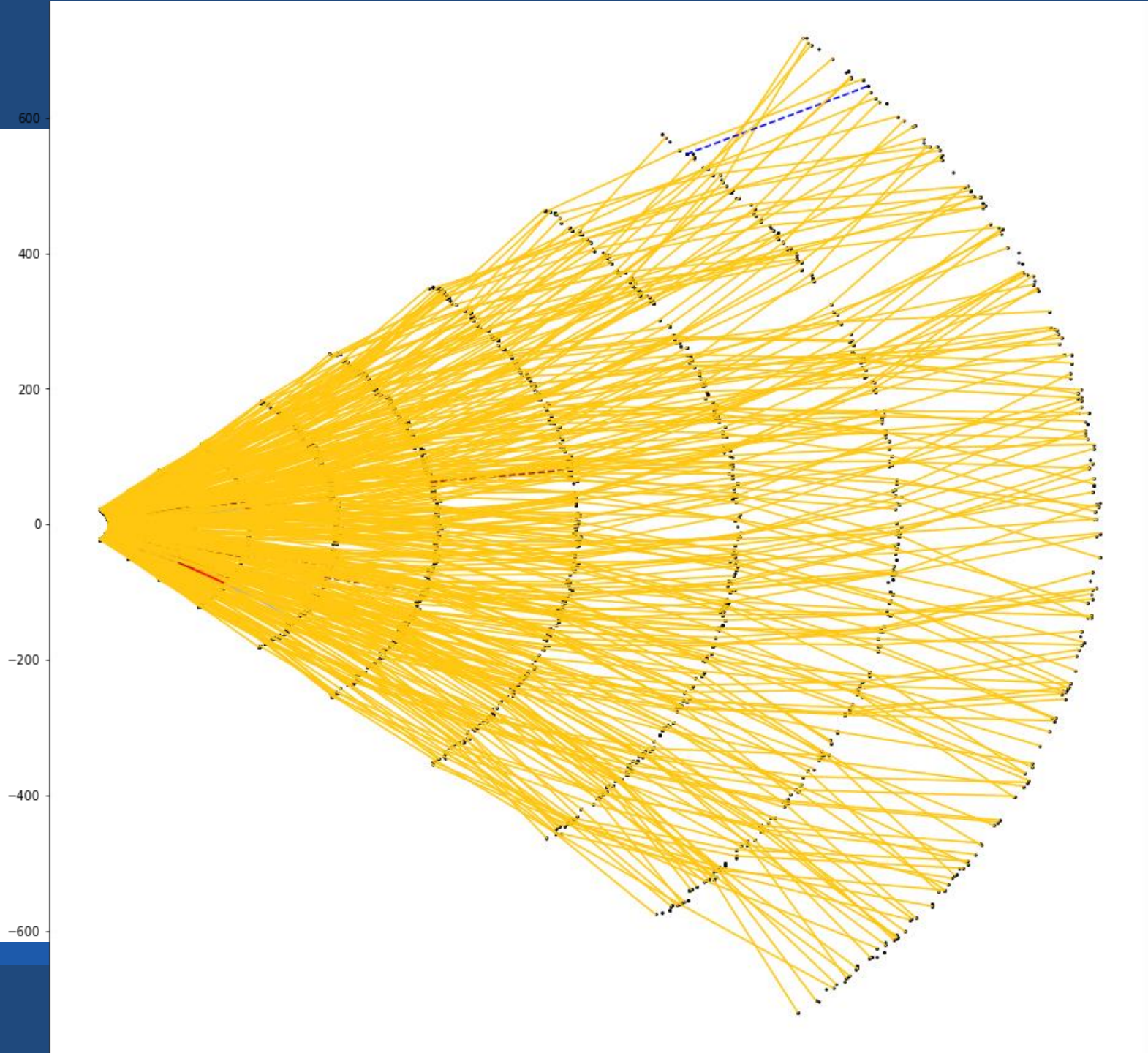
Key:

Silver: Ambiguously correct triplet or quadruplet (i.e. edge shared by correct triplet and false positive triplet)

Bronze dashed: Correct triplet, but missed quadruplet (i.e. edge shared by correct triplet and false negative triplet)

Red: Completely false positive triplet

Blue dashed: Completely false negative triplet



Triplet GNN Performance

Gold: Unambiguously correct triplet or quadruplet

Other colours: False positive/negative

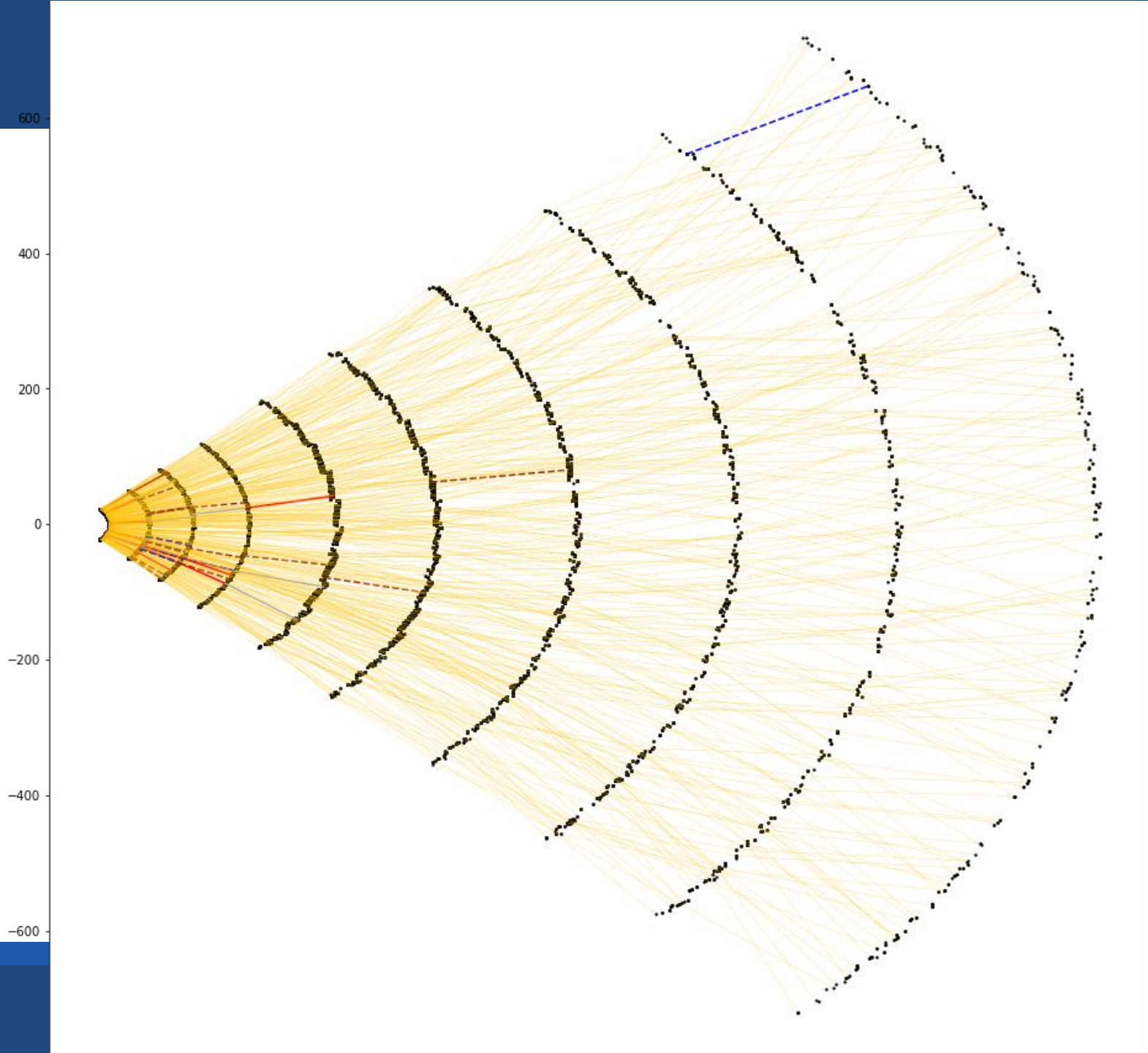
Key:

Silver: Ambiguously correct triplet or quadruplet (i.e. edge shared by correct triplet and false positive triplet)

Bronze dashed: Correct triplet, but missed quadruplet (i.e. edge shared by correct triplet and false negative triplet)

