PHYSICS AND COMPUTING PERFORMANCE OF THE EXA.TRKX TRACKML PIPELINE

DANIEL MURNANE

ON BEHALF OF THE
EXA.TRKX COLLABORATION

THE EXATRKX PROJECT

MISSION

Optimization, performance and validation studies of ML approaches to Exascale track reconstruction

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
- *Illinois:* Markus Atkinson, Mark Neubauer
- *LBNL*: Paolo Calafiura (PI), Nicholas Choma, Sean Conlon, Steve Farrell, Xiangyang Ju, Daniel Murnane
- *ORNL*: Aristeidis Tsaris
- *Princeton*: Isobel Ojalvo, Savannah Thais
- *SLAC*: Pierre Cote De Soux, Francois Drielsma, Kasuhiro Terao, Tracy Usher

CODE

https://exatrkx.github.io/

PAPER

https://arxiv.org/abs/2103.06995

TRACK RECONSTRUCTION GOAL & DATASET

- **Track reconstruction physics goal**
  Given hits with features (space co-ordinates and cell cluster data) assign track labels to each hit.

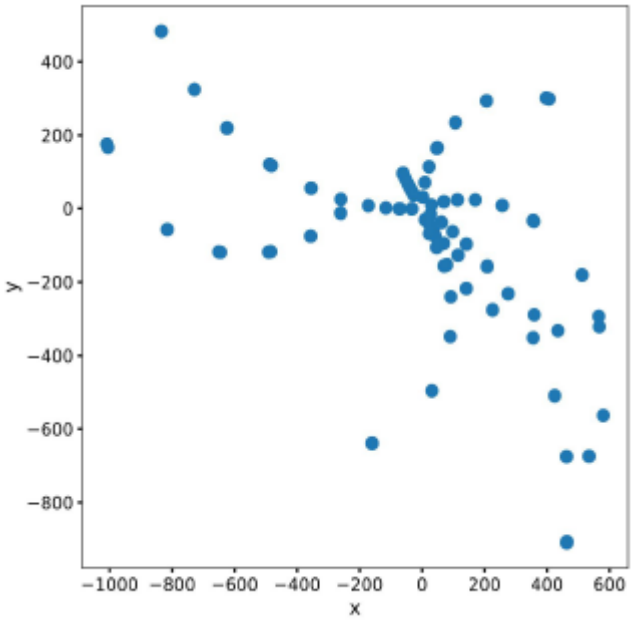
  Start of project: Good accuracy on TrackML barrel (middle of detector)

  Aim: Keep good accuracy, on whole detector, move to Itk geometry

- **Computing goal**
  Reconstruct tracks with low latency, high throughput

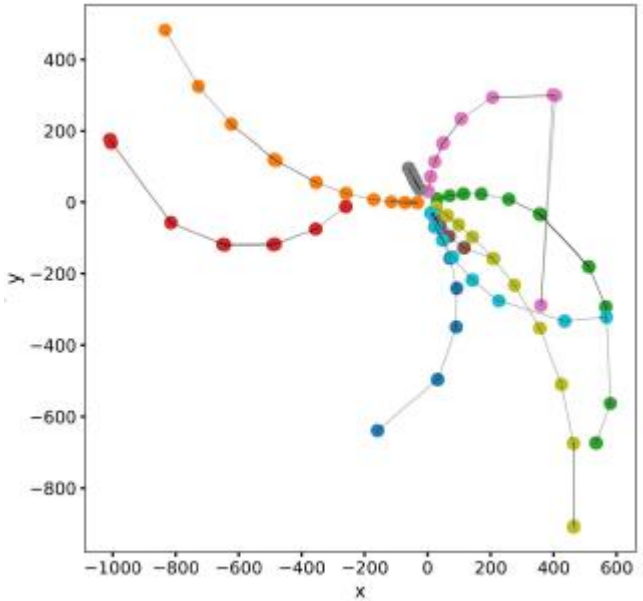
  Start of project: 32.1-second runtime for pipeline

  Aim: Sub-second runtume



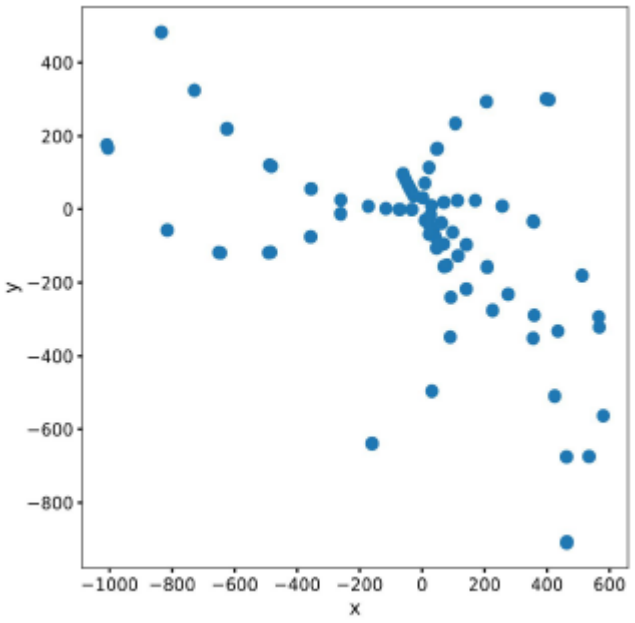
Label hits

= 32.1s

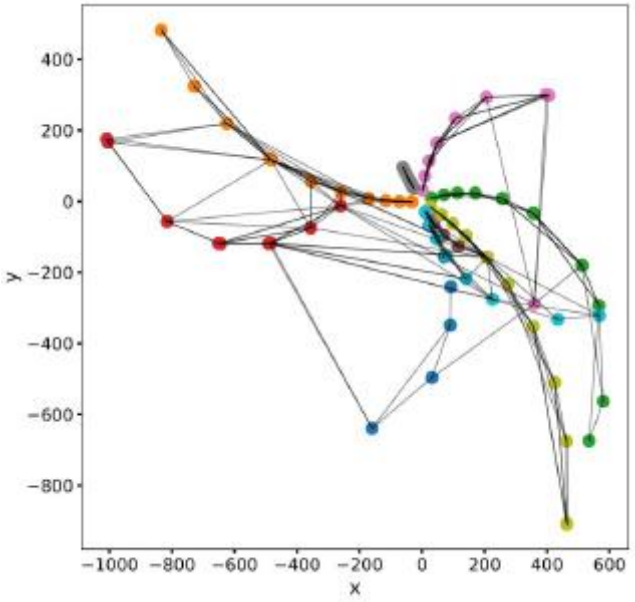


TrkX

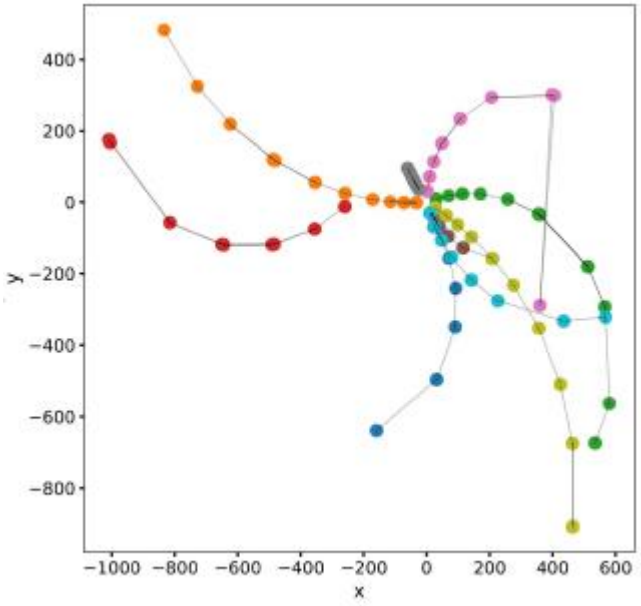
CURRENT LBNL EXATRKX PIPELINE



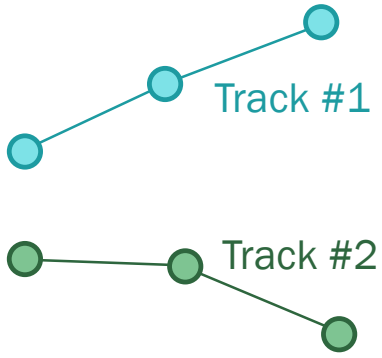
Construct graph



Classify edges



Identify connected components



- **Goal:** Given hits with features, return labels
- **Our approach:** Reframe as a graph classification & segmentation problem

TrkX

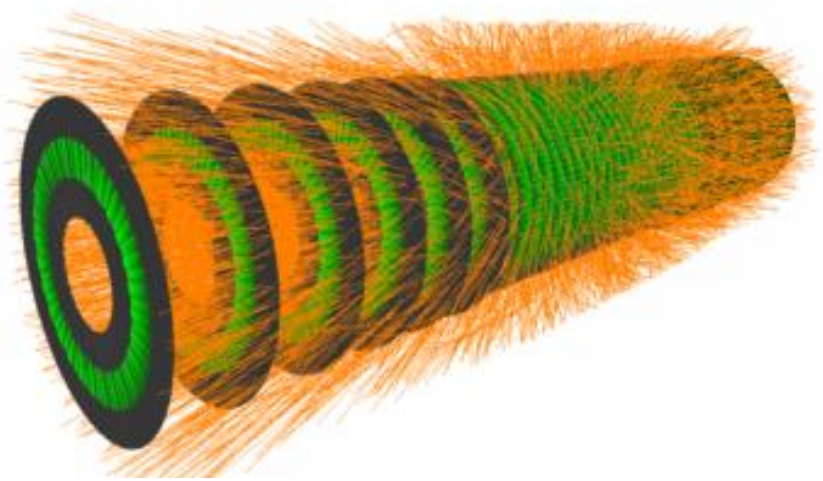
OUTLINE

1. Dataset
2. Details of Pipeline
   - Metric learning
   - Filter model
   - GNN architectures
   - Track labelling
3. Physics Performance
   - Edge eff/pur
   - Tracking eff/pur
   - Robustness to noise
   - Physics-motivated augmentations
4. Computing Performance
   - Timing studies
   - Scaling studies
   - Distributed training/inference
5. Future work
   - Itk geometry
   - ACTS integration
   - Model shrinking for inference (quantization, pruning, etc.)
   - Novel architectures: Pooling, Gravnet-style end-to-end contrastively trained GNN for labelling

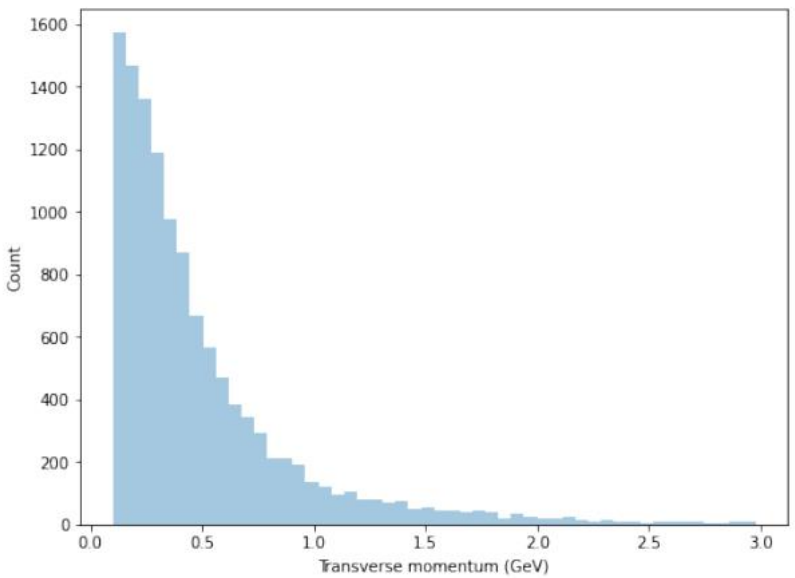
TrkX

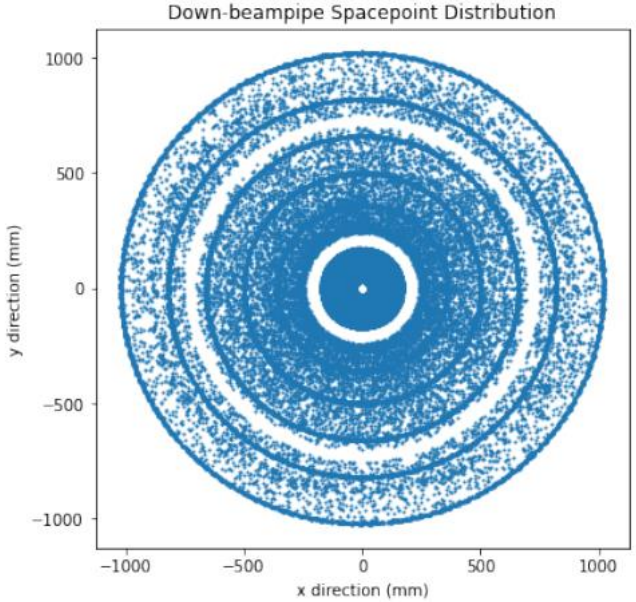
THE EXATRKX PIPELINE

TRACKML DATASET



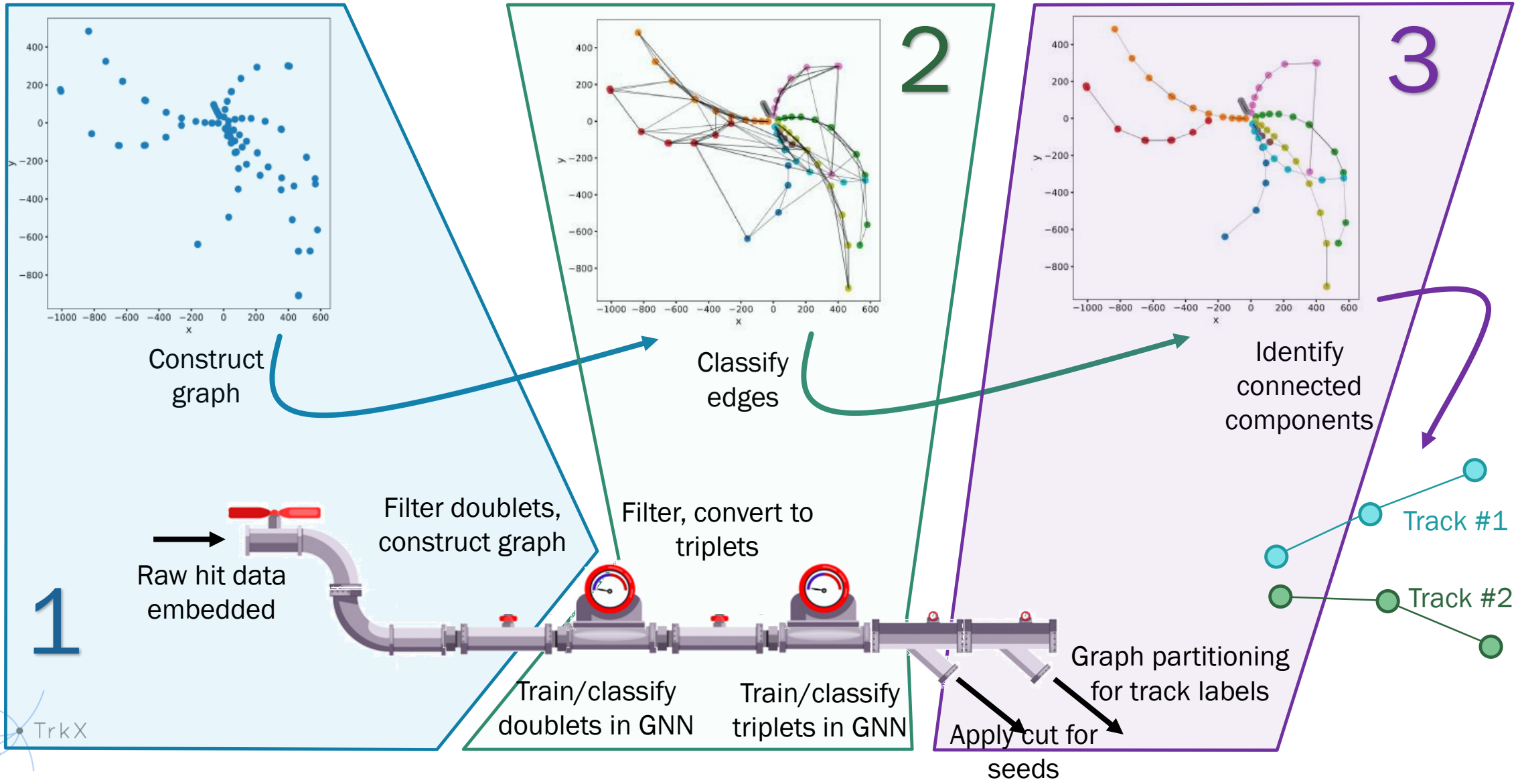
- 9000 events to train on
- Each event has up to 100,000 layer hits from around 10,000 particles
  = 200 pileup
- Layers can be hit multiple times by same particle (*duplicates*)
- Non-particle hits present (*noise*)
- Include module channel information (giving another 9 features per hit)





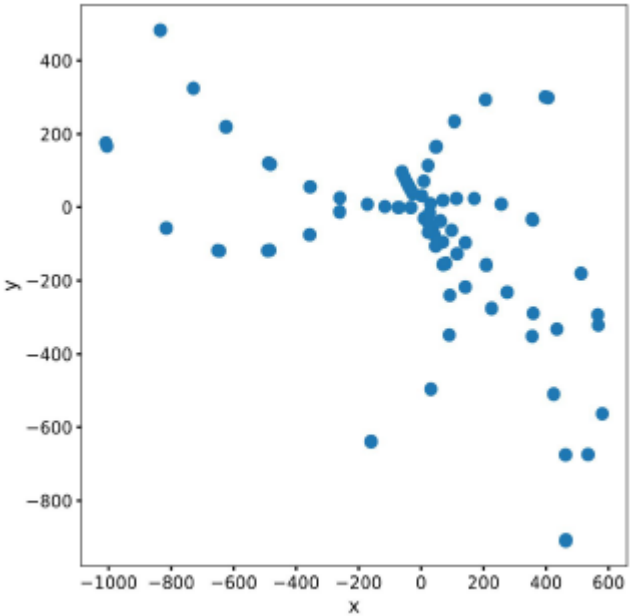
THE EXATRKX TRACK RECONSTRUCTION PIPELINE

Code available @ https://github.com/HSF-reco-and-software-triggers/Tracking-ML-Exa.TrkX

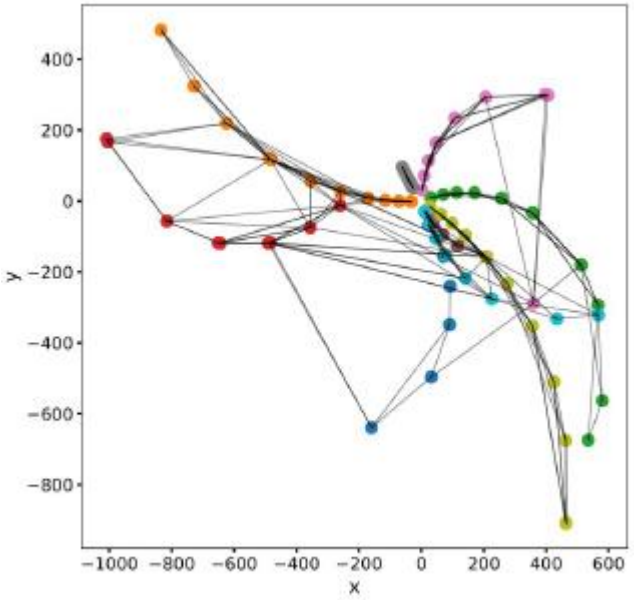


GRAPH CONSTRUCTION

- Want high-efficiency way to connect hits to likely neighbors
- Previously, connected with geometric heuristics (fast, but low efficiency)
- Studied look-up table from module-to-module connections (slower, higher efficiency)
- Why not train a model to *learn* the geometry, agnostic of module/layer configurations?