Red: Completely false positive triplet

Blue dashed: Completely false negative triplet



Triplet GNN improves doublet GNN results

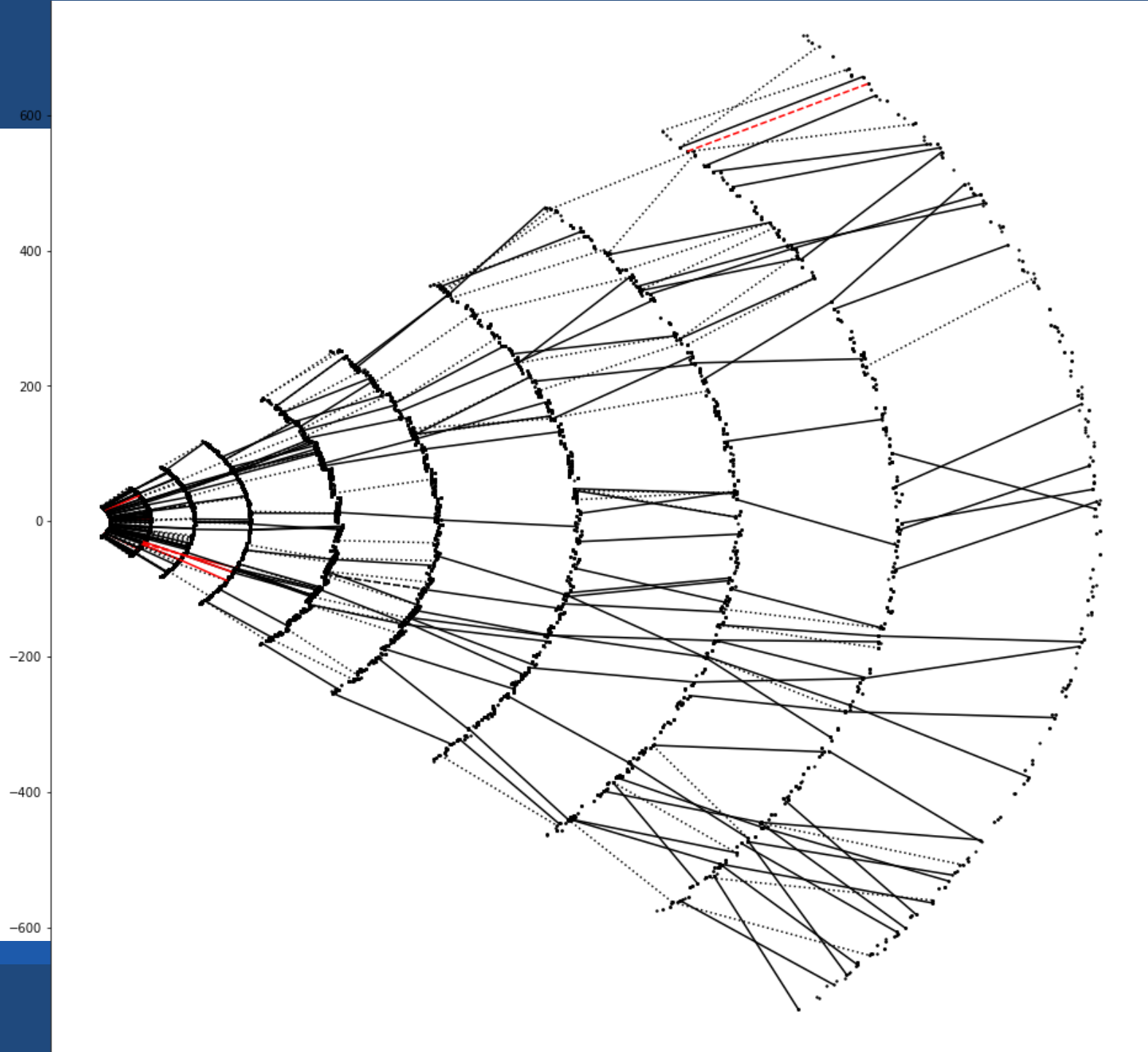
Black: Triplet classifier correctly labelled, doublet classifier mislabelled

Red: Doublet classifier correctly labelled, triplet classifier mislabelled

In this graph, triplet classifier

Fixes 389 edges

Worsens 10 edges



Seeding: Final Performance

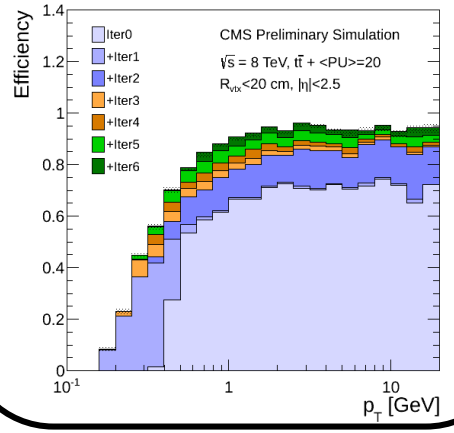
Benchmarks

ATLAS PU=40

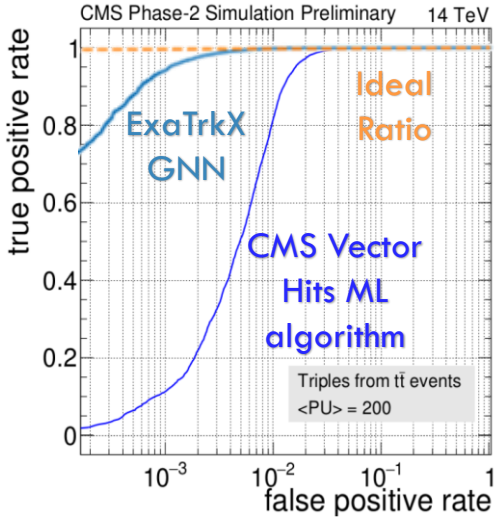
Triplet efficiency: 90%
Triplet purity: 35%
Quadruplet purity: 70%
Timing: 4.7 seconds

Elsing, Lecture Series, Frieburg University 2016
https://elsing.web.cern.ch/elsing/talks/FreiburgLecture3.pdf

CMS PU=20

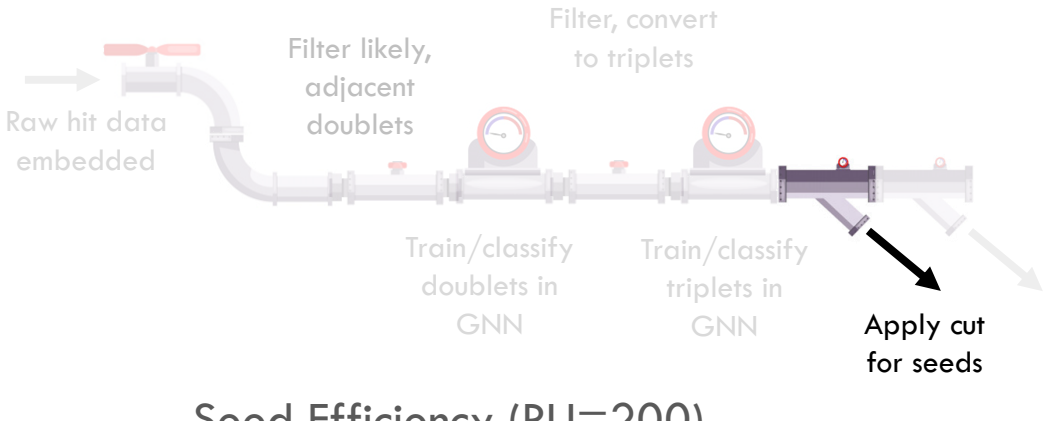


CMS PU=200

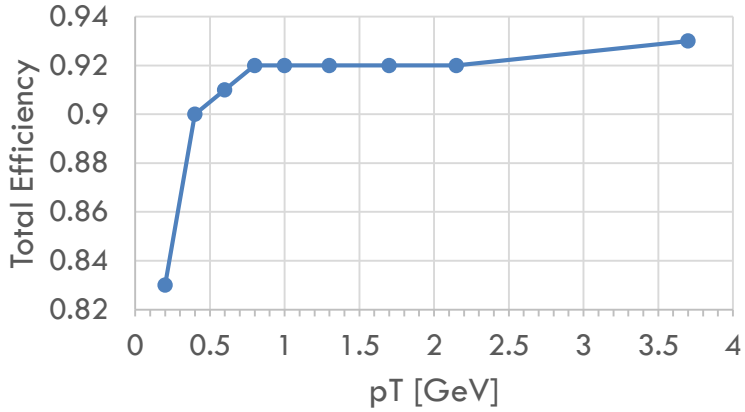


Brondolin, et al. ACAT 2017 proceedings
https://indi.to/m7y9s

ExaTrkX GNN Seeding



Seed Efficiency (PU=200)



Purity: $99.1\% \pm 0.07\%$

Inference time: ~ **5 seconds per event per GPU,** split between:

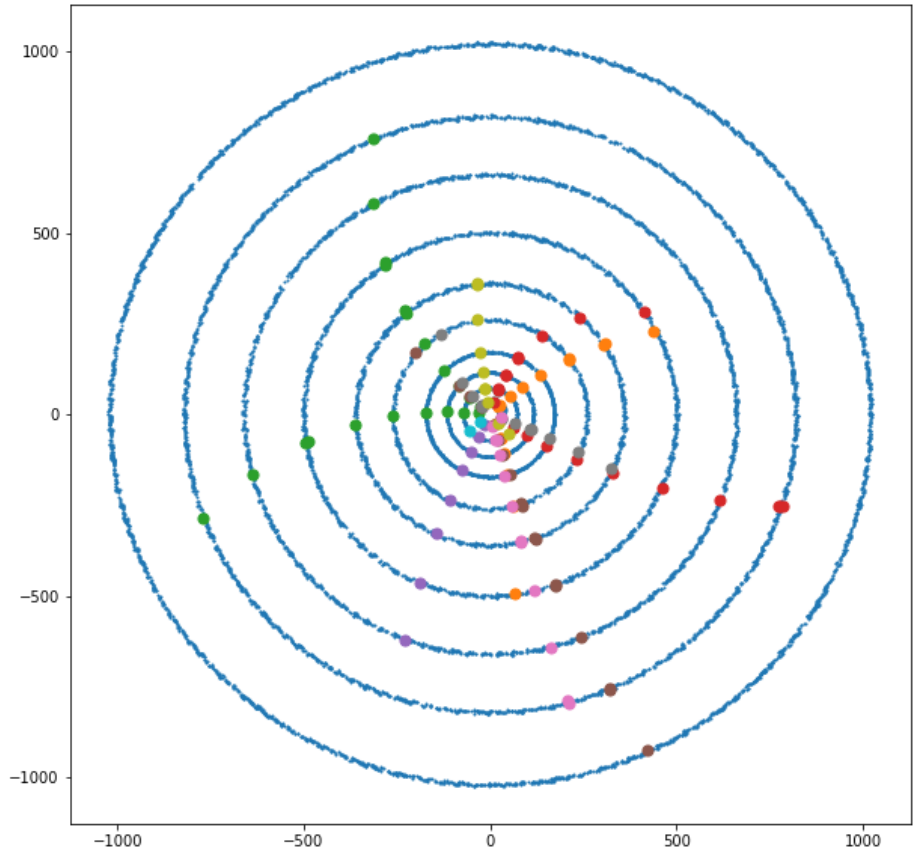
~ 3 ± 1 seconds for embedding construction

~ 2 ± 1 seconds for two GNN steps and processing

Track Labelling

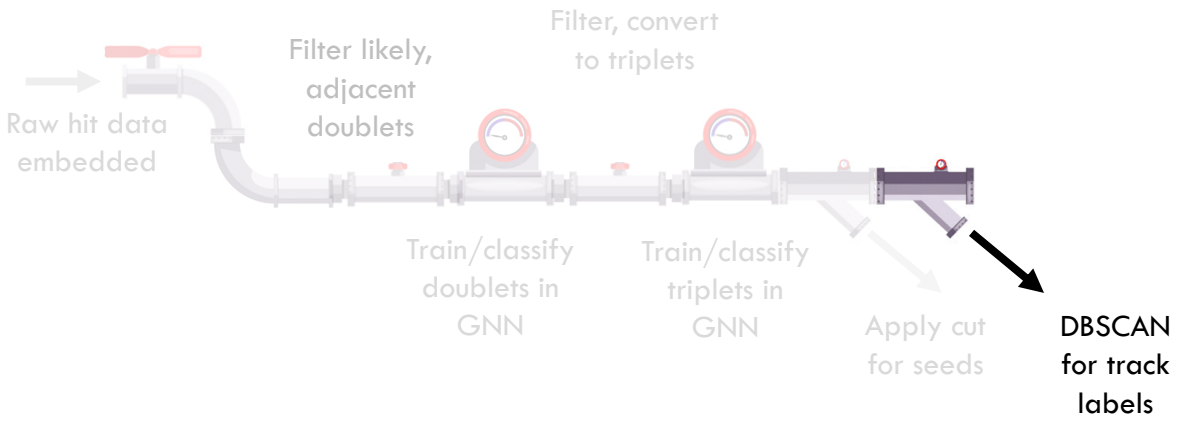
GOAL

Given a classified doublet and/or triplet graph, use edge scores to group likely nodes into tracks and label with unique identifier.



DBSCAN on a Graph

- DBSCAN typically calculates a distance metric and clusters based on neighbourhood density



- To use on graph representation, feed the edge scores e_{ij} as a *precomputed,* sparse, metric matrix, with each distance element given by

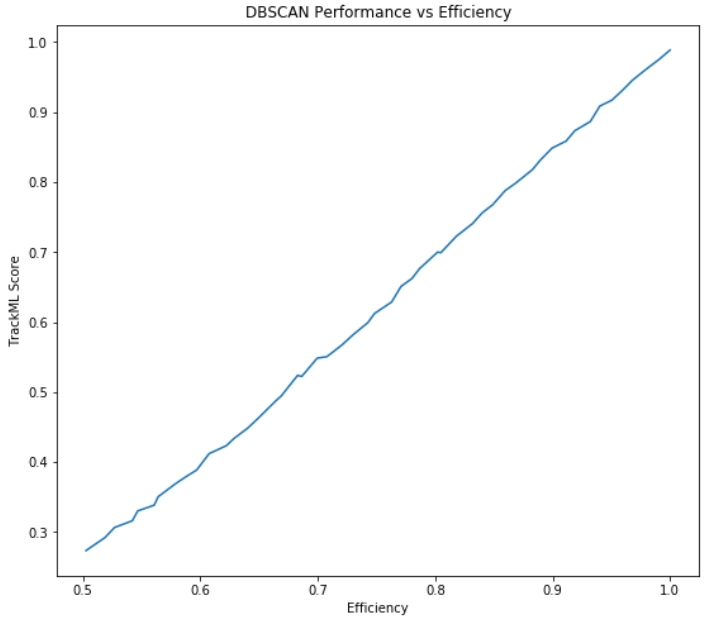
$$d_{ij} = 1 - e_{ij}$$

- N.B. We fill out sparse matrix to ensure it is diagonal, i.e. undirected. A directed graph does not perform well with DBSCAN.

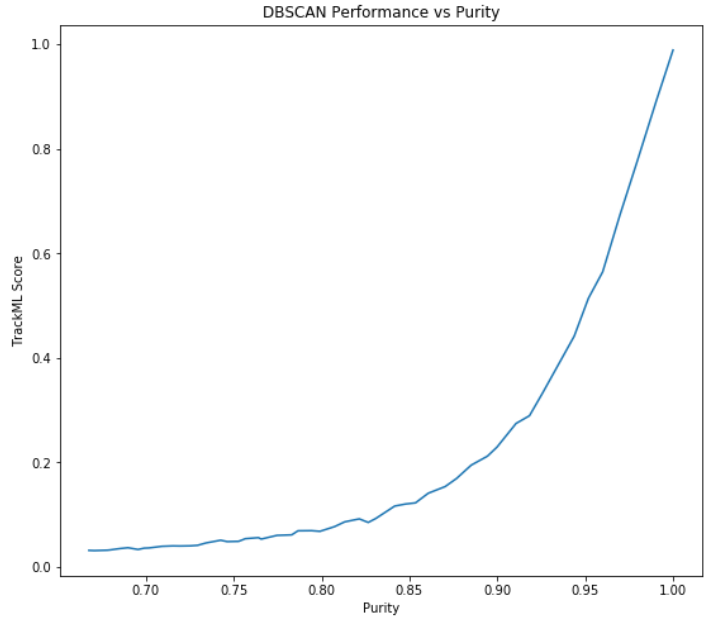
DBSCAN Performance

- We construct a “truth graph” from TrackML data:
  Every hit is connected to hits of a shared track in adjacent layers, with true score (i.e. 1), and randomly connected to other hits with a fake score (i.e. 0)

- We can randomly mislabel true edges to reduce efficiency, or mislabel fake edges to reduce purity



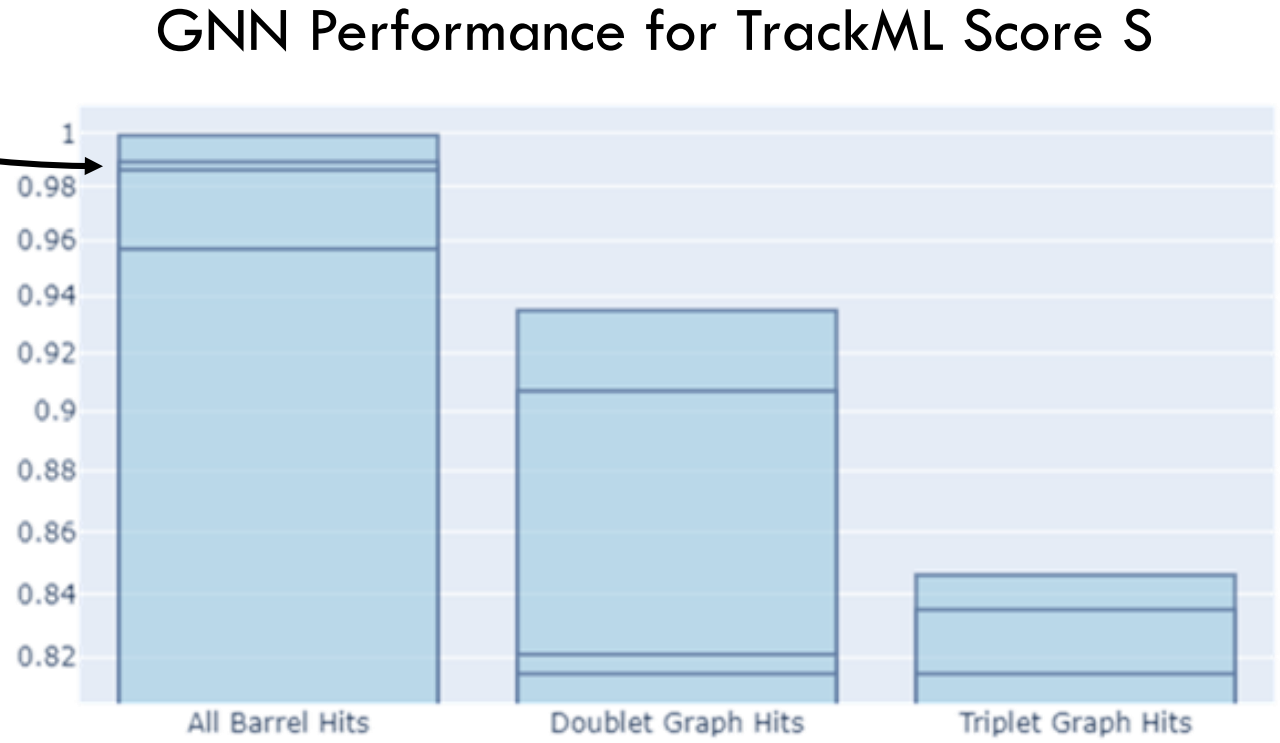
Linear reduction in TrackML score against efficiency