Construct graph



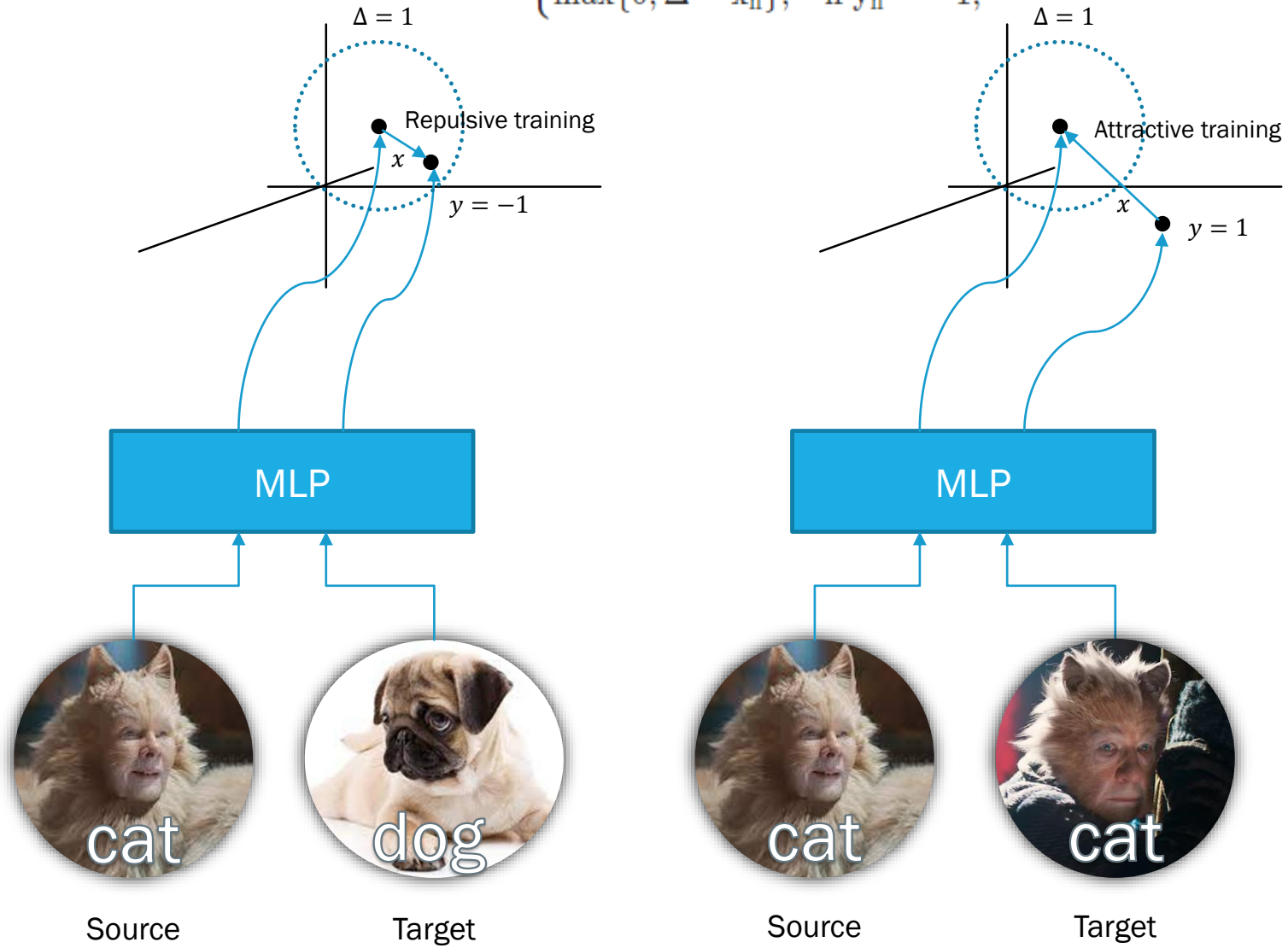
TrkX

METRIC LEARNING OVERVIEW

- Encode / embed input into N-dimensional space
- Reward (low loss) matching pairs within unit distance
- Punish (high loss) mismatching pairs within unit distance
- Repeat for many pairs

"Comparative" hinge loss

$$l_n = \begin{cases} x_n, & \text{if } y_n = 1, \\ \max\{0, \Delta - x_n\}, & \text{if } y_n = -1, \end{cases}$$



DEFINING "TRUE NEIGHBORS"

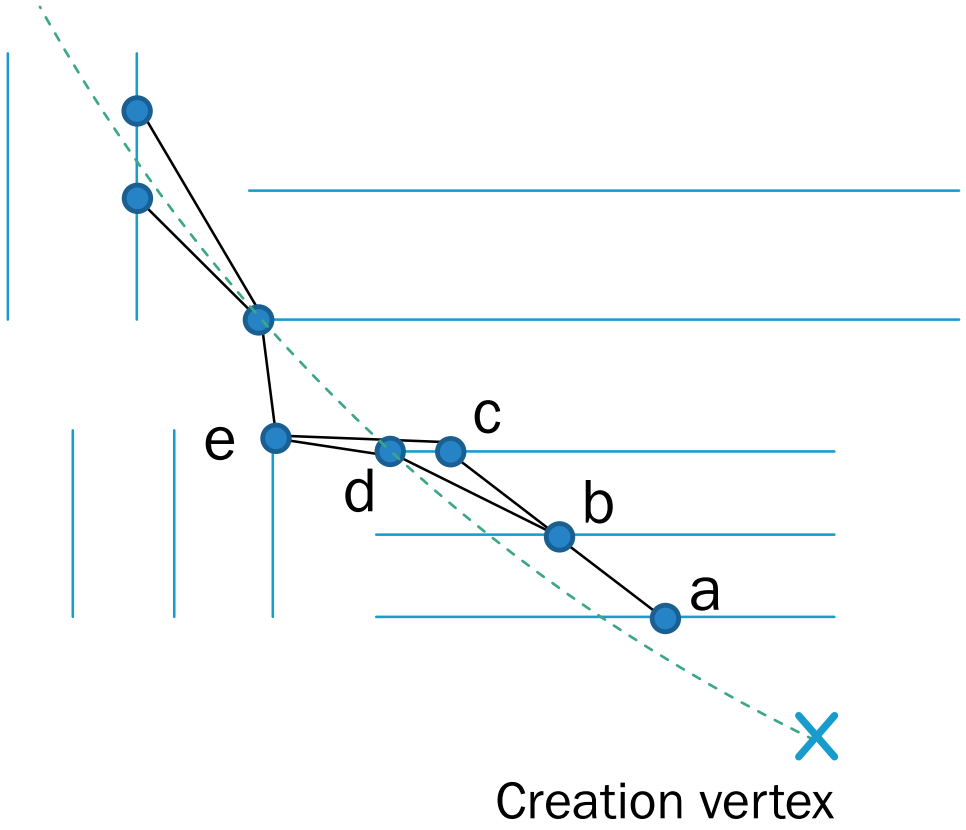
GEOMETRY-FREE GROUND TRUTH

1. For each particle, order hits by increasing distance from creation vertex,

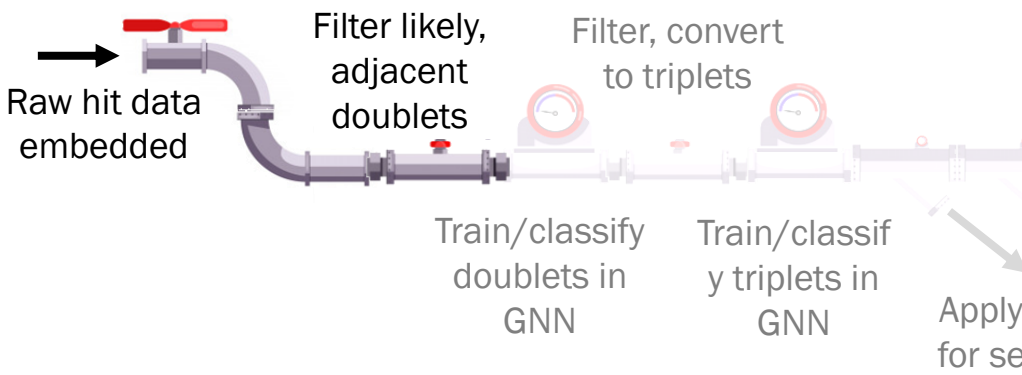
$$R = \sqrt{x^2 + y^2 + z^2}$$

2. Group by shared module ID

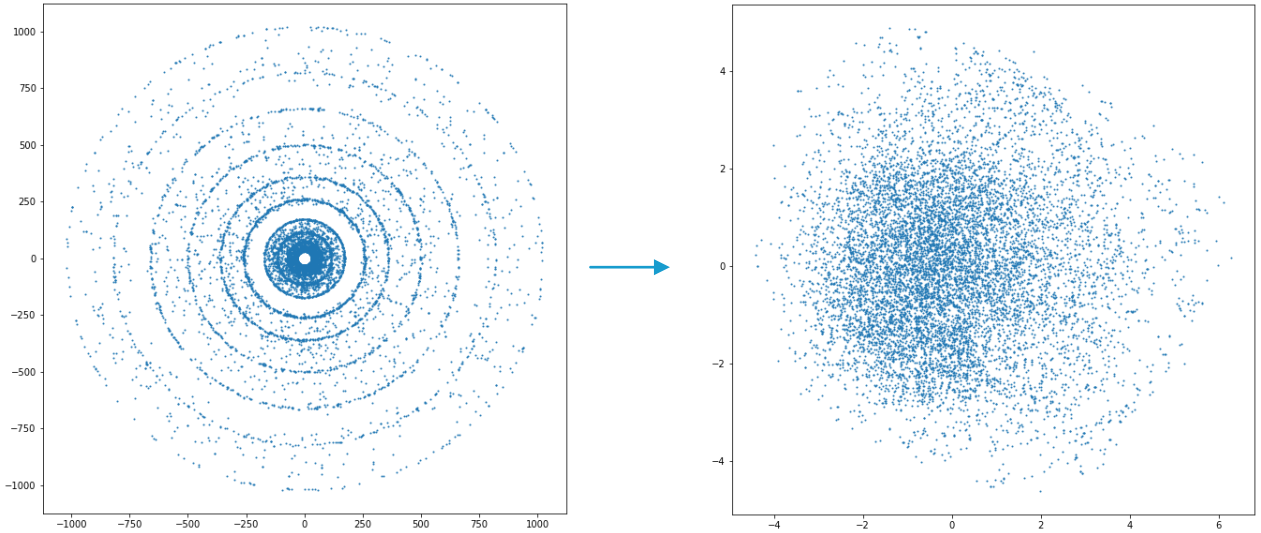
3. Connect all combinations from layer L_i to L_{i+1}, where $R_{i-1} < R_i < R_{i+1}$