Exponential reduction in TrackML score against purity

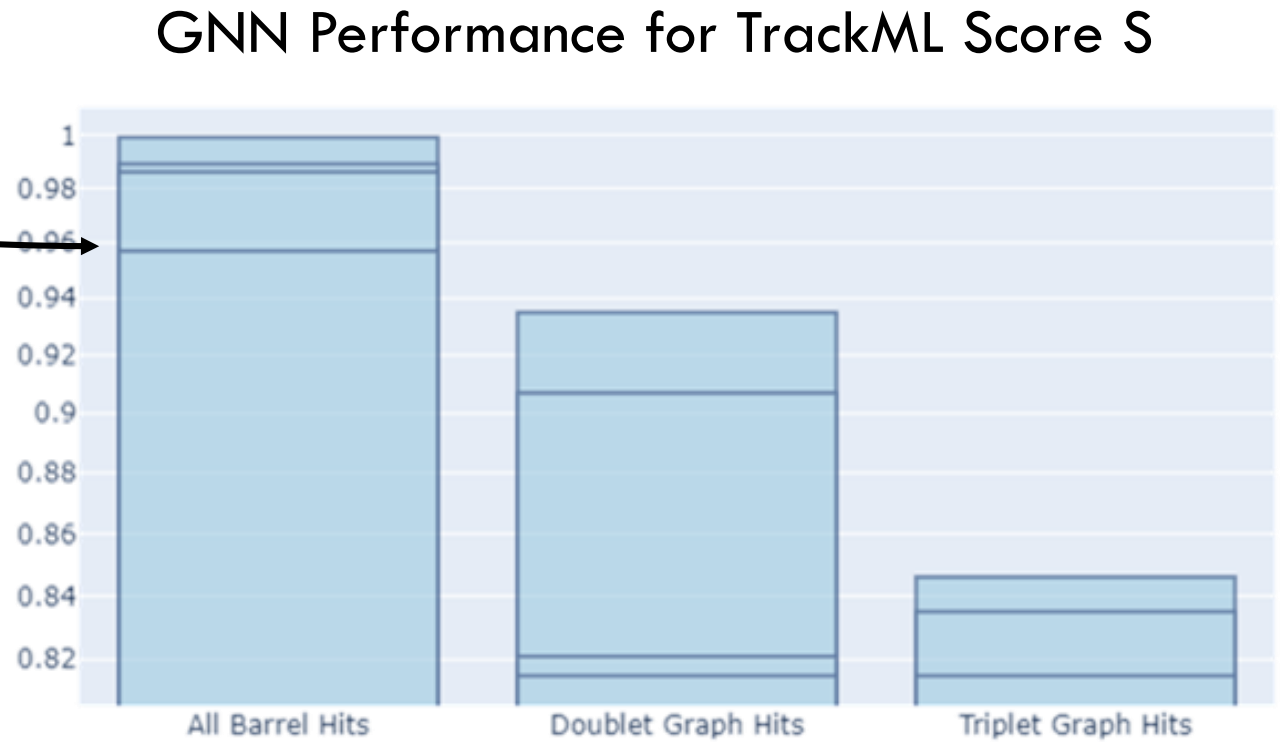
GNN TrackML Score Performances

- DBSCAN on truth graph
  S = 0.989



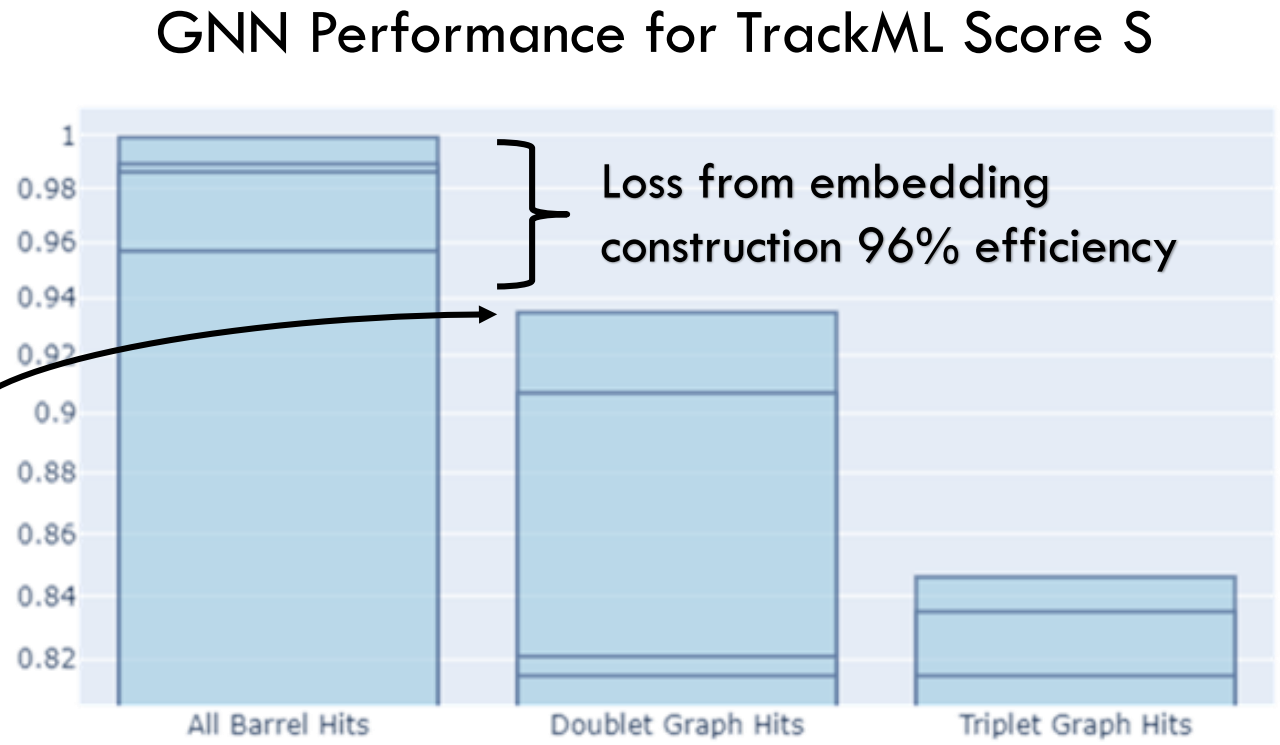
GNN TrackML Score Performances

- DBSCAN on truth graph
  S = 0.989
- DBSCAN on adjacent-layer truth graph
  S = 0.957



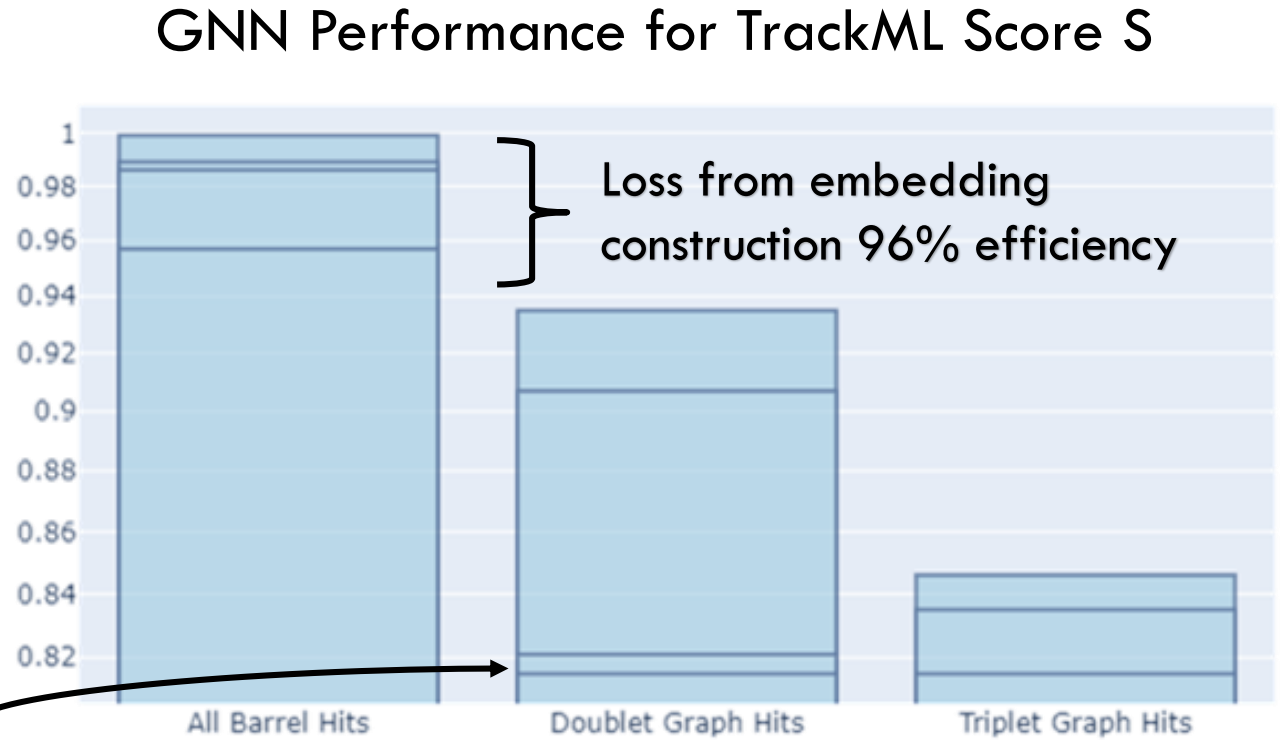
GNN TrackML Score Performances

- DBSCAN on truth graph
  S = 0.989
- DBSCAN on adjacent-layer truth graph
  S = 0.957
- Embedding-constructed doublet graph using truth
  S = 0.935



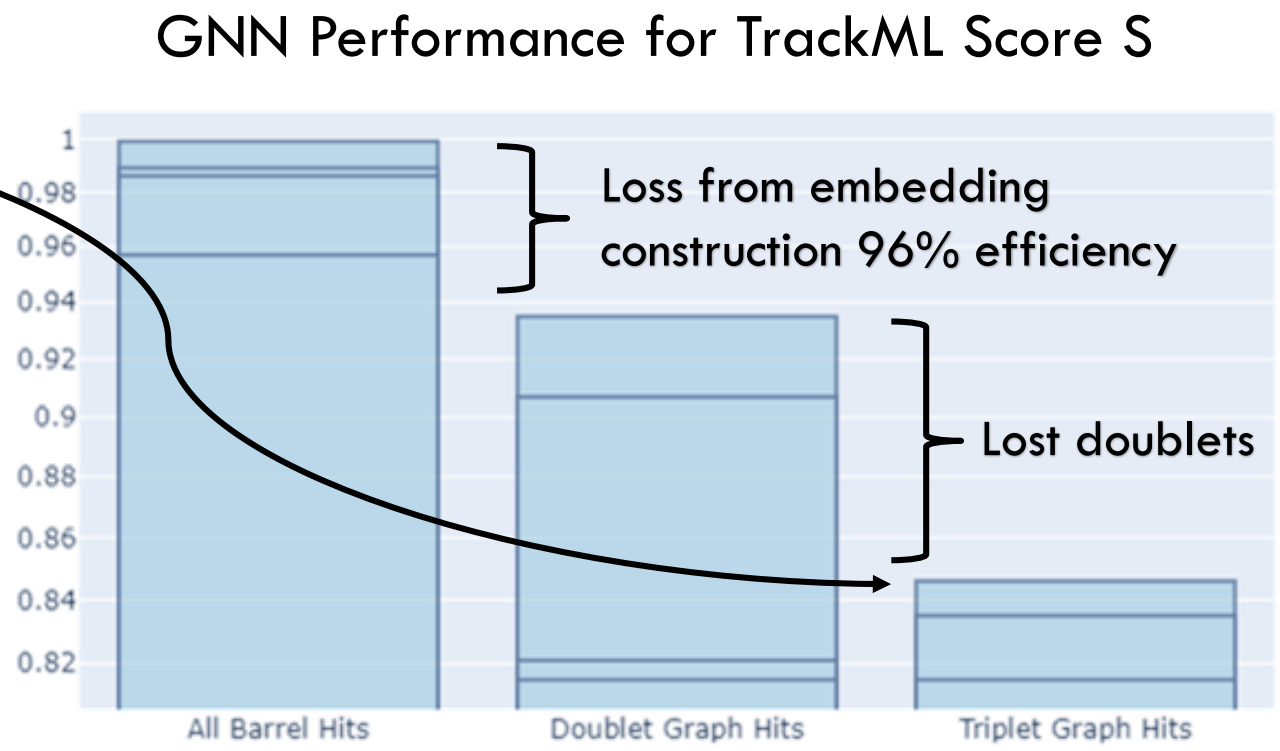
GNN TrackML Score Performances

- DBSCAN on truth graph
  S = 0.989
- DBSCAN on adjacent-layer truth graph
  S = 0.957
- Embedding-constructed doublet graph using truth
  S = 0.935
- DBSCAN on doublet GNN classification
  S = 0.805



GNN TrackML Score Performances

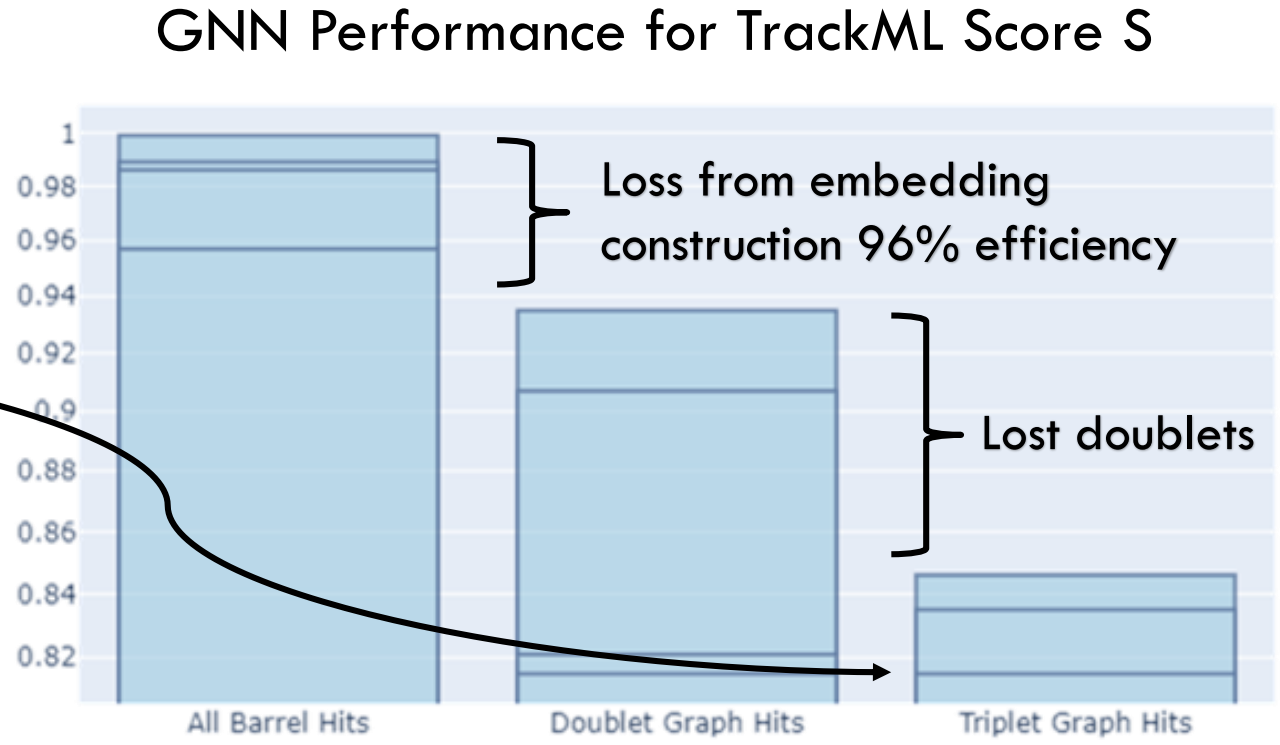
- Triplet graph constructed from doublet graph (truth)
  S = 0.846



GNN TrackML Score Performances

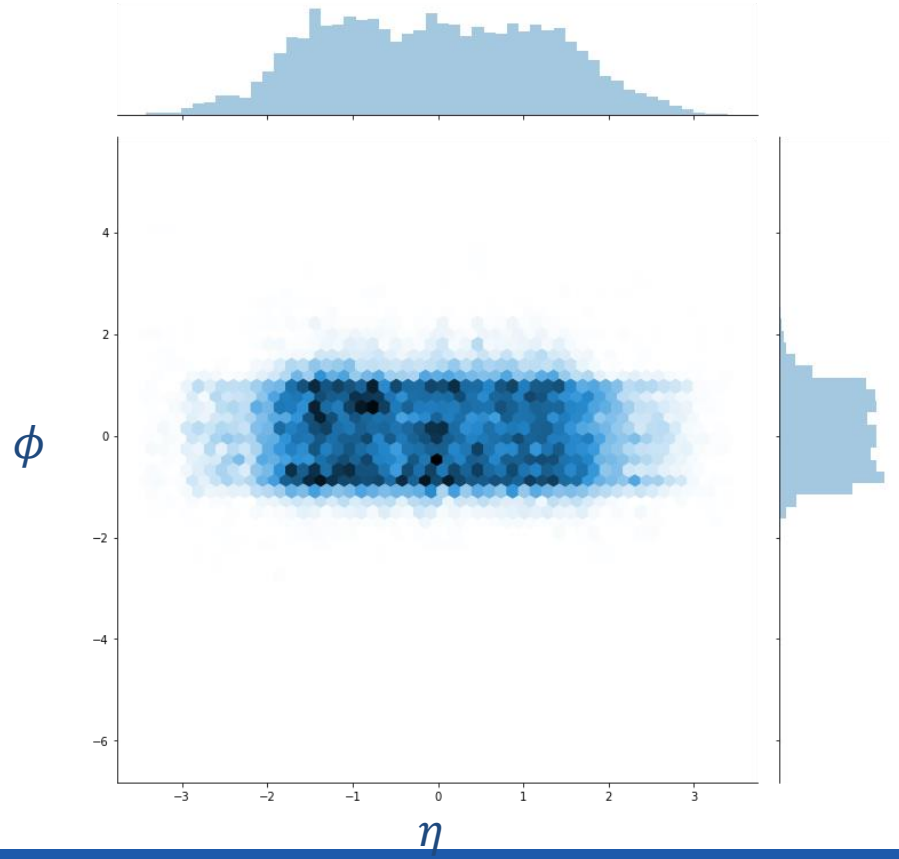
- Triplet graph constructed from doublet graph (truth)
  S = 0.846
- DBSCAN on triplet graph from triplet GNN classification
  S = 0.815