METRIC LEARNING

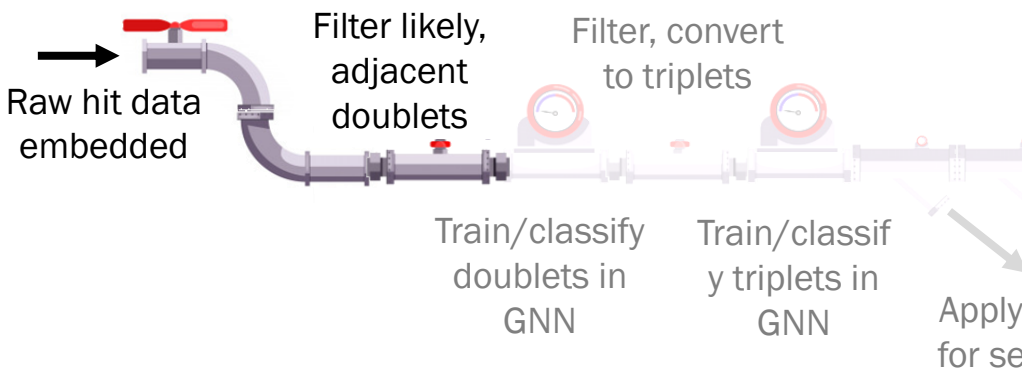


1. For all hits in detector, **embed** features (co-ordinates, cell direction data, etc.) into N-dimensional space

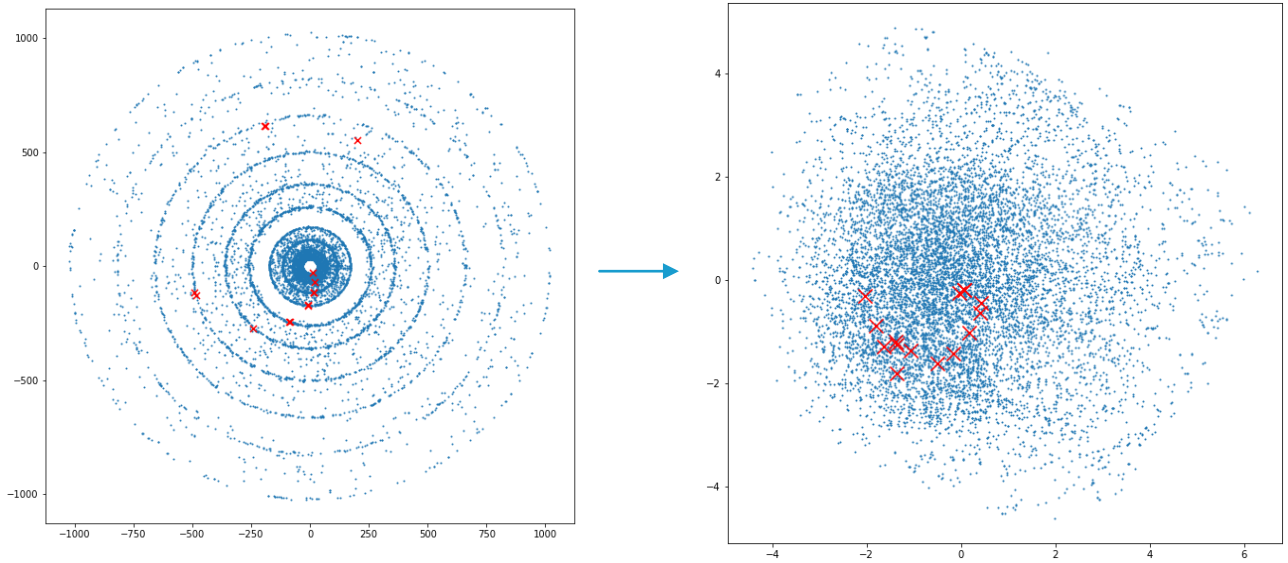


TrkX

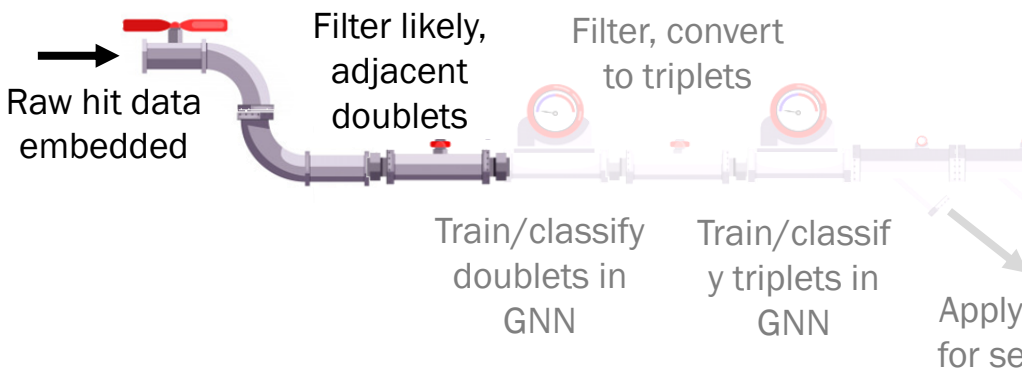
METRIC LEARNING



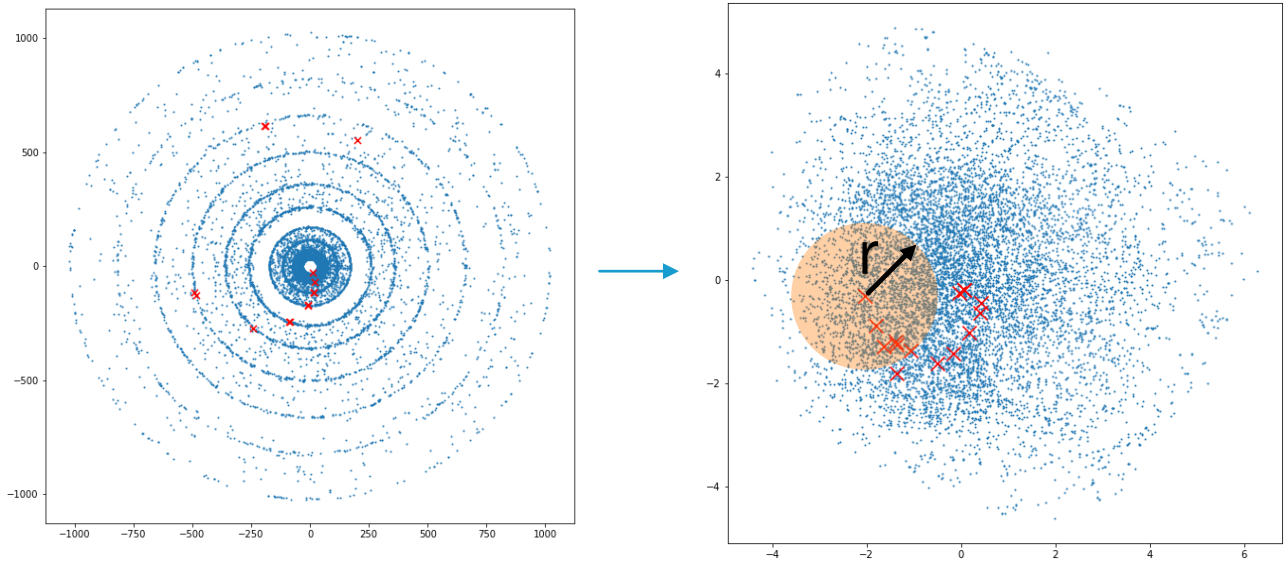
1. For all hits in detector, **embed** features (co-ordinates, cell direction data, etc.) with multi-layer perceptron (MLP) into N-dimensional space
2. **Associate neighboring** hits as close in N-dimensional distance (close = within Euclidean distance *r*)



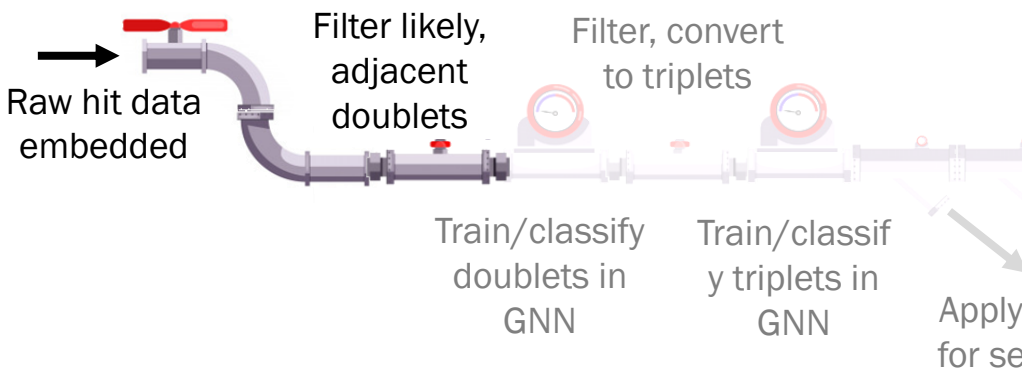
METRIC LEARNING



1. For all hits in detector, **embed** features (co-ordinates, cell direction data, etc.) with multi-layer perceptron (MLP) into N-dimensional space
2. **Associate neighboring** hits as close in N-dimensional distance **(close = within Euclidean distance *r*)**

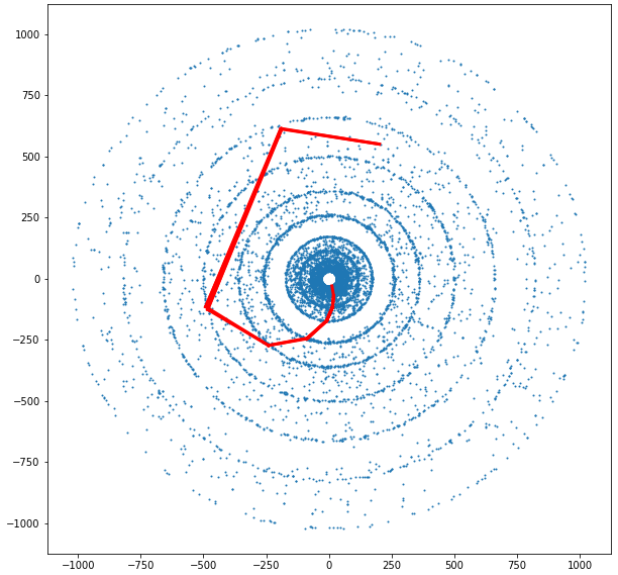


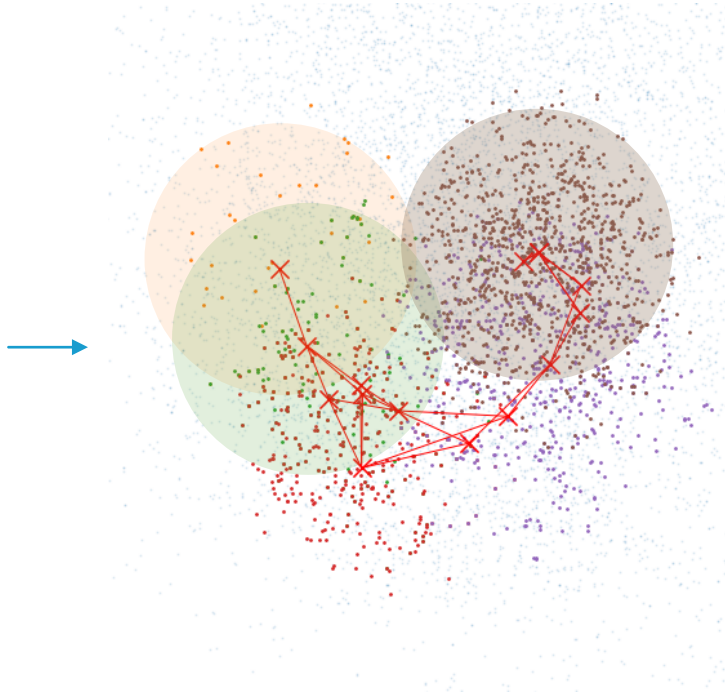
METRIC LEARNING



1. For all hits in detector, **embed** features (co-ordinates, cell direction data, etc.) with multi-layer perceptron (MLP) into N-dimensional space
2. **Associate neighboring** hits as close in N-dimensional distance
3. **Score** each "neighbour" hit *within embedding neighbourhood* against the "target" hit at centre

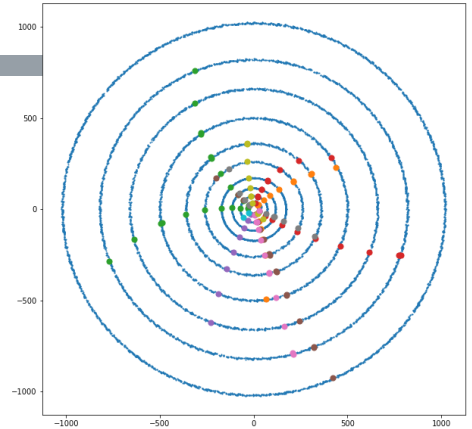
N.B. *Hard negative mining* can vastly improve training speed and accuracy)



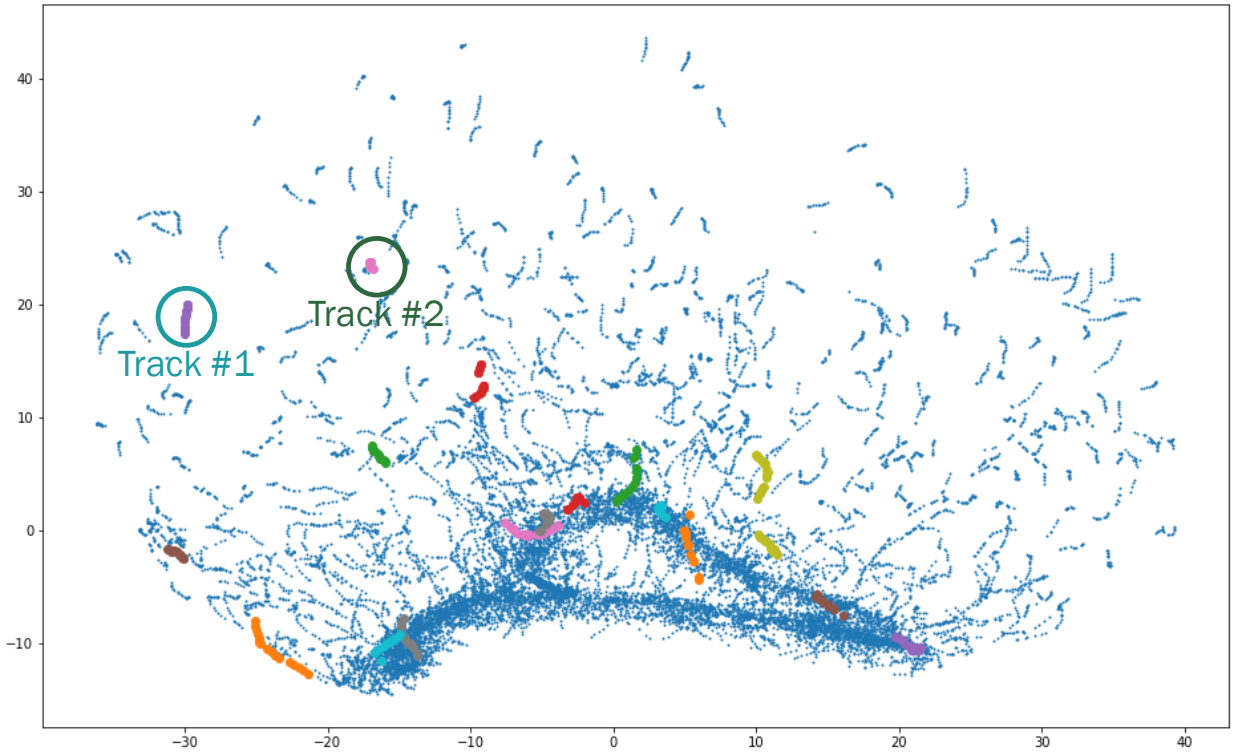


HITS IN THE EMBEDDED SPACE

Embedding
MLP



- Hits from the same track colored the same
- **For all hits,** we want to be able to join all neighboring hits within radius R (hyperparameter)
- All these connections lead to the input graph of the GNN
- **NOTE:** Here spacepoints are projected 8D → 2D



EMBEDDED SPACE IS BETTER THAN DETECTOR SPACE

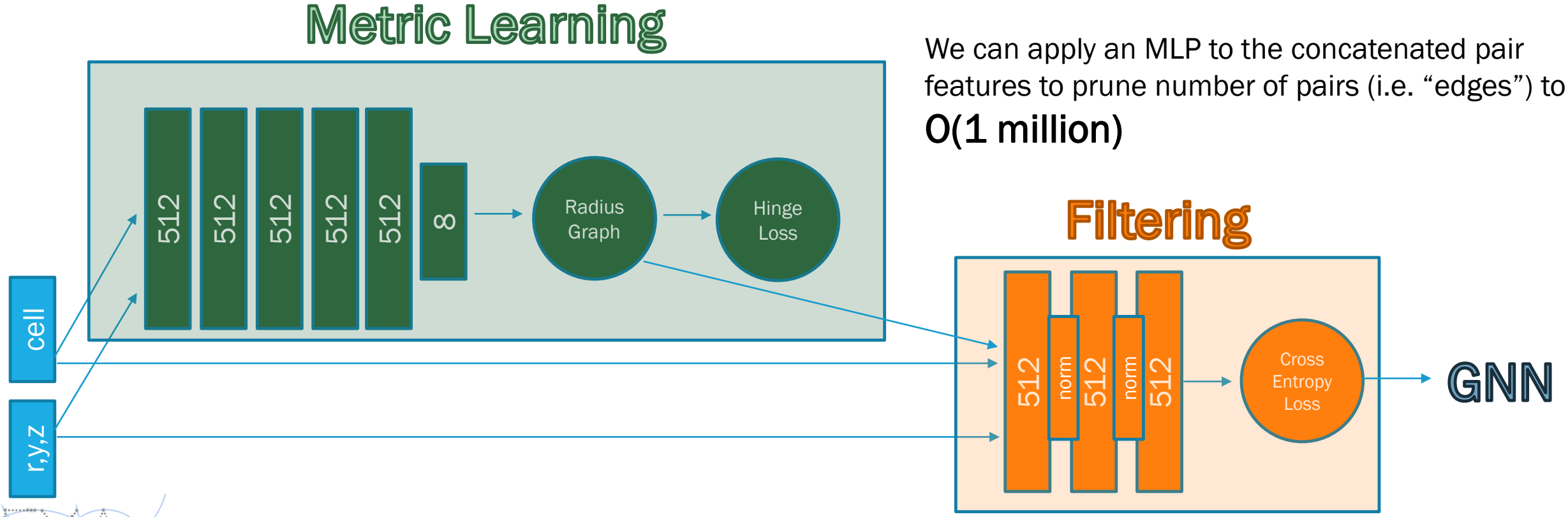
BUT STILL TOO MANY FAKE EDGES IN GRAPH

Number of neighbour pairs in embedding space:

O(10 million)

We can apply an MLP to the concatenated pair features to prune number of pairs (i.e. “edges”) to

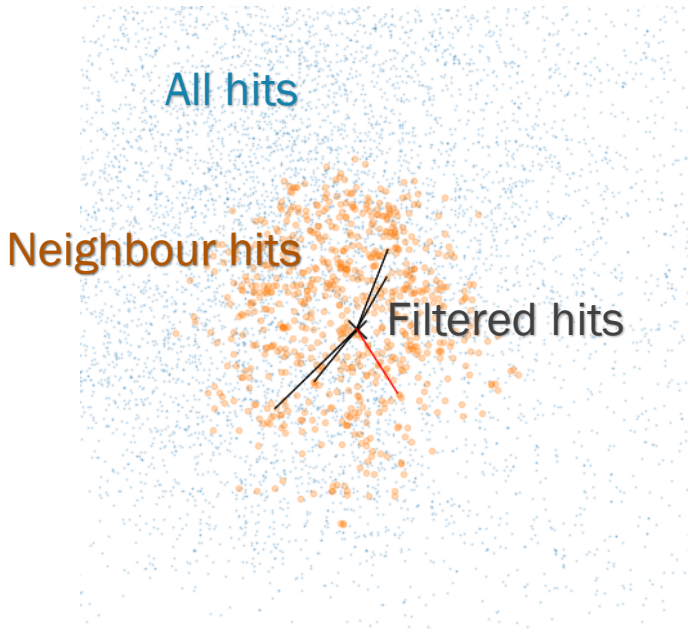
O(1 million)



TrkX

AN EXAMPLE
TOWARDS REALISTIC TRACKING

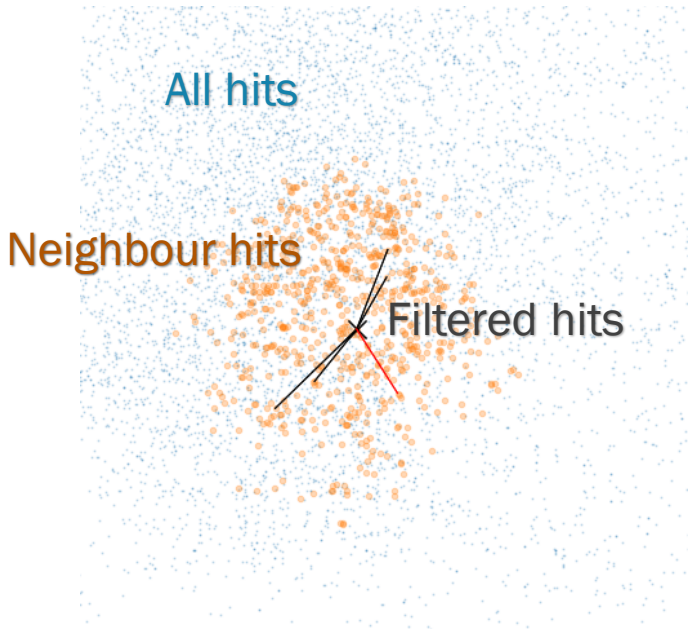
- Does it work? Let's check an example:



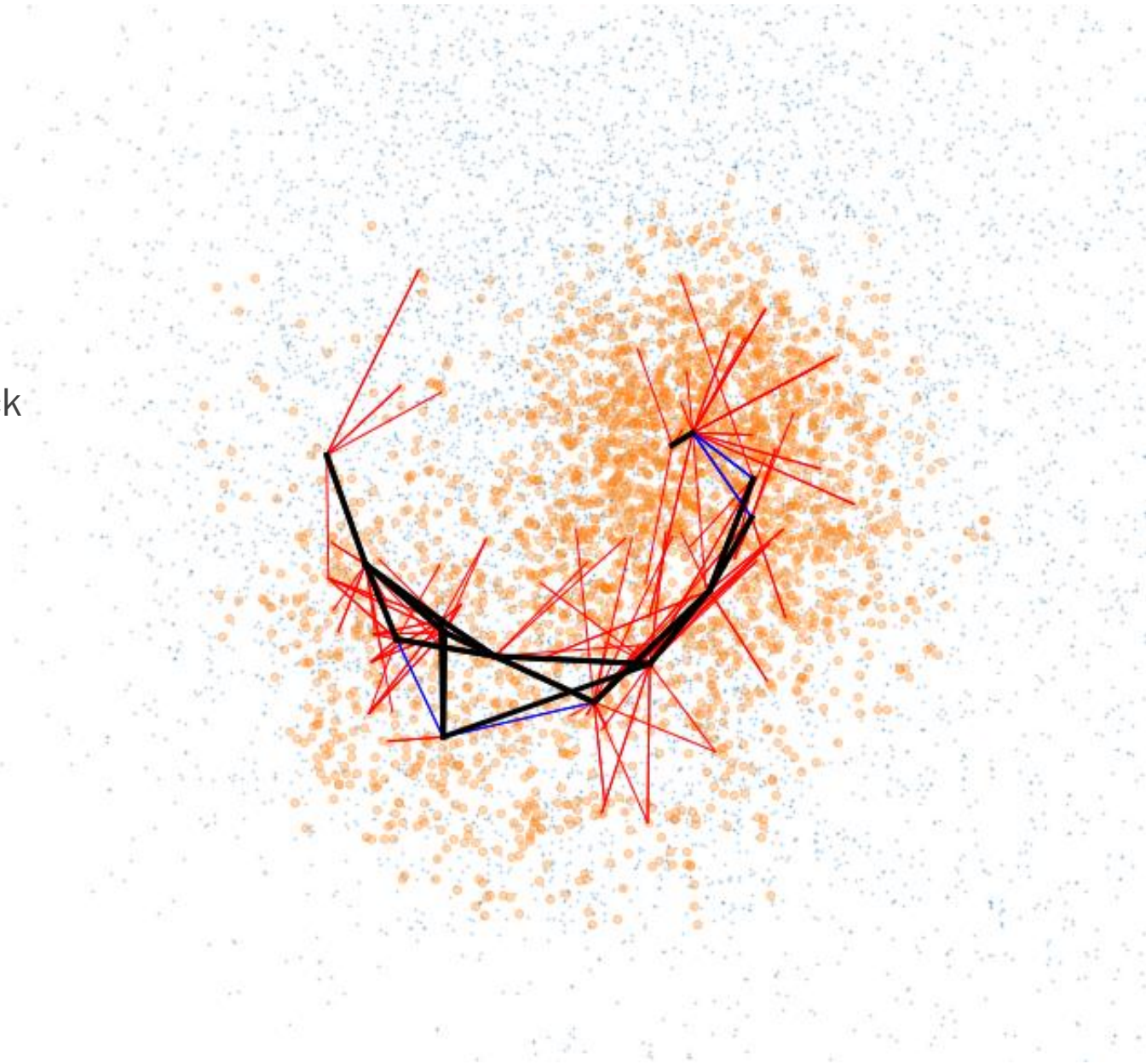
- Pretty good!
- True positives
- False positives
- No false negatives

AN EXAMPLE
TOWARDS REALISTIC TRACKING

- Does it work? Let's check an example:



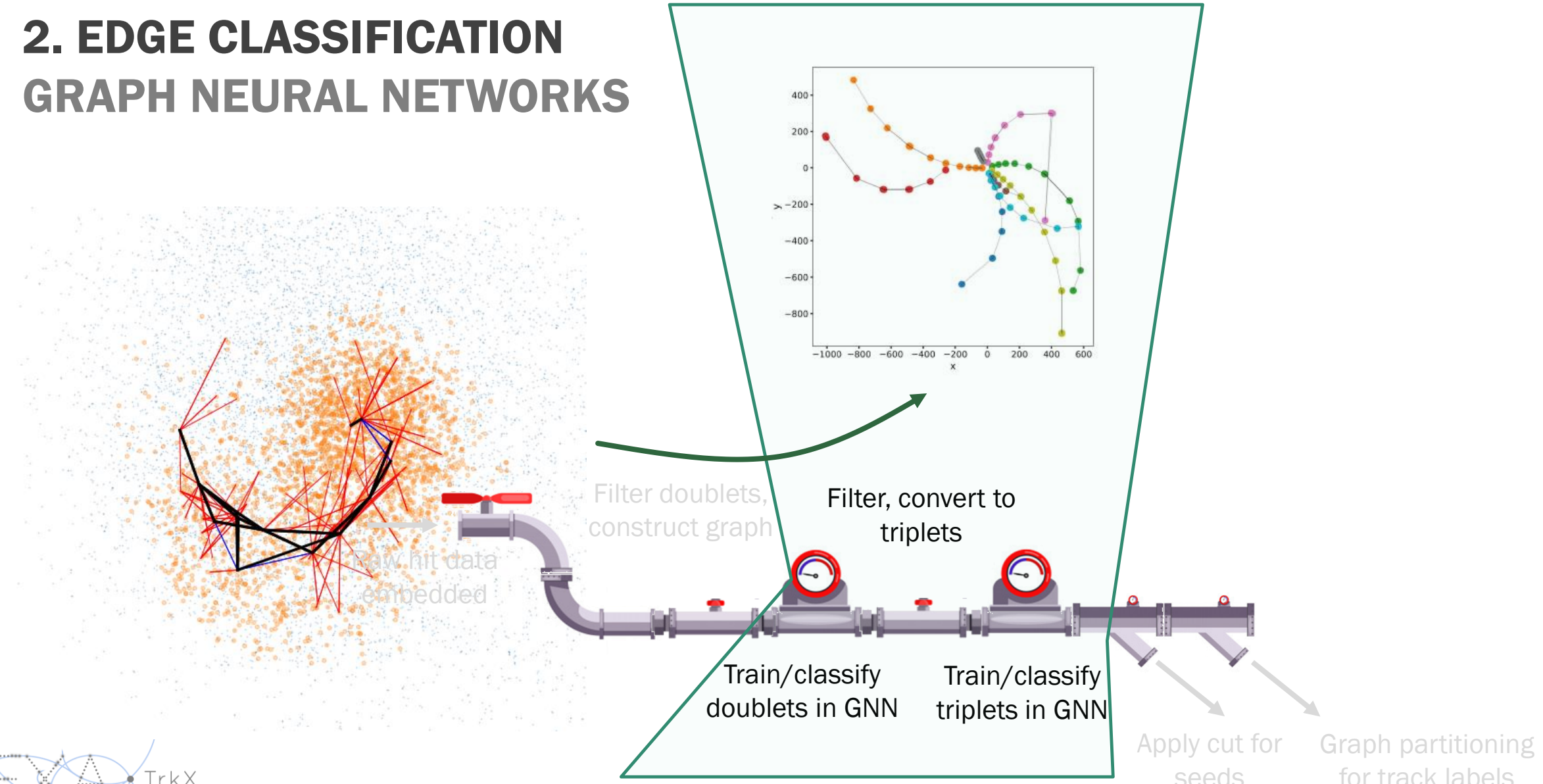
Whole example track



- Not quite as good..
- True positives
- False positives
- False negatives
- This is where GNN comes in

TrkX

2. EDGE CLASSIFICATION
GRAPH NEURAL NETWORKS

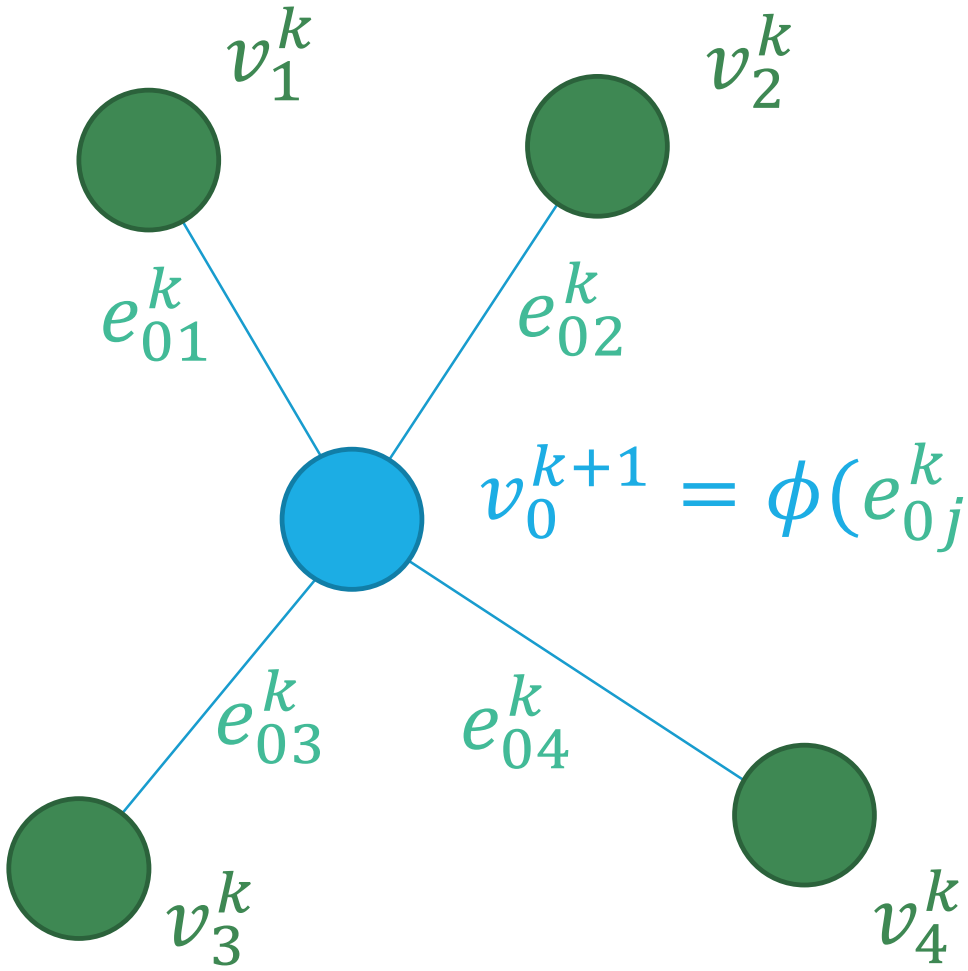


TrkX

GRAPH NEURAL NETWORKS

ARCHITECTURES

v_i^k node features
e_{ij}^k edge features
at iteration k

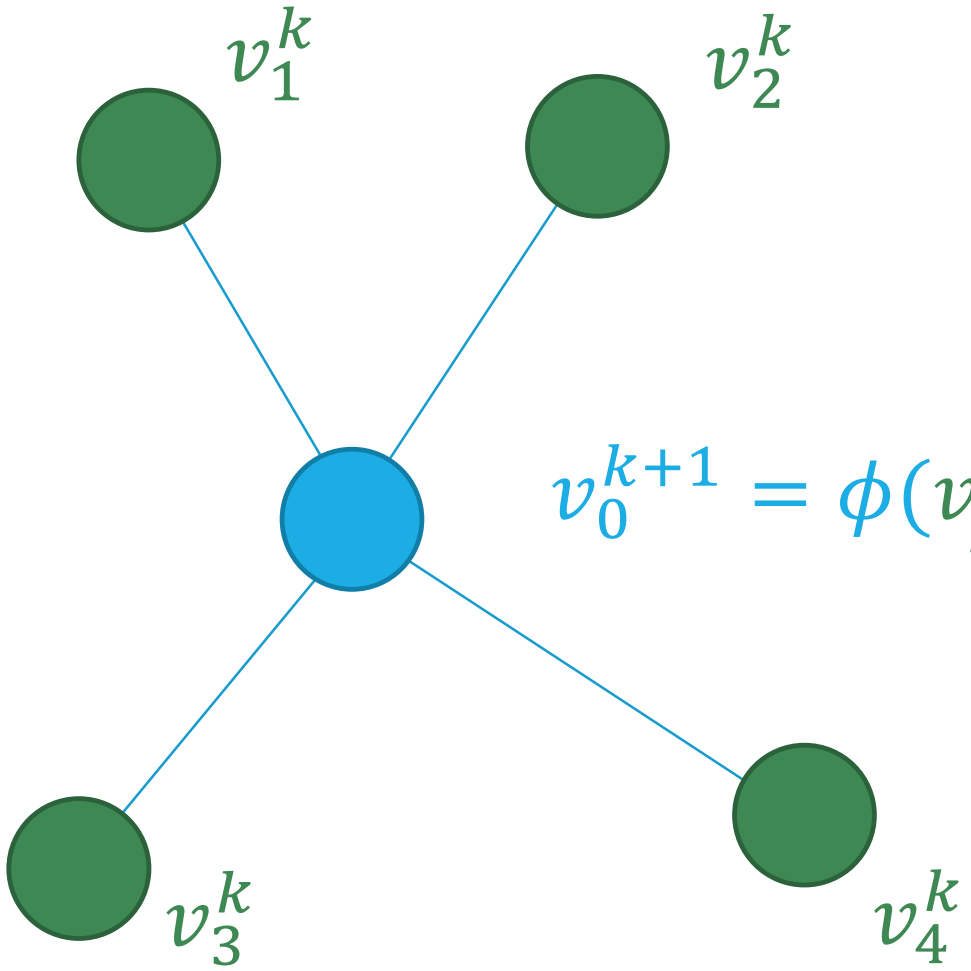


$$v_0^{k+1} = \phi(e_{0j}^k, v_j^k, v_0^k)$$

MESSAGE PASSING

GRAPH NEURAL NETWORKS
ARCHITECTURES

v_i^k node features at iteration k



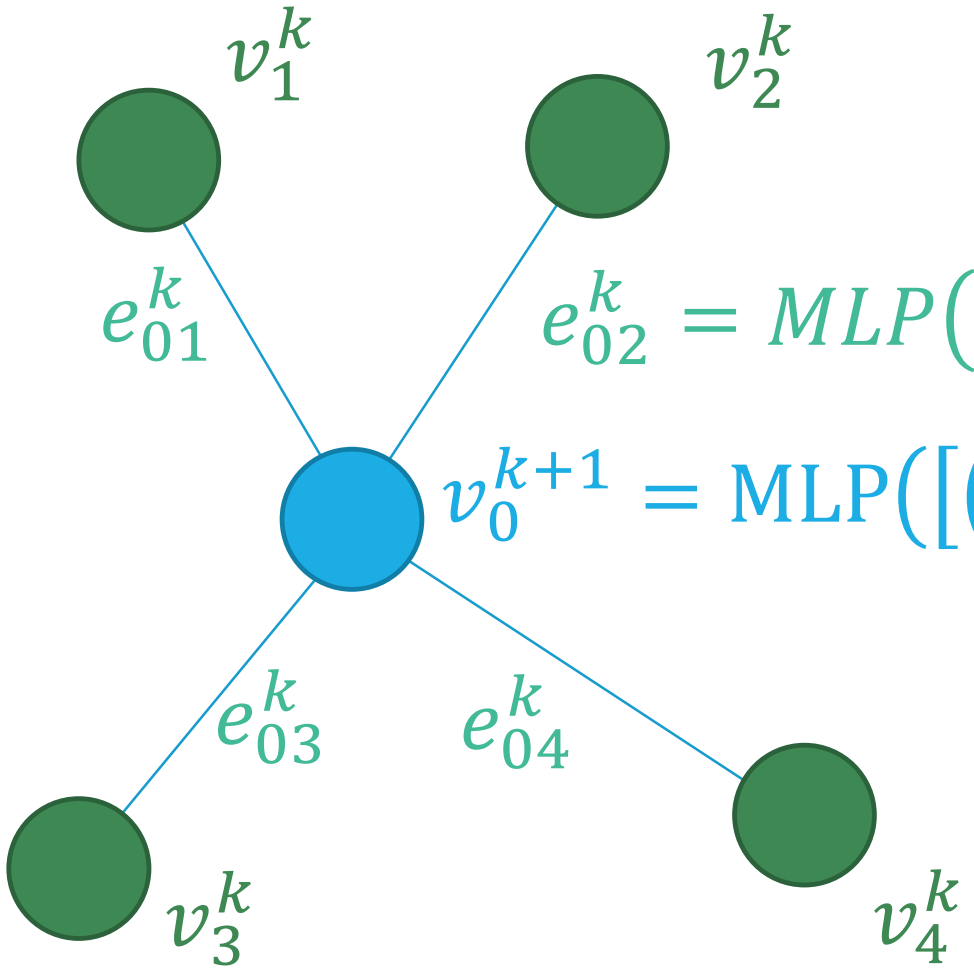
$$v_0^{k+1} = \phi(v_j^k, v_0^k)$$

GRAPH CONVOLUTION NETWORK

Kipf & Welling "Semi-Supervised Classification with Graph Convolutional Networks" *arXiv:1609.02907* (2016).

GRAPH NEURAL NETWORKS
ARCHITECTURES

v_i^k node features
e_{ij}^k edge **score** [0, 1]
at iteration k



$$e_{02}^k = MLP\left([v_0^k, v_2^k]\right)$$

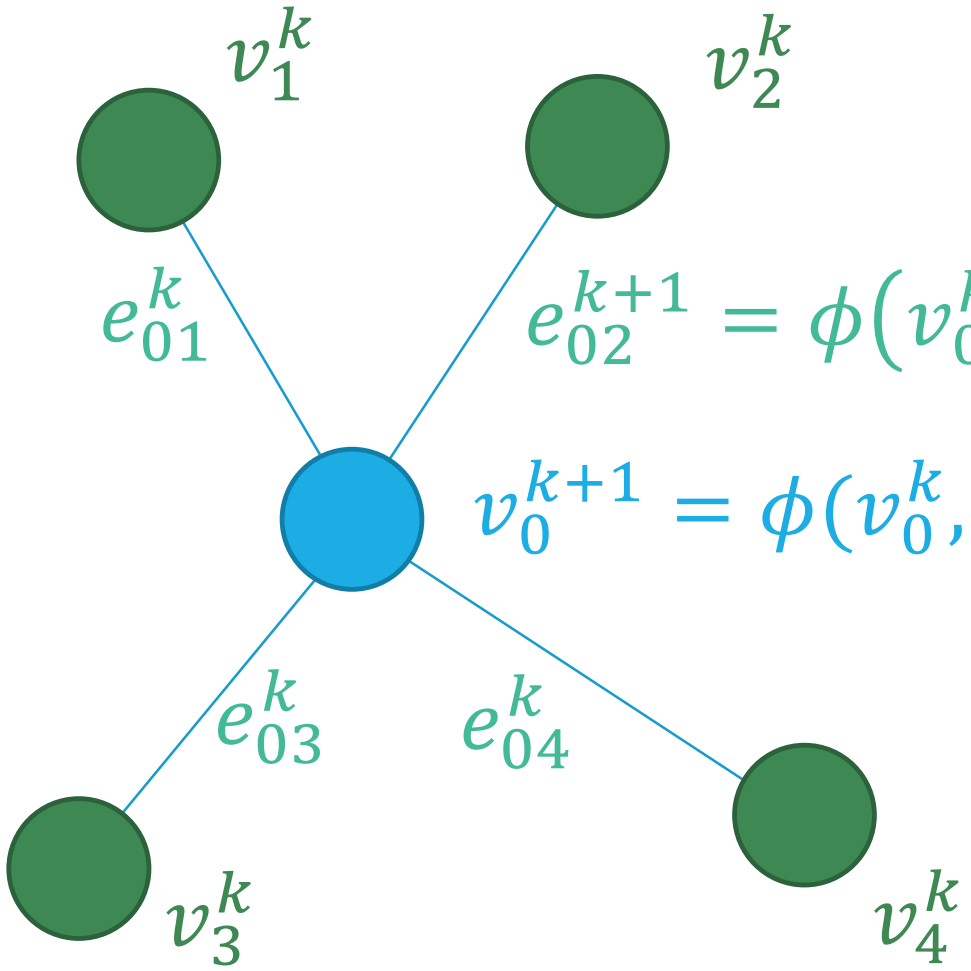
$$v_0^{k+1} = \mathrm{MLP}\left(\left[\left(\Sigma e_{0j}^k v_j^k\right), v_0^k\right]\right) + v_0^k$$

ATTENTION GNN

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

GRAPH NEURAL NETWORKS
ARCHITECTURES

v_i^k node features
e_{ij}^k edge features
at iteration k



$$e_{02}^{k+1} = \phi\left(v_0^k, v_2^k, e_{02}^k\right)$$

$$v_0^{k+1} = \phi(v_0^k, \Sigma e_{0j}^{k+1})$$

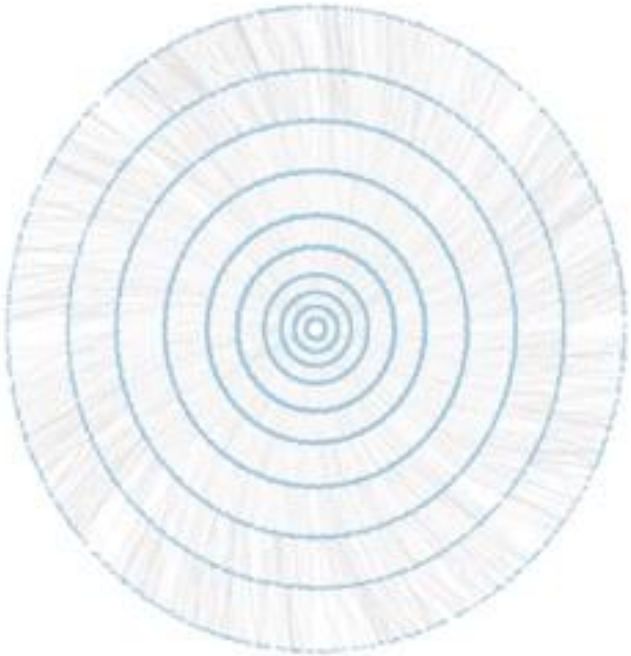
INTERACTION NETWORK

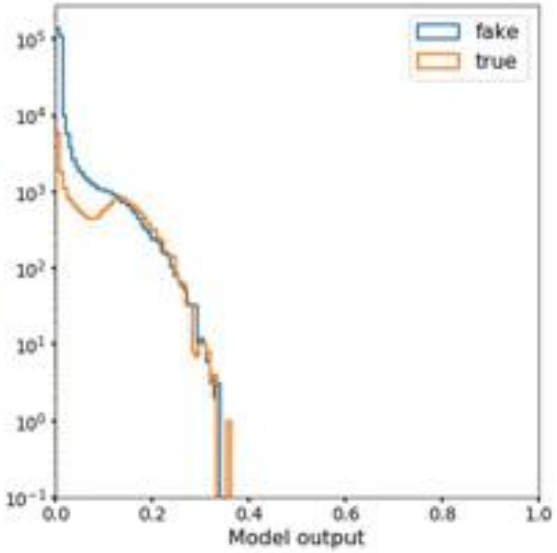
Battaglia, Peter, et al. "Interaction networks for learning about objects, relations and physics." *Advances in neural information processing systems*. 2016.

TrkX

APPLY GNN TO TRACKING EDGE CLASSIFICATION

1. **(r, phi, z) features** are encoded (n.b. we do not use the embedded space features, as they appear to not help performance)
2. Encoded features are concatenated to create edge features
3. Edge features are aggregated around nodes to create next round of encoded node features (i.e. message passing)
4. Each iteration of message passing improves distinguishing power

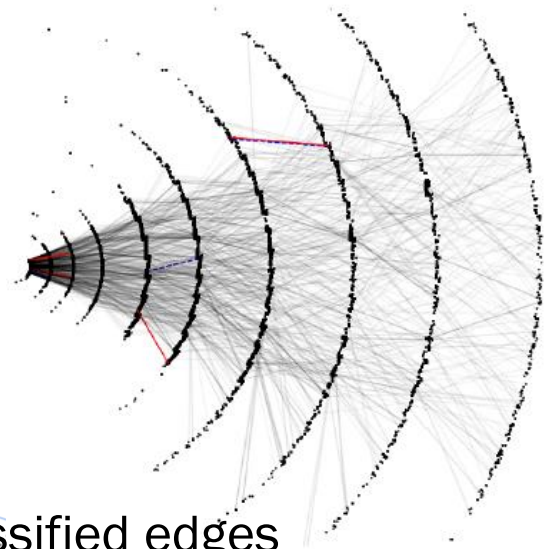




TrkX

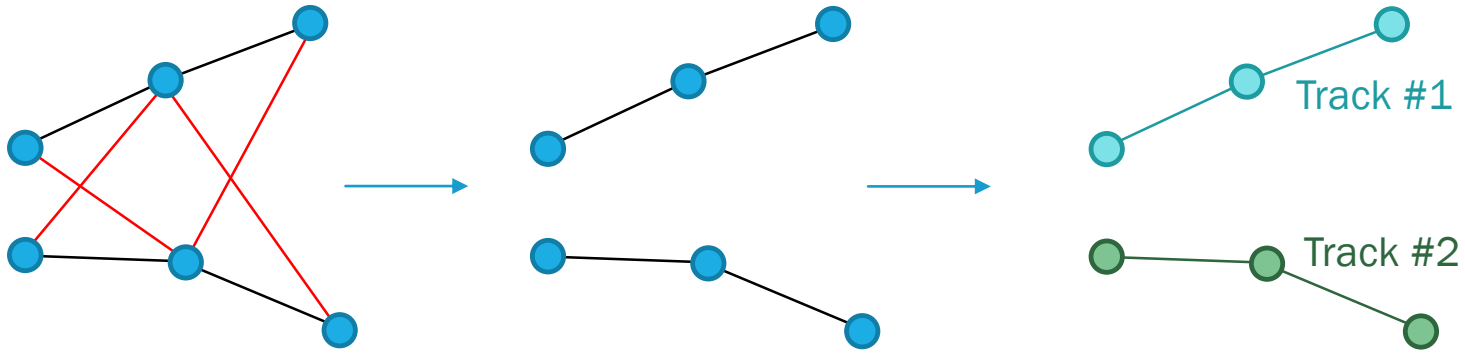
CLASSIFIED EDGES TO TRACK LABELS

- CuGraph used to label tracks after converting Tensorflow/Pytorch tensors (on-GPU)
- `connected_components` algorithm returns a component label for each vertex



Classified edges

=



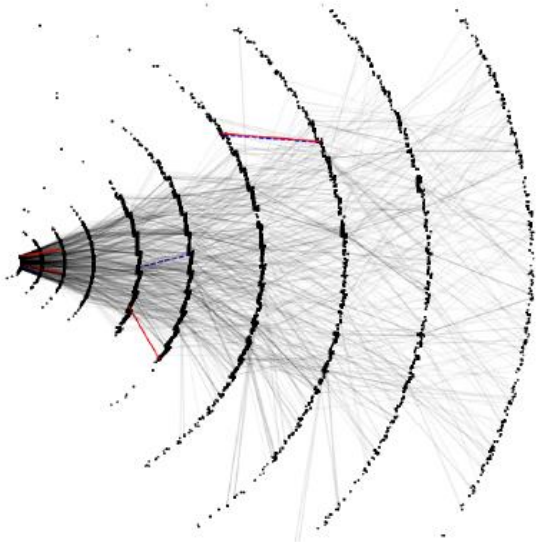
Classified edges

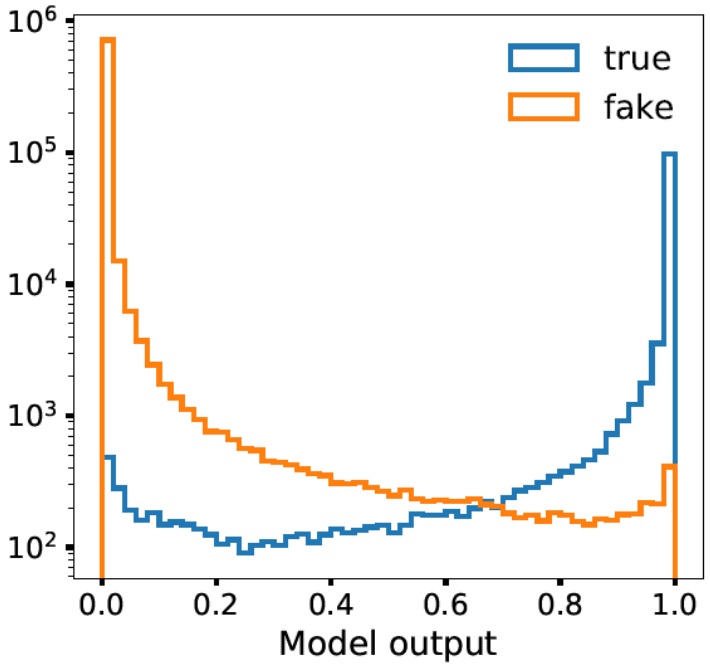
Ignore cut edges

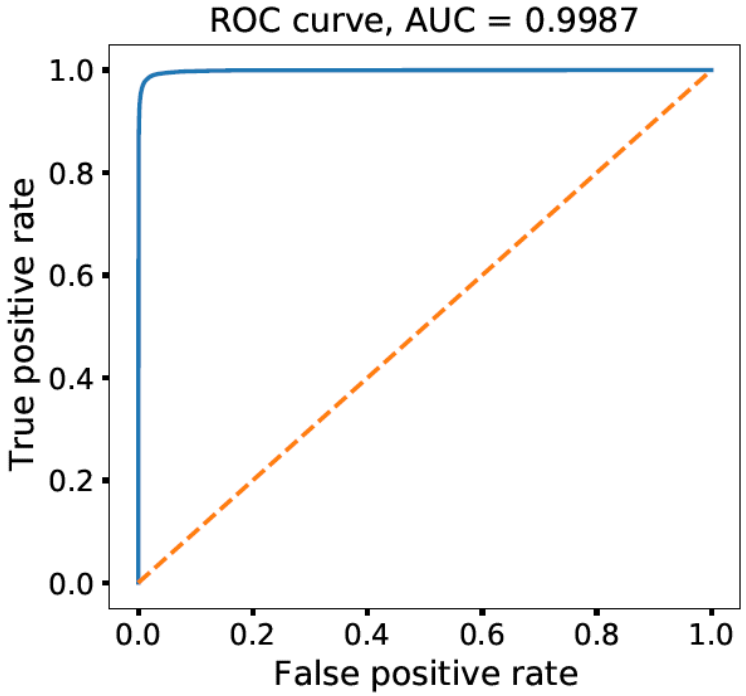
Labelled connected components

PHYSICS PERFORMANCE & EXTENSION

PERFORMANCE – EDGE EFFICIENCY & PURITY







PERFORMANCE – TRACKING DEFINITIONS

- Pile-up: 200
- Physics cuts: {Charged particles, $|\eta| < 4.0$, $p_T > 100\text{MeV}$}
- *Selected:* Particles passing physics cuts
- *Reconstructable:* Particles with truth tracks of at least five spacepoints (s.p.)
- *Matched track:* A given track satisfies a double majority condition ($> 50\%$ s.p. reconstructed are true, **and** $> 50\%$ s.p. true are reconstructed)

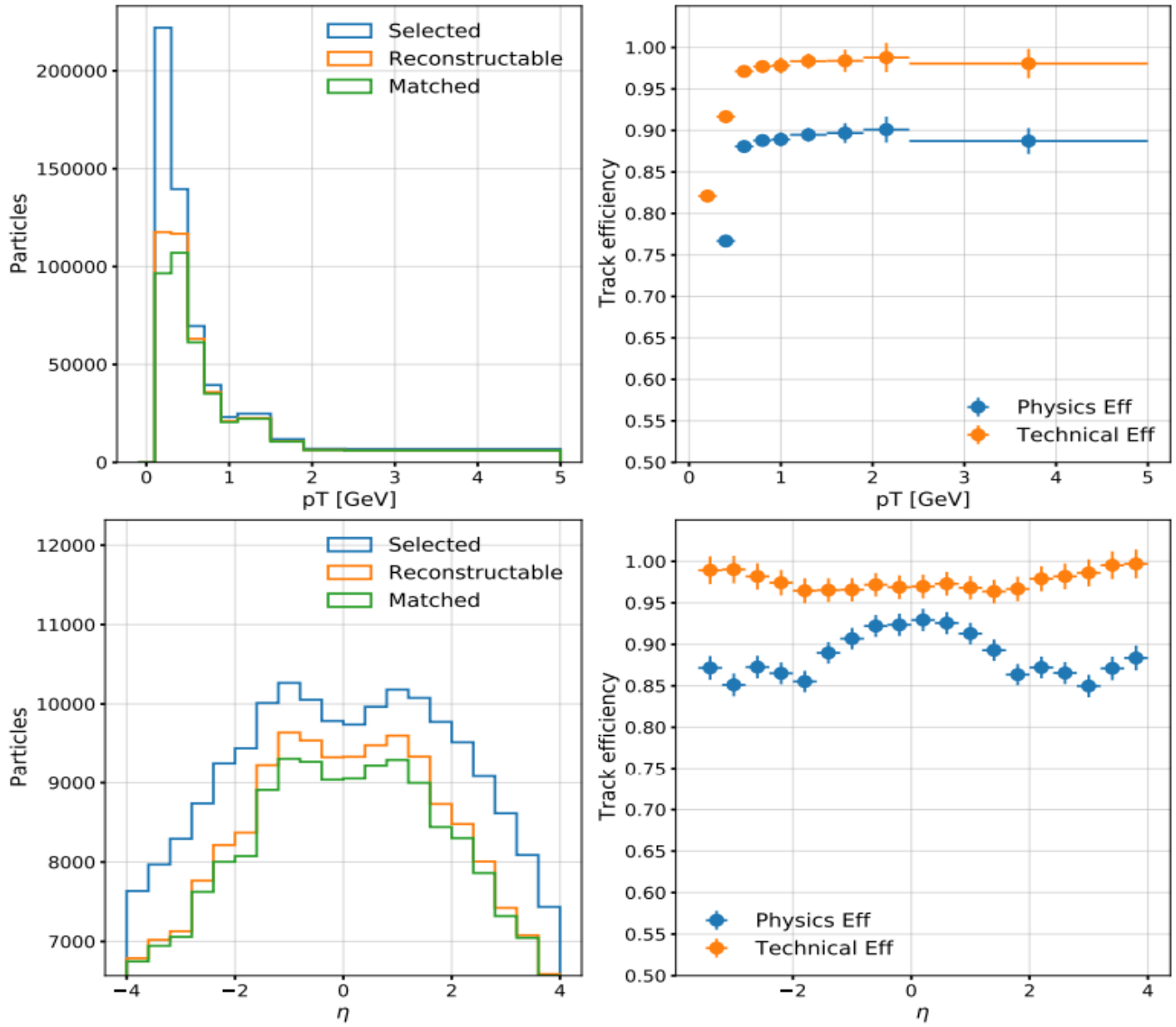
$$\text{Physics Efficiency} = \frac{N_{particles}(\text{selected, matched})}{N_{particles}(\text{selected})}$$

$$\text{Technical Efficiency} = \frac{N_{particles}(\text{selected, reconstructable, matched})}{N_{particles}(\text{selected, reconstructable})}$$

$$\text{Purity} = \frac{N_{tracks}(\text{selected,matched})}{N_{tracks}(\text{selected})}$$