We have two holes to mend:
Embedding efficiency
Lost doublets

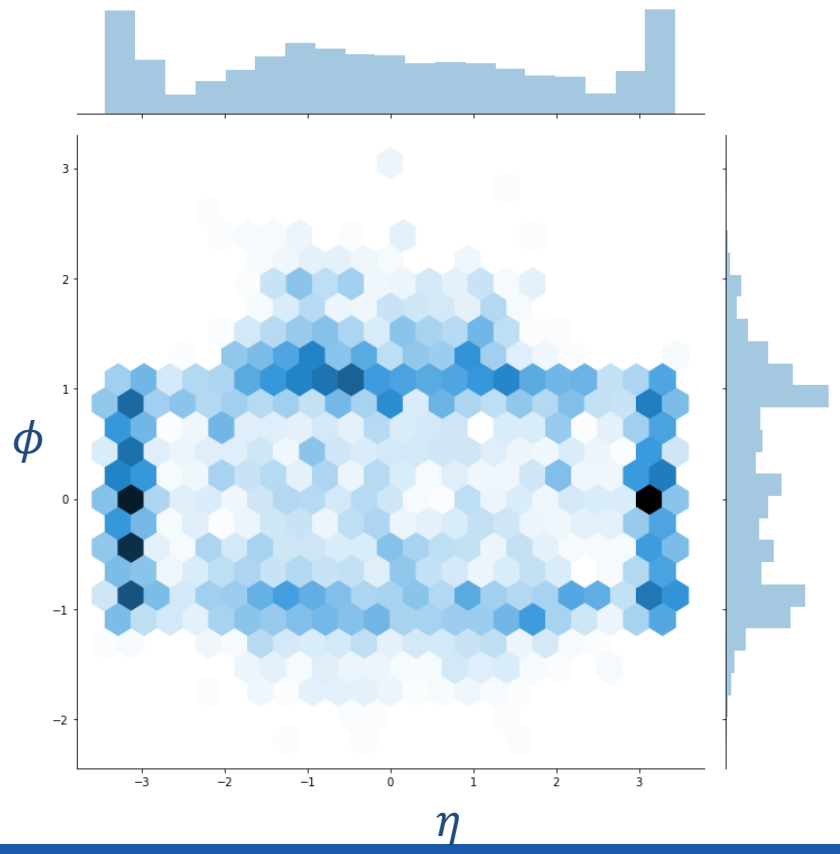


Missing Doublets

All Hits

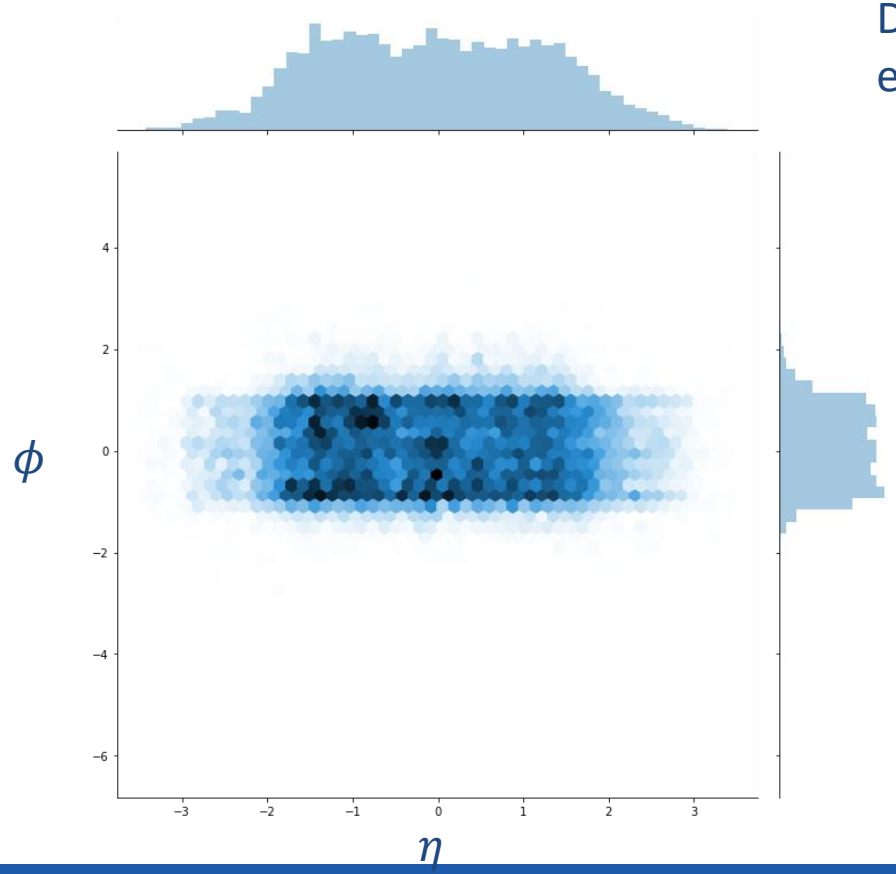


Missing Doublet Hits



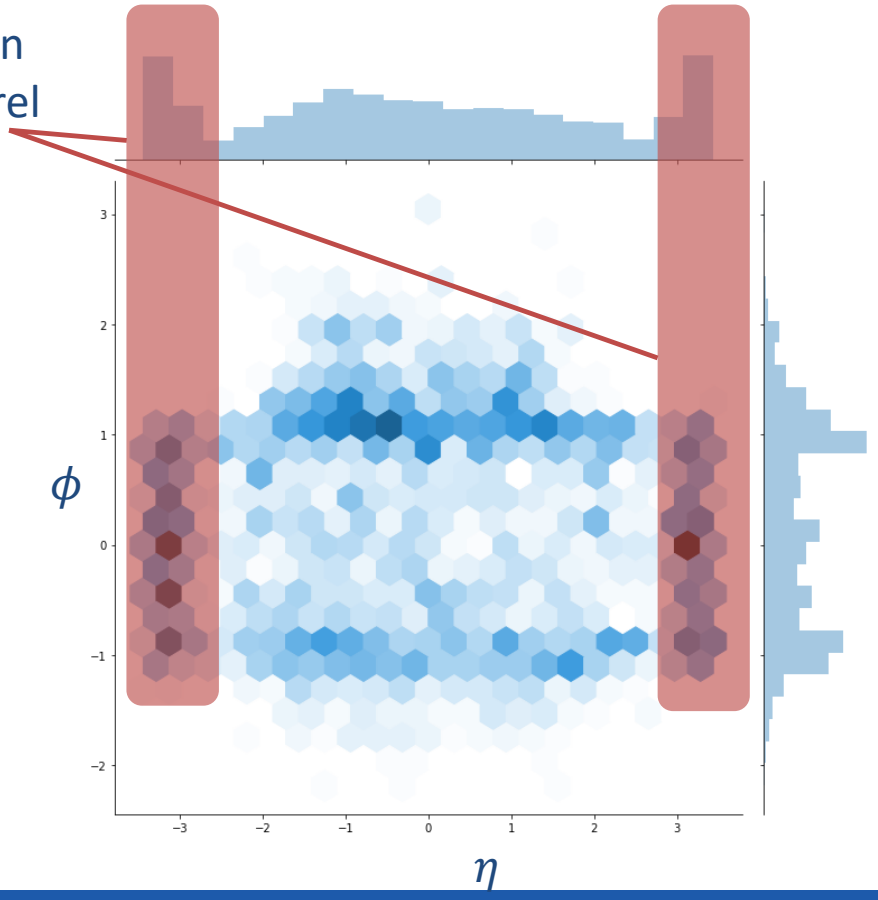
Missing Doublets

All Hits



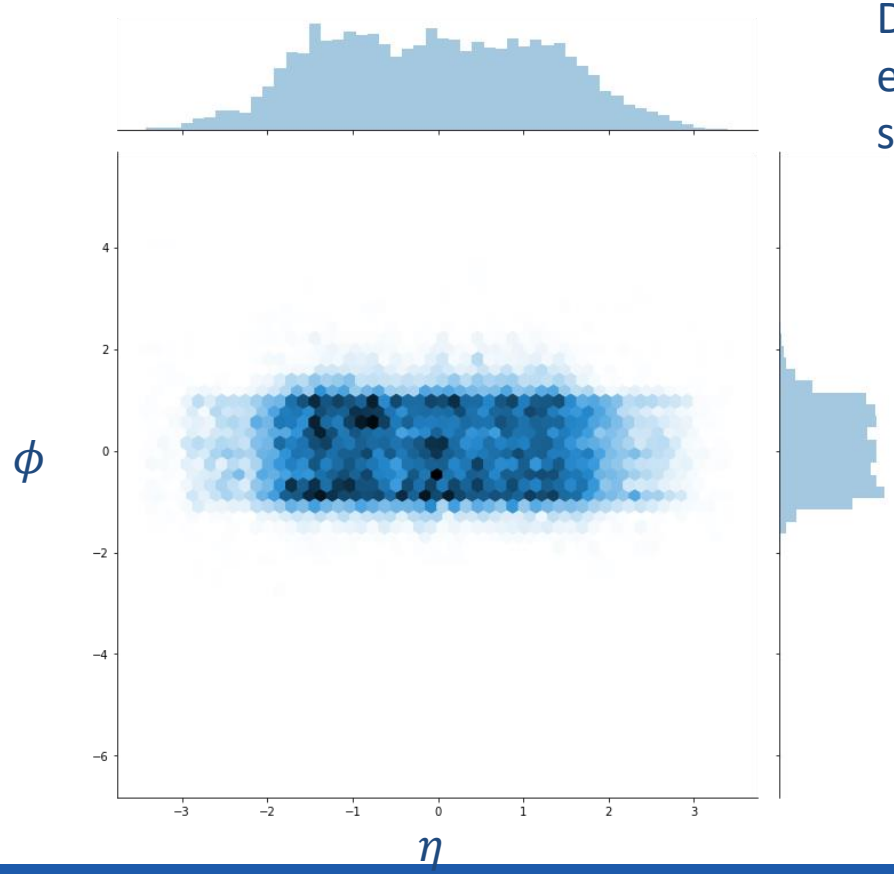
Missing Doublet Hits