TrkX

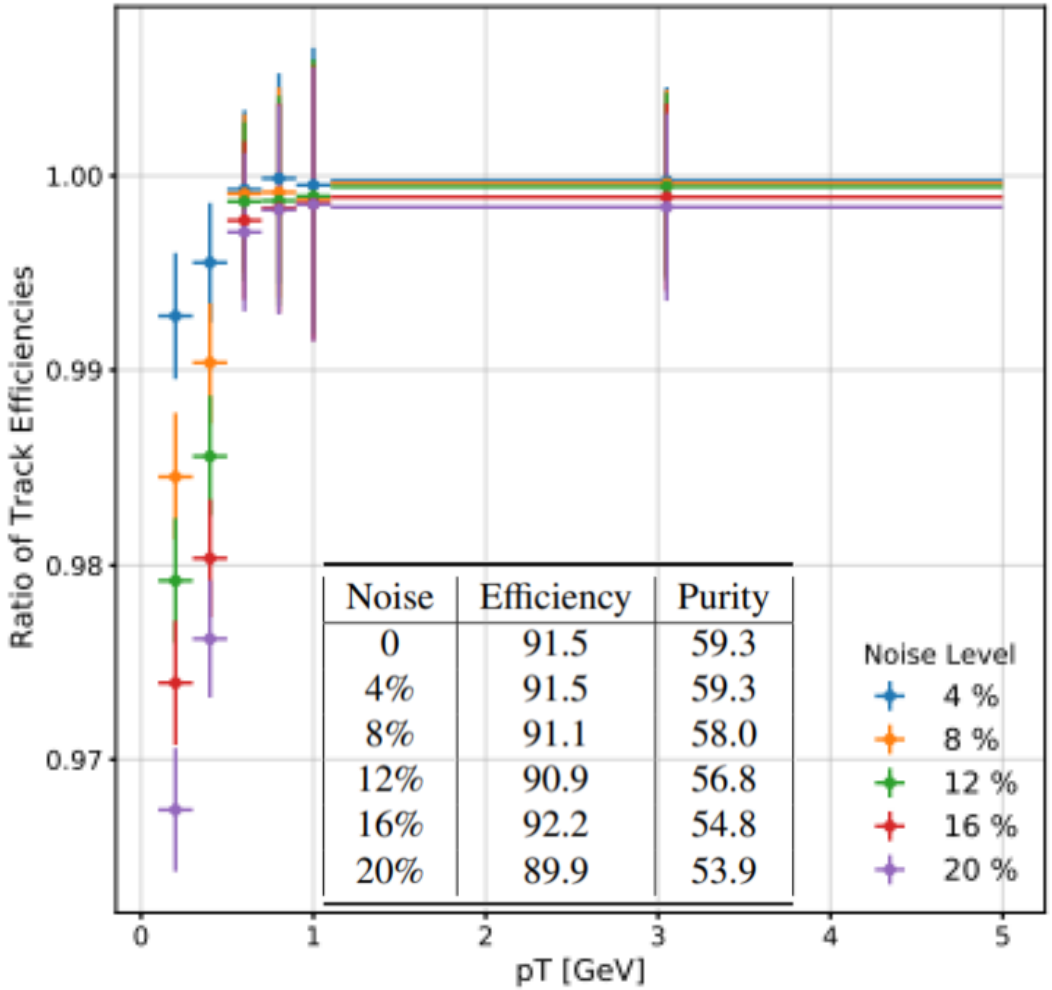
PERFORMANCE – TRACKING EFFICIENCY & PURITY



- $pT > 500\text{MeV}$
  Phys. eff: 88.7 ± 0.3%
  Tech. eff: 97.6 ± 0.3%
- $pT > 100\text{MeV}$
  Phys. eff: 67.2 ± 0.1%
  Tech. eff: 91.3 ± 0.2%
- Purity: 58.3 ± 0.6%

PERFORMANCE – ROBUSTNESS TO NOISE

- Noise re-introduced to dataset *without* retraining (i.e. how “robust” is the pipeline to unseen noise)
- Purity decreases linearly and proportionally with noise – this is to be expected
- Efficiency remains robust to noise (relative efficiency > 96%)
- Efficiency *particularly* robust for pT > 500MeV (relative efficiency > 99.5%)



Noise	Efficiency	Purity
0	91.5	59.3
4%	91.5	59.3
8%	91.1	58.0
12%	90.9	56.8
16%	92.2	54.8
20%	89.9	53.9

Relative efficiency = efficiency ratio = $\frac{eff_{\%noise}}{eff_{noiseless}}$

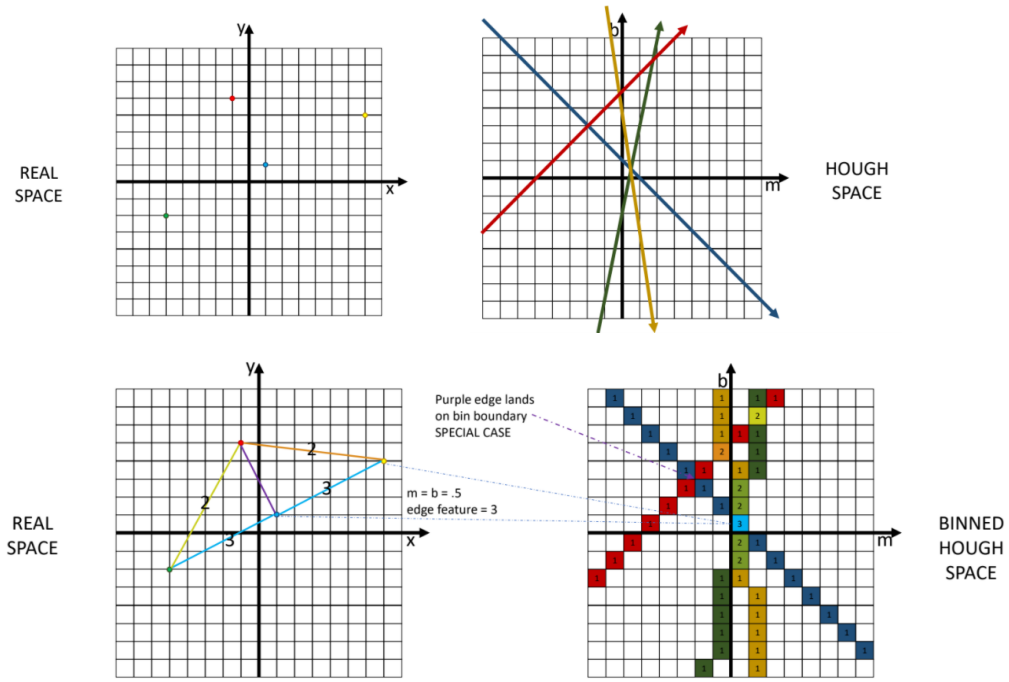
TrkX

EXTENSIONS – PHYSICS-MOTIVATED AUGMENTATION

HOUGH PARAMETERS

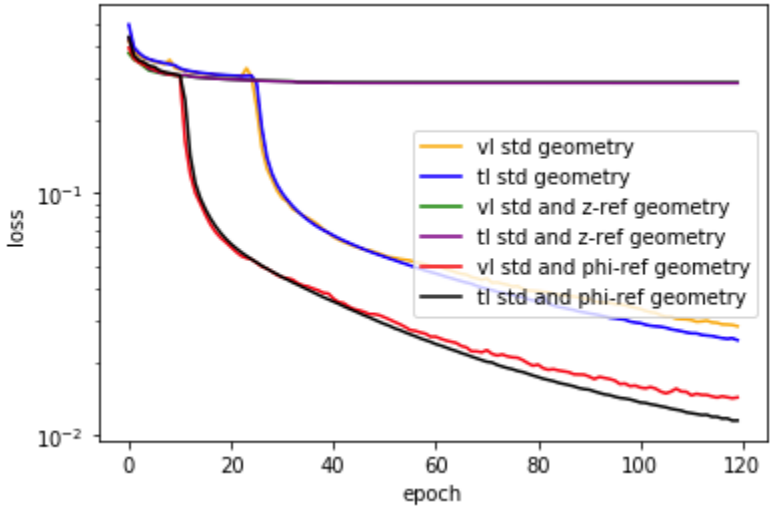
- Preprocess Hough transform features
- Including Hough parameters per edge (fast) improves efficiency 2%, purity 1%
- Including Hough parameters accumulated across edges as “votes” (slow) improves efficiency 5%, purity 3%

Edge Matrix Contents	Track Efficiency	Track Purity
Null	0.43 ± 0.02	0.67 ± 0.02
Zeros	0.66 ± 0.04	0.81 ± 0.03
Hough Parameters	0.68 ± 0.03	0.82 ± 0.04
Hough Accumulator Votes	0.71 ± 0.05	0.84 ± 0.03



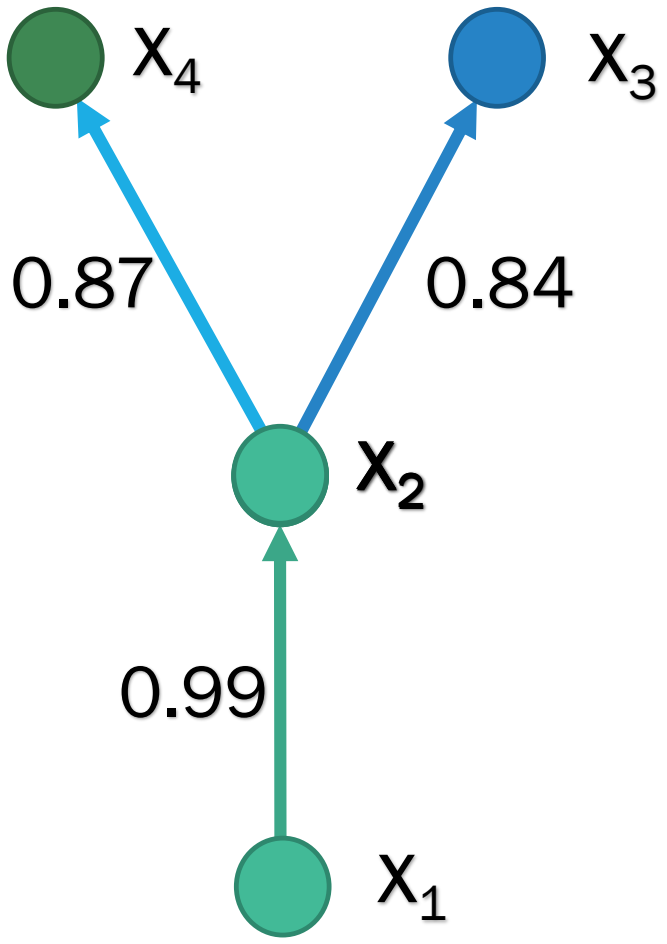
REFLECTIONS & ROTATIONS

- Including reflections around x and y axes reduces loss by factor of 2
- Including reflections around z axis destroys performance (B-field not symmetric)



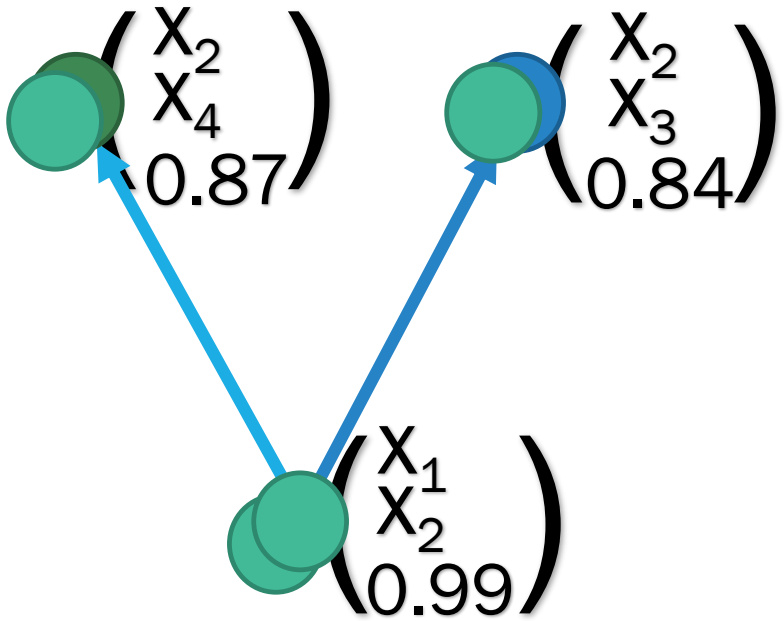
tl = train loss, vl = val. loss

EXTENSIONS – TRIPLET GNN & SEEDING



- 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

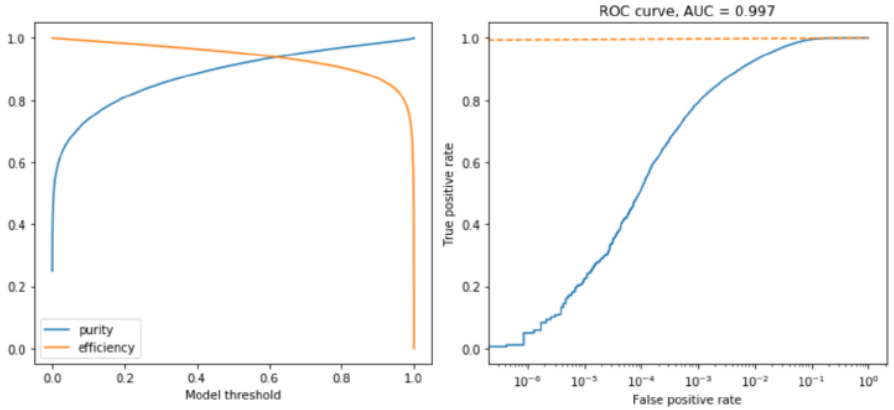
EXTENSIONS – TRIPLET GNN & SEEDING



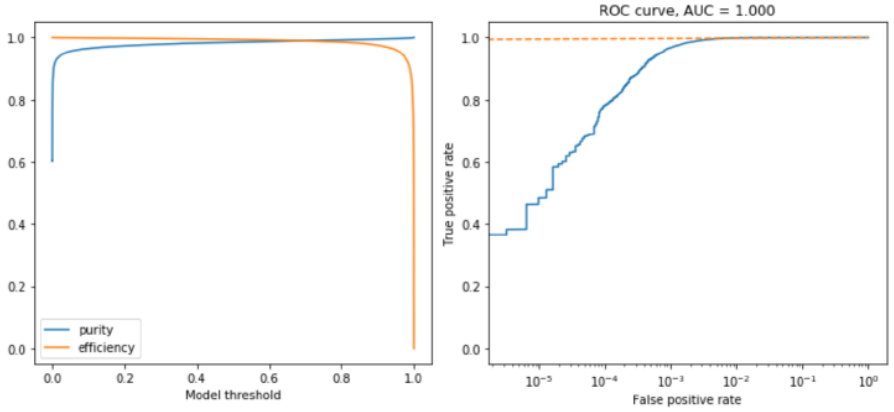
- We convert from a doublet graph to triplet graph: triplet edges have direct access to curvature information, therefore we hypothesise the accuracy should be even better
- Doublets are associated to nodes, triplets are associated to edges