Doublets on end of barrel



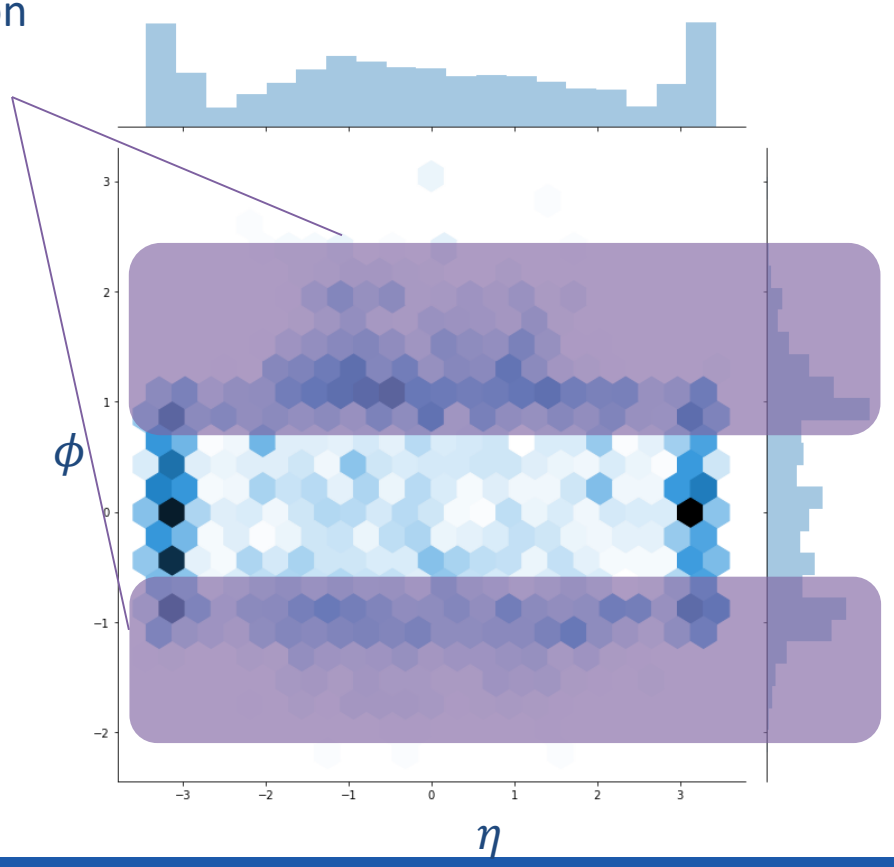
Missing Doublets

All Hits



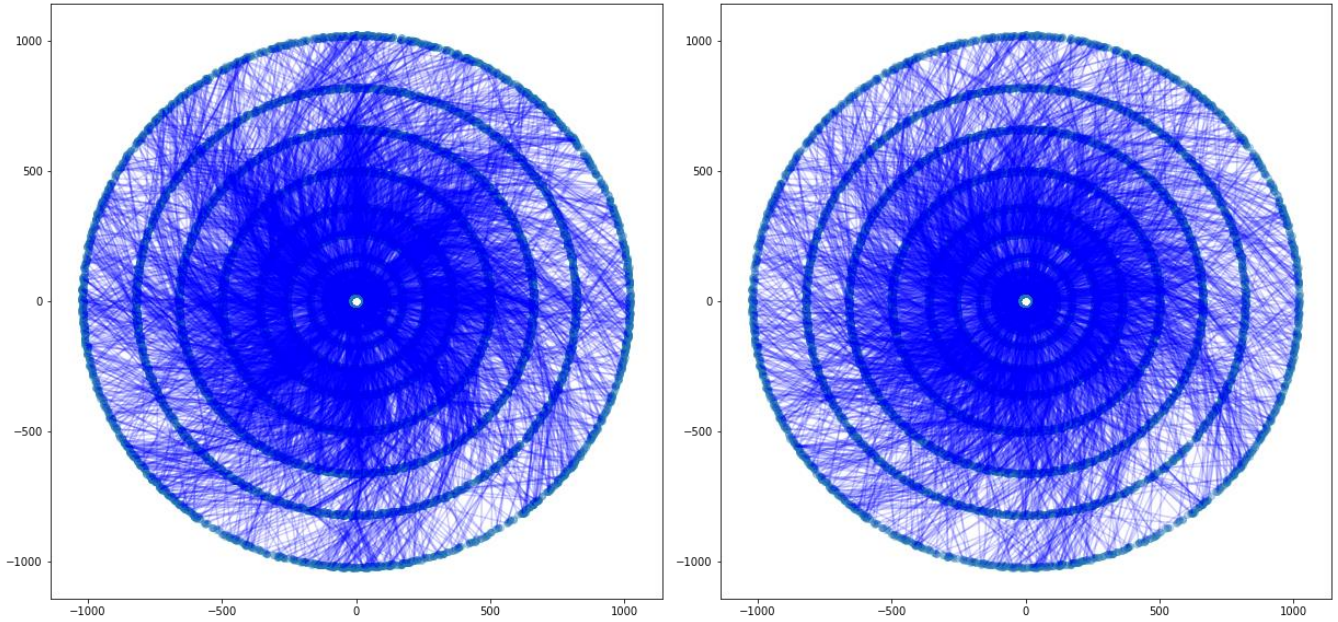
Missing Doublet Hits

Doublets on edge of segments



Stitching

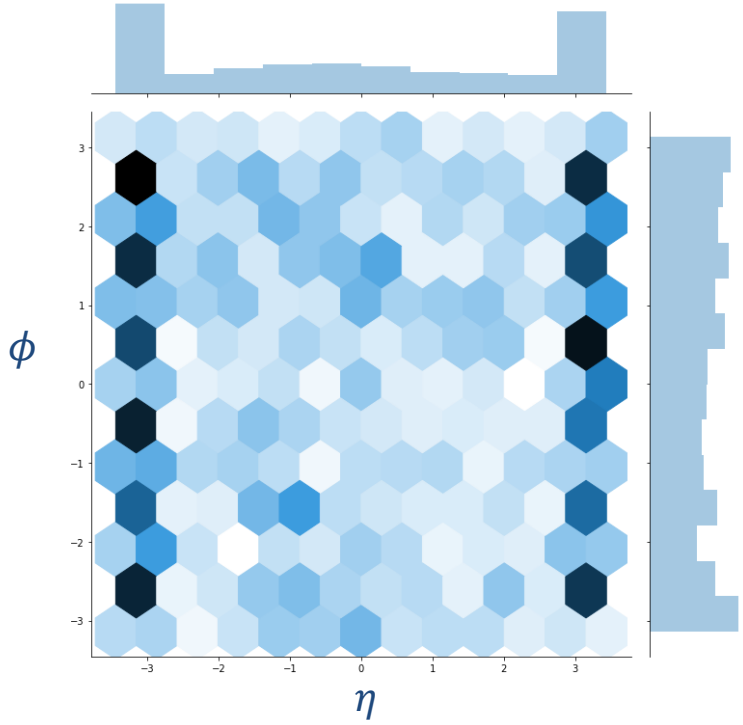
Significant speed up and lowered memory usage (~70%) from eliminated duplicates on edges of segments