EXTENSIONS – TRIPLET GNN & SEEDING

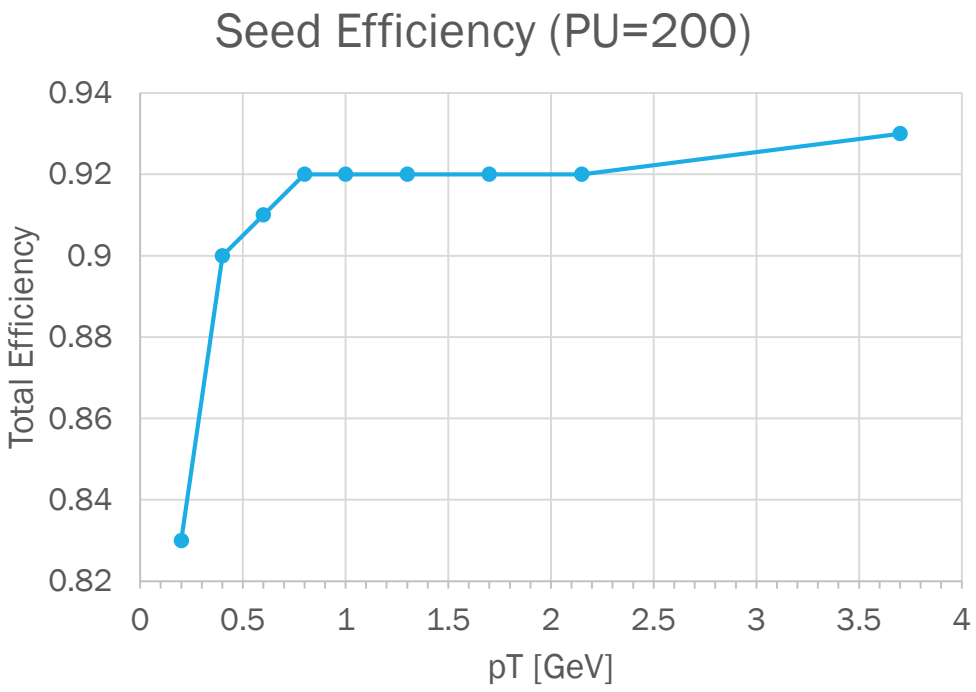
Barrel-only, adjacent-layers



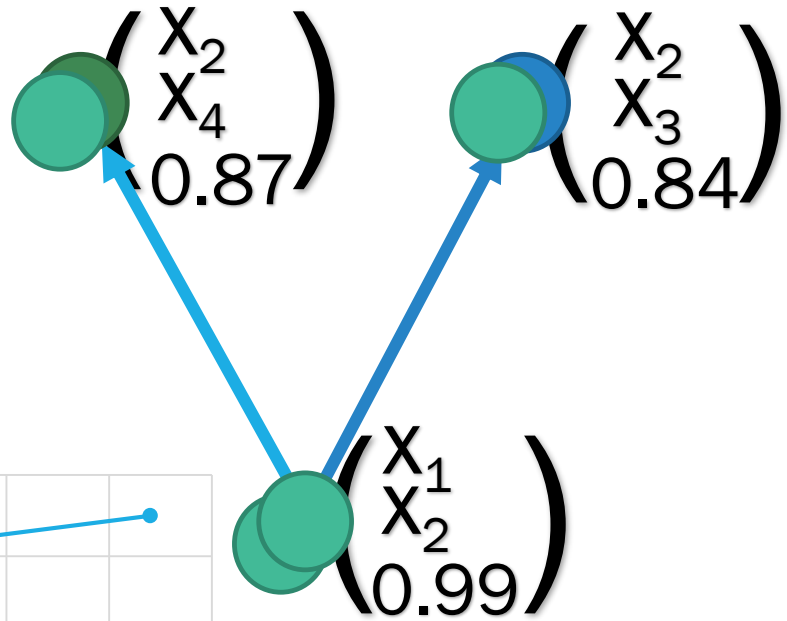
(a) Doublet GNN



(b) Triplet GNN



- Total edge purity: $99.1\% \pm 0.07\%$



COMPUTING PERFORMANCE & EXTENSIONS

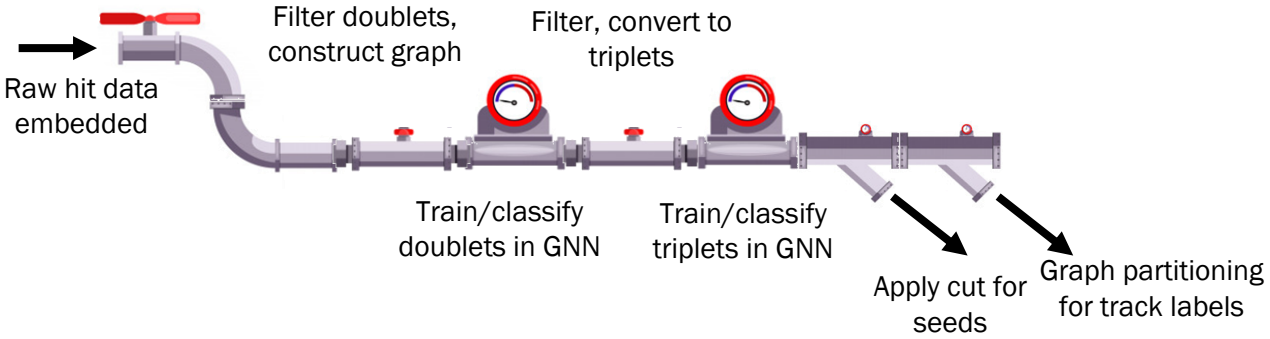
PERFORMANCE – TIMING

- Before GPU optimisations, pipeline took 32.1 seconds
- Approx. **31x speed-up**
- **Thanks to work done in conjunction with Alina Lazar, Alex Ballow (YSU) & Yao Xu (UC Berkeley)**
- Optimisations include:
  1. Custom neighbor search kernel (>2x faster than FAISS)
  2. CuGraph labelling
  3. CuPy preprocessing
  4. Mixed precision inference
  5. Model pre-loading
  6. Batch size hyperparameter tuning

Step	CPU (s)	16Gb GPU (s)	32Gb GPU / 2 GPUs (s)
Preprocessing	0.005	0.002	0.002
Graph construction	168	0.68	0.72
GNN	27	0.24	0.22
(Seed making	*4*	*0.31)*	
Track Labelling	3.32	0.08	0.10
Other (IO, etc.)	3.7	0.11	-0.30
Total	**202**	**1.11**	**0.74** (average)

TrkX

PERFORMANCE – TIMING



Step	CPU (s)	1-GPU (s)	2-GPU (s)
Preprocessing	0.005	0.002	0.002
Graph construction	168	0.68	0.72
GNN	27	0.24	0.22
(Seed making	*4*	*0.31)*	
Track Labelling	3.32	0.08	0.10
Other (IO, etc.)	3.7	0.11	-0.30
Total	**202**	**1.11**	**0.74** (average)

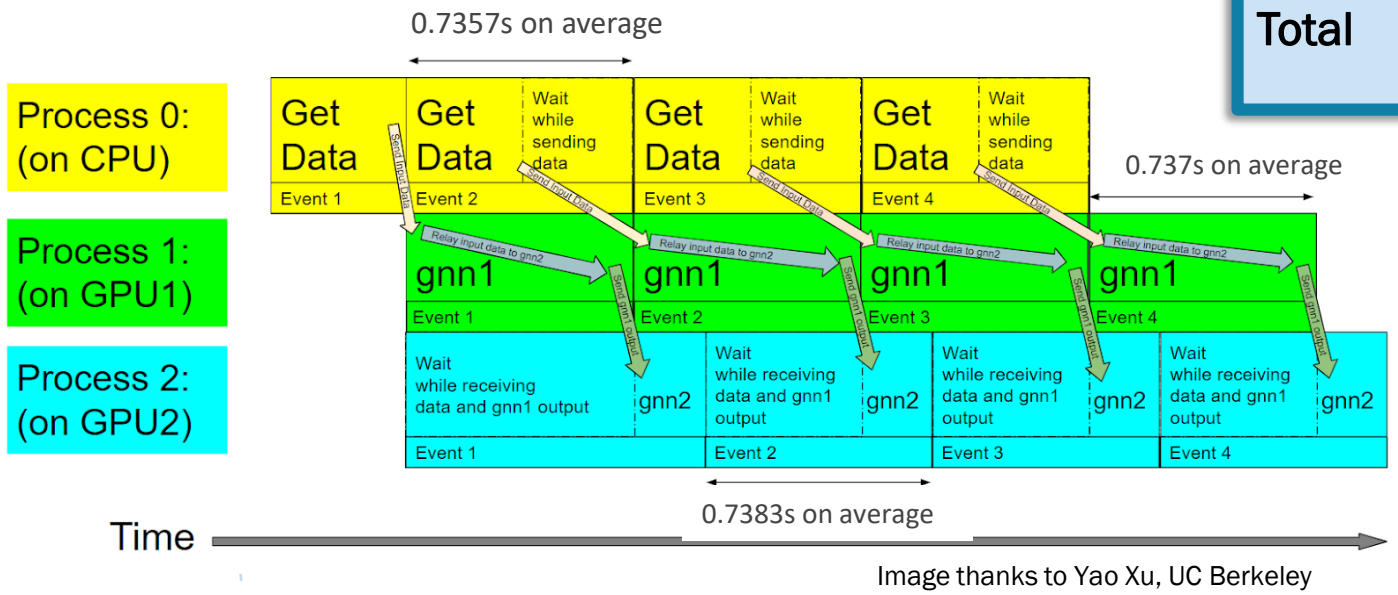


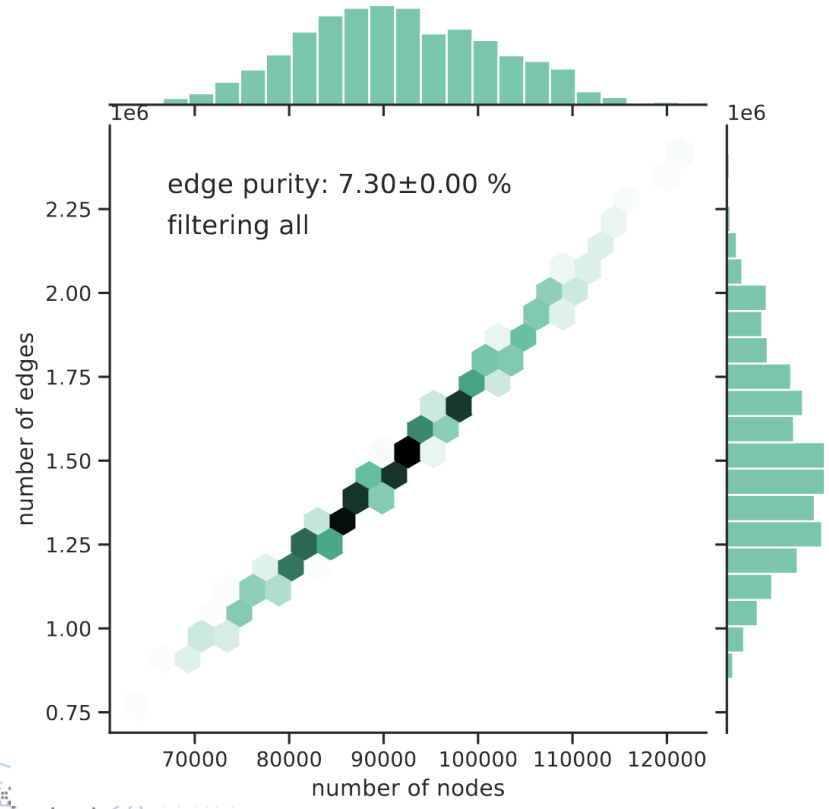
Image thanks to Yao Xu, UC Berkeley

- Before GPU optimisations, pipeline took 32.1 seconds
- Approx. **31x speed-up**

PERFORMANCE – SCALING

MEMORY

- We get sub-quadratic scaling of **graph size** (number of edges) vs number of hits



TIME

- We get sub-quadratic scaling of **total inference time** with number of hits (*gnn1,2* is arbitrary section of pipeline)

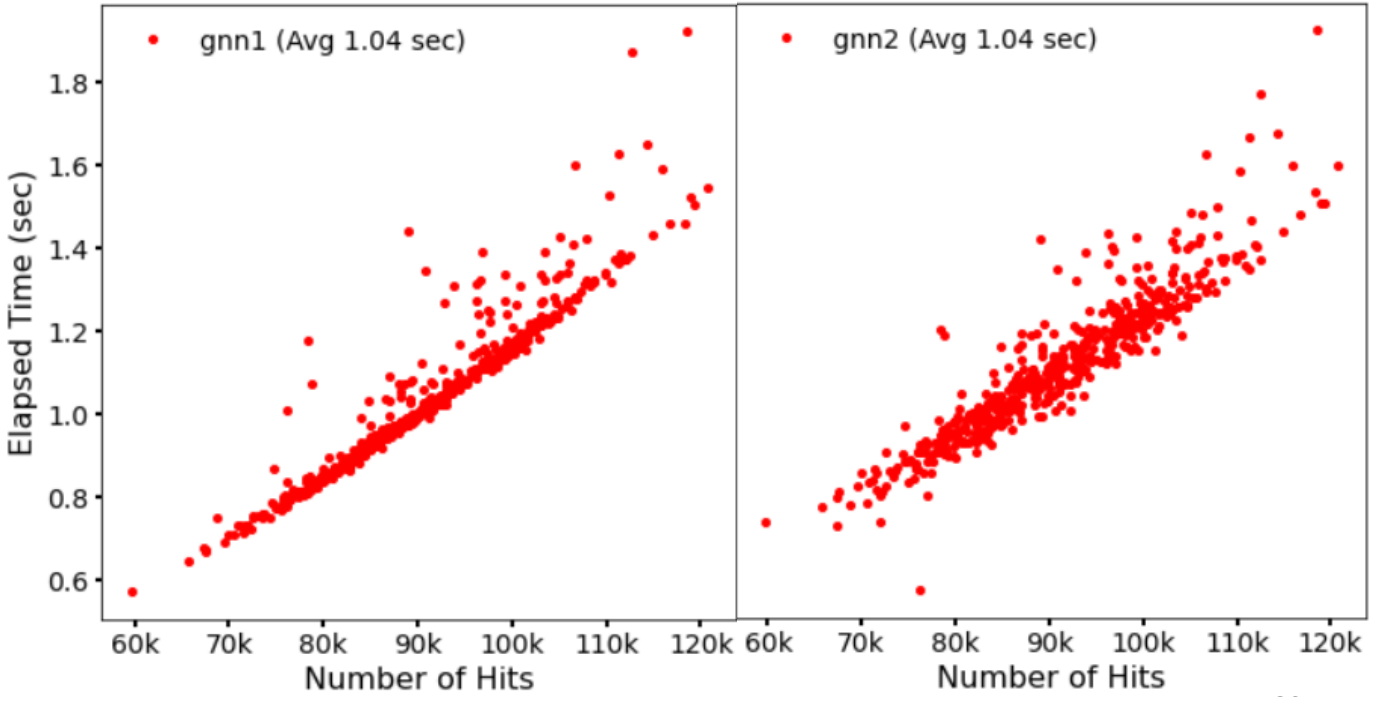
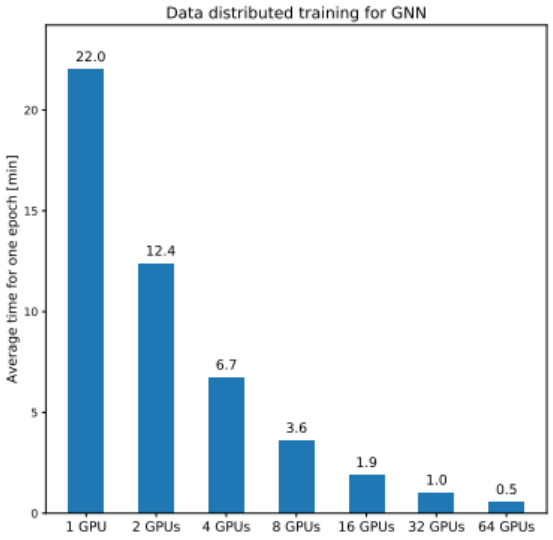


Image thanks to Yao Xu (UC Berkeley) & Alina Lazar (YSU)

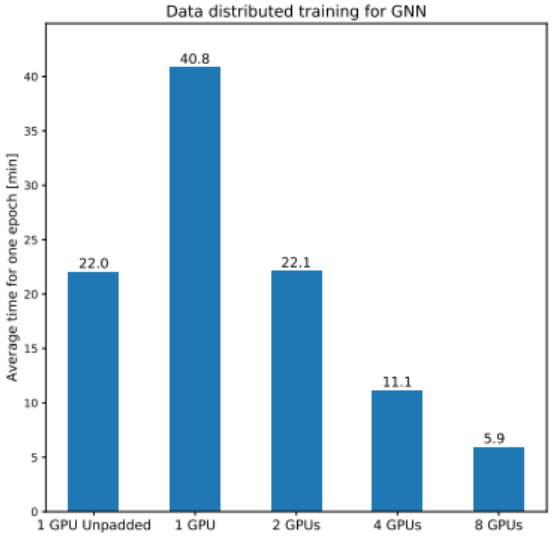
EXTENSIONS – GRAPH-LEVEL DISTRIBUTED TRAINING

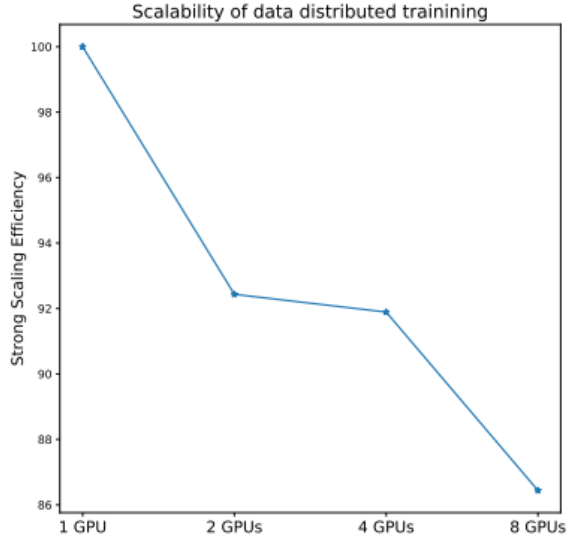
- Event-level parallelization: up to 64 Nvidia V100 across 8 NERSC Cori-GPU nodes
- Horovod has low overhead and supports multi-node MPI
- `tf.distributed` **scales better** than Horovod distributed training
- *However,* TF **requires padding** for uniform input size, giving greater total time