Pre-clean-up

Post-clean-up

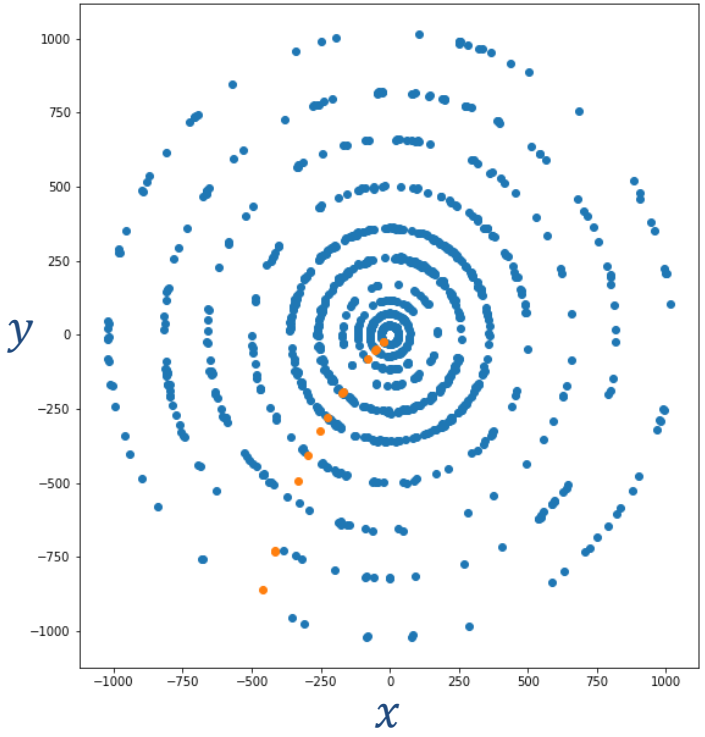
Missing doublet distribution



Periodic misclassification is now gone (would see peaks at each $\phi = \frac{n*\pi}{4}$

Ignoring Fragmented Tracks

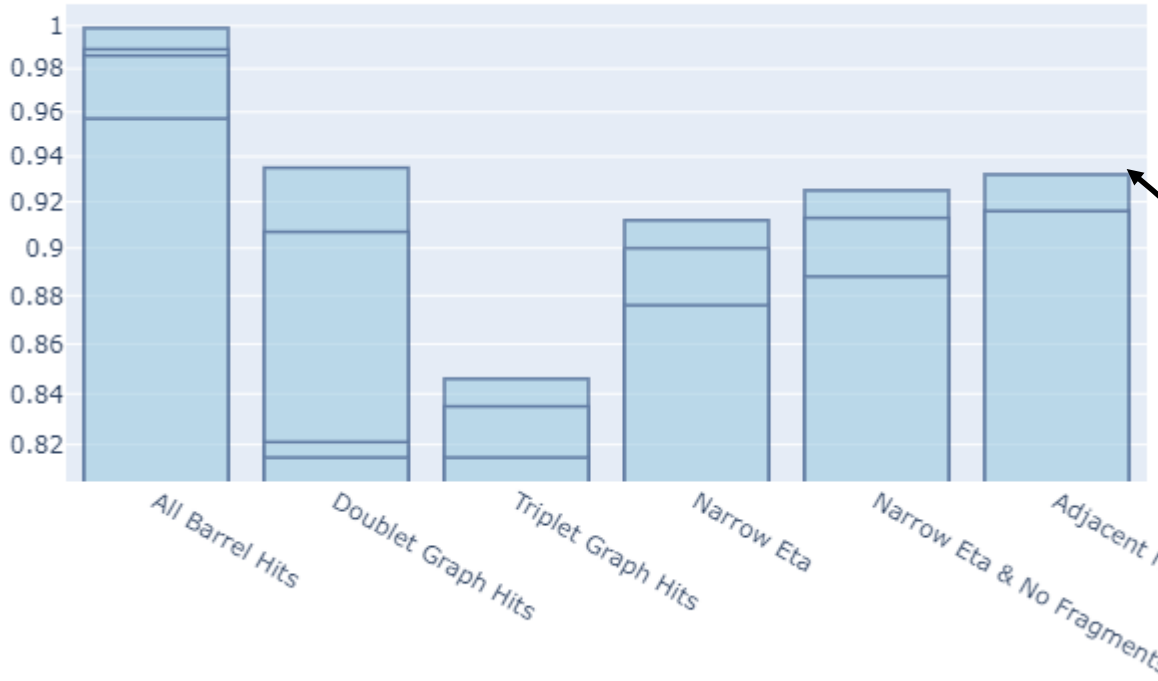
- We throw away all tracks that:
  - Have less than 4 non-duplicate layer hits
  - Skip a layer in the track (i.e. a track with layers [0,1,2,4] will be disregarded)



E.g. We throw away this track as it unreasonably punishes the barrel-only, adjacent-only GNN setup

Track Labelling: Final-ish Performance

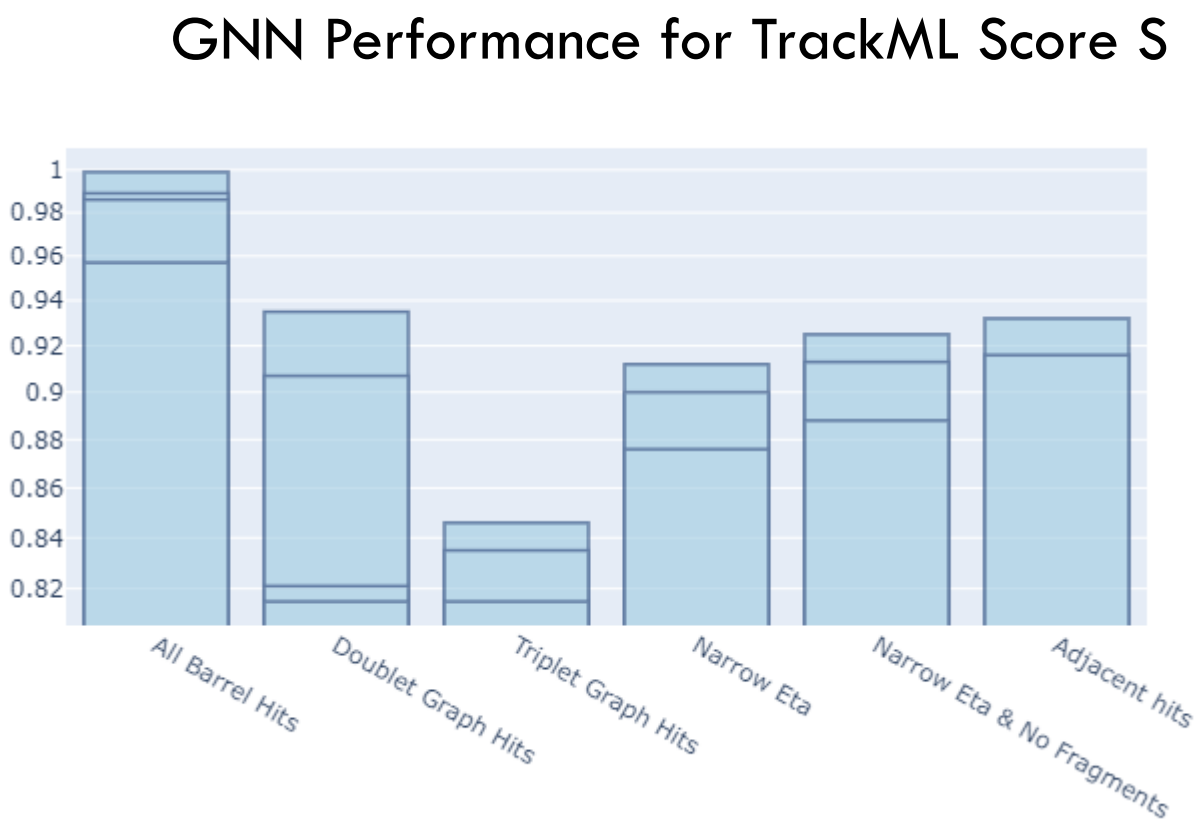
GNN Performance for TrackML Score S



This is the take-away

- Triplet graph truth in eta range (-2.1, 2.1)
  S = 0.912
- DBSCAN on triplet GNN classification in eta (-2.1, 2.1)
  S = 0.876
- DBSCAN on triplet GNN *at least 4 adjacent hits*
  S = 0.916
- **DBSCAN on triplet GNN *at least 5 adjacent hits***
  S = 0.932

Track Labelling: Final-ish Performance



- 0.932 TrackML Score in barrel, emulating whole detector (no punishment for tracks crossing detector volumes) recovers almost all missing doublets

- This is an early result – two big improvement areas are now seen:
  1. Doublet-to-triplet efficiency, and
  2. Embedding construction efficiency

- Every 1% of efficiency gained $\approx +\,0.015$ TrackML score
- Winning score is 0.922…

Summary

- Seeding pipeline complete, with good performance
- Track labelling just beginning, with promising performance
- Many low-hanging-fruit optimisations to try and boost efficiency and speed
  - HPO on embedding and GNN
  - Mixed-precision in GNN
  - Include cell features in GNN
  - Some GPU processing with CuPy, but much more could be transferred to work on GPU
  - A multitude of different GNN architectures, one may be especially suited to the physics