Horovod implementation



`tf.distributed` implementation

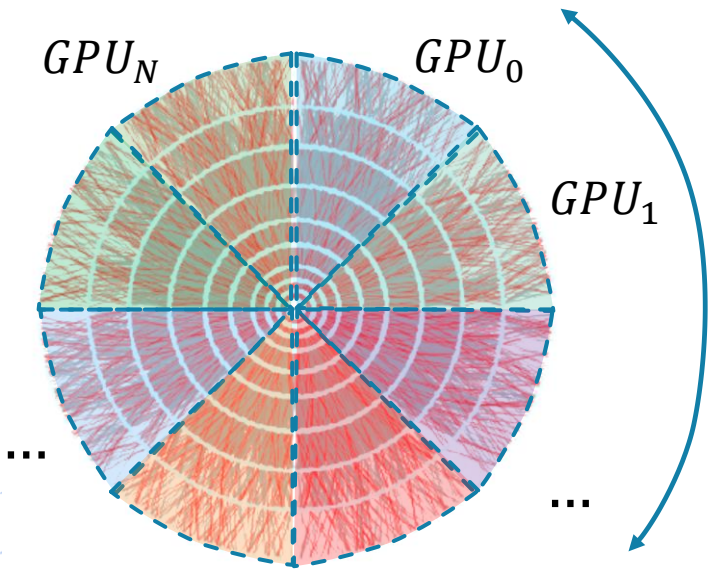




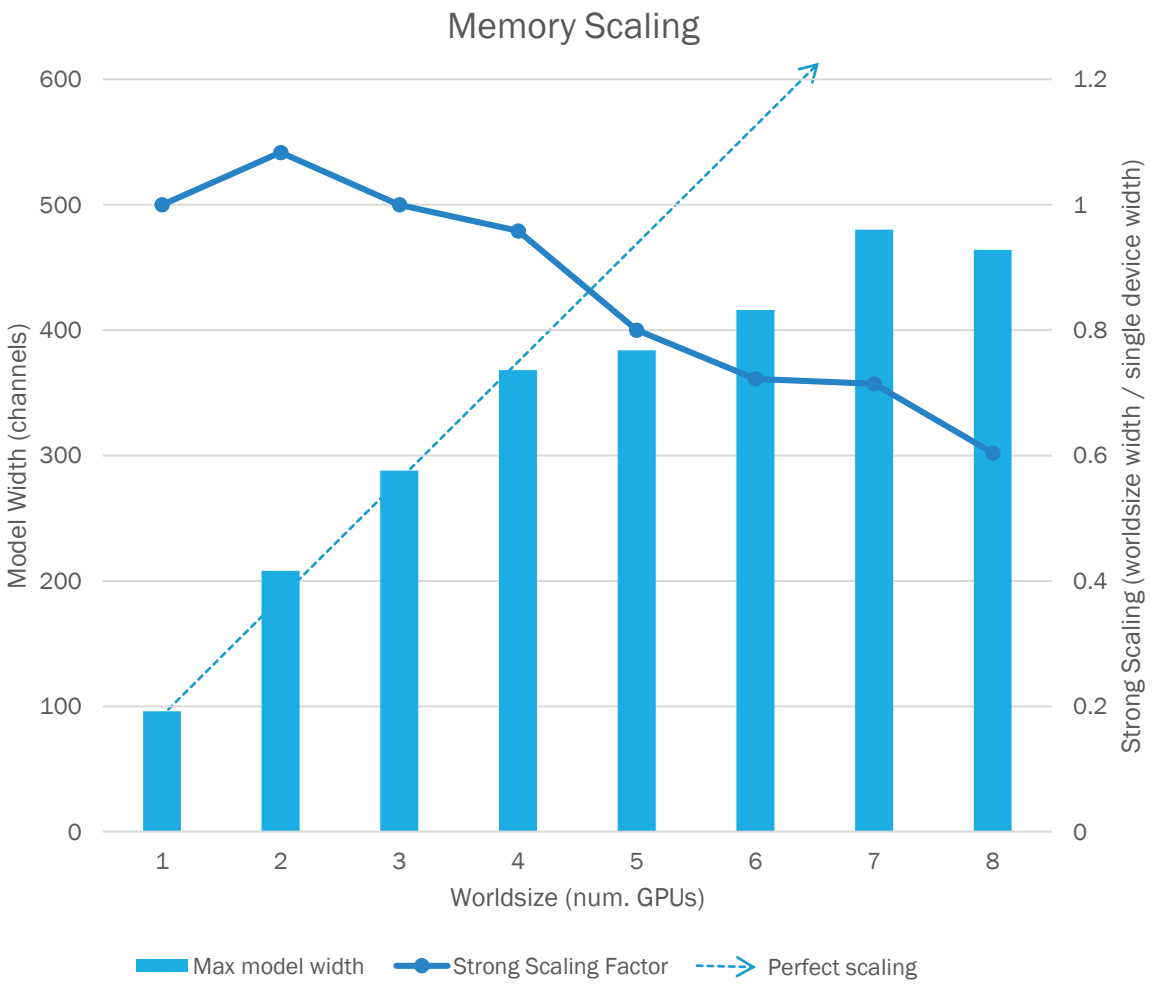
TrkX

EXTENSIONS – NODE/EDGE-LEVEL DISTRIBUTED TRAINING

- We would like to train *very* large GNN models
- True distributed training with Pytorch `DataDistributedParallel`
- Custom `reduce_scatter` hooks written for node-to-node communication across GPUs
- Time more or less constant across worldsize



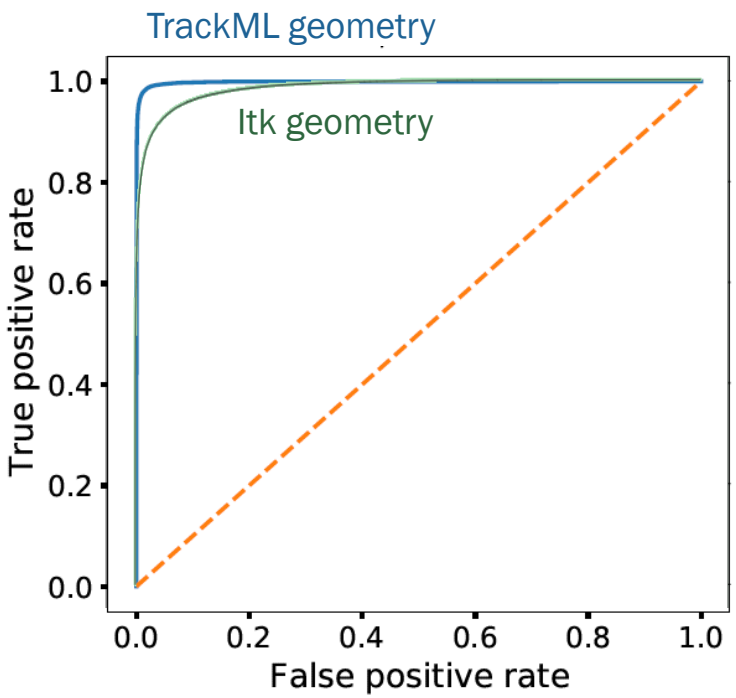
NVCC & InfiniBand `reduce_scatter` gives very fast edge convolutions across GPUs

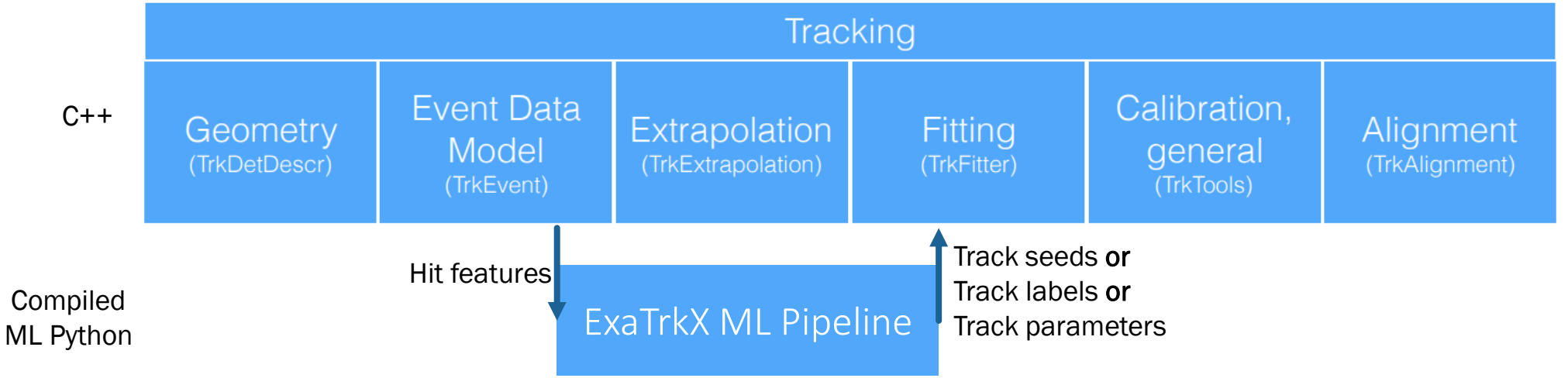


ONGOING & FUTURE WORK

MAIN GOAL: REALISTIC INTEGRATION WITH TRACKING

- **Realistic dataset**
  Currently working to train/infer on ATLAS high pile-up ITk detector dataset – more complicated spacepoint physics
- **Realistic pipeline/implementation**
  Integrating GPU-only Exatrkx track reconstruction into ACTS pipeline – allows metric learning + GNN to be implemented without leaving simulation environment





[A. Salzburger, https://indico.cern.ch/event/699252/contributions/2881457]

OTHER FUTURE WORK

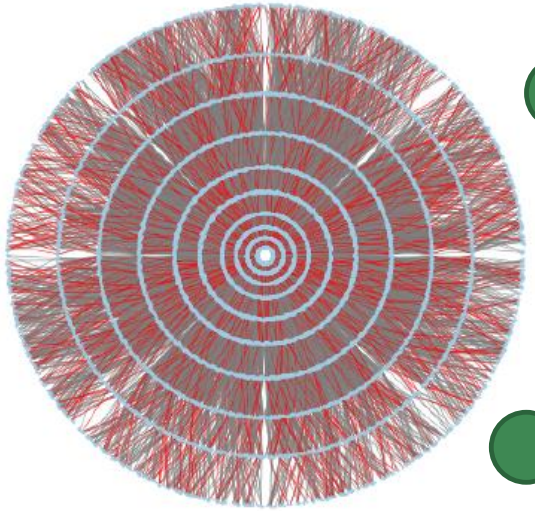
1. Custom graph building kernel (possibly including approx. methods)
2. Model shrinking for inference (quantization, pruning, etc.)
3. Weight training by interesting (e.g. high p_T) tracks
4. Novel architectures:
    1. Pooling/edge contraction,
    2. Gravnet-style end-to-end contrastively trained GNN for labelling
    3. Lorentz equivariant GNNs / metric learning

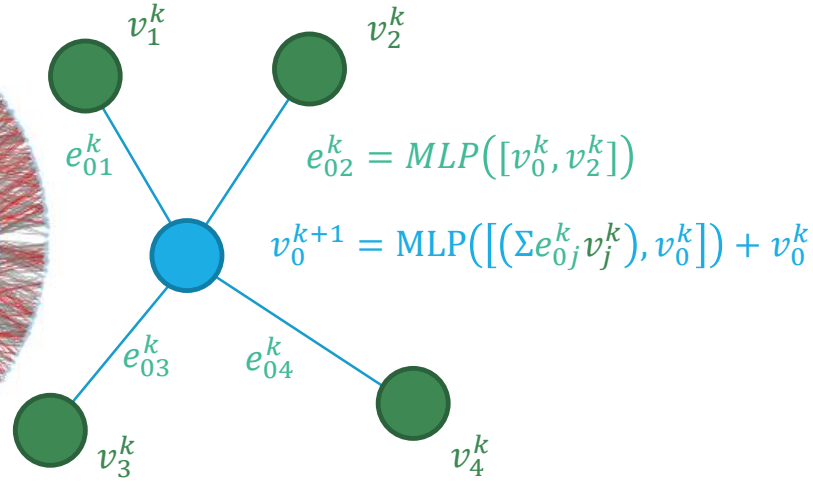
BACKUP

CHALLENGES WITH GNN

- Training takes *significant* memory for a model large enough to give good accuracy.

16Gb

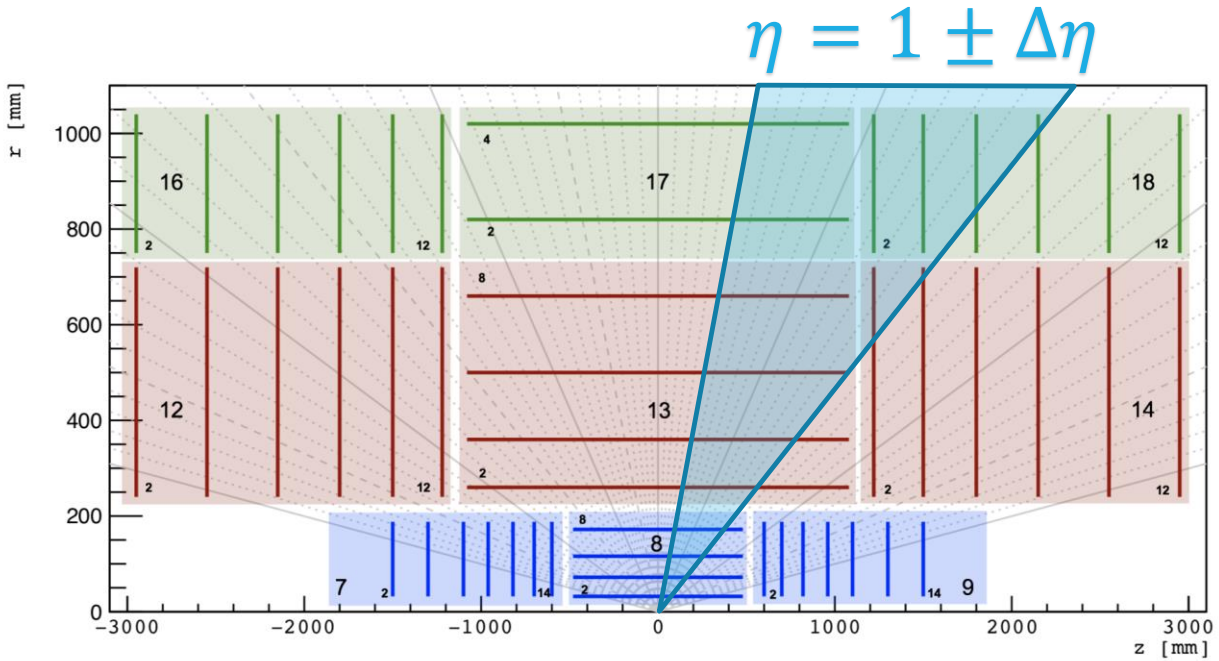




40Gb – …?

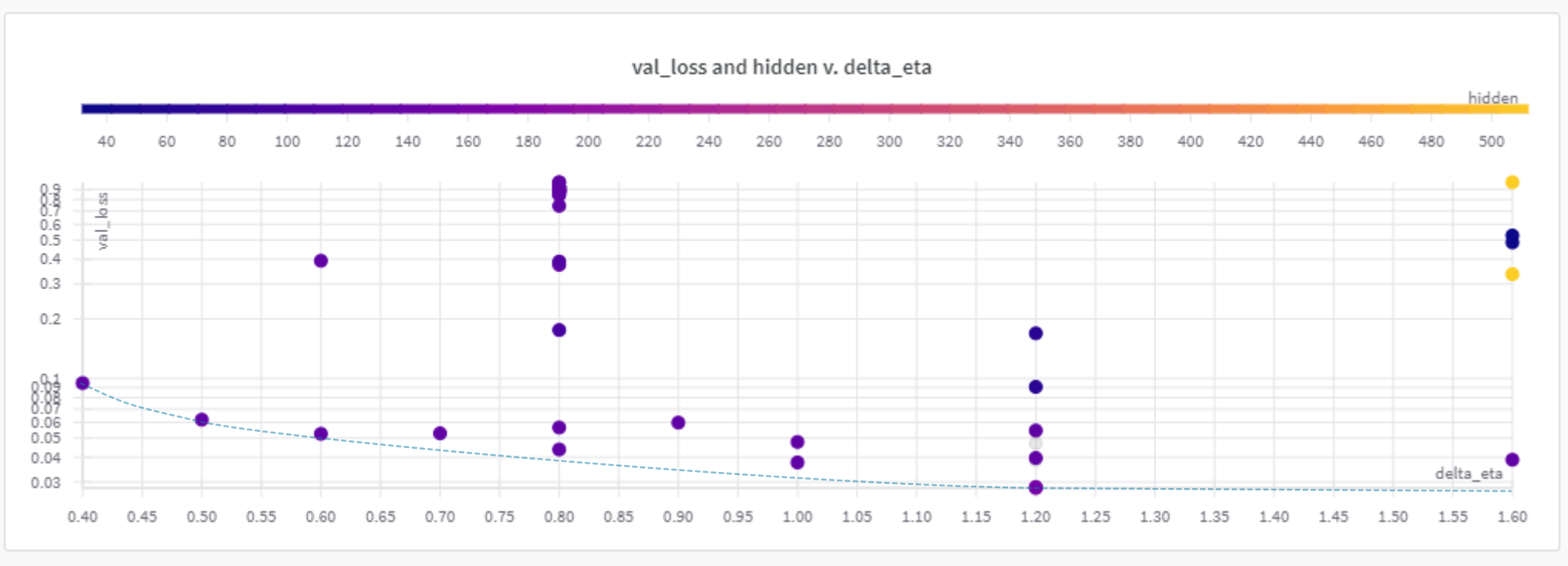
GRAPH INFERENCE PROBLEMS

- **Can** fit the GNN model on the GPU, and **can** fit the filtered edges on the GPU, so what's the problem?
- Memory issues: Cannot naively fit the output of the filter onto a GPU **for training**, therefore any simplification to the training regime (e.g. splitting the graph) will impact inference performance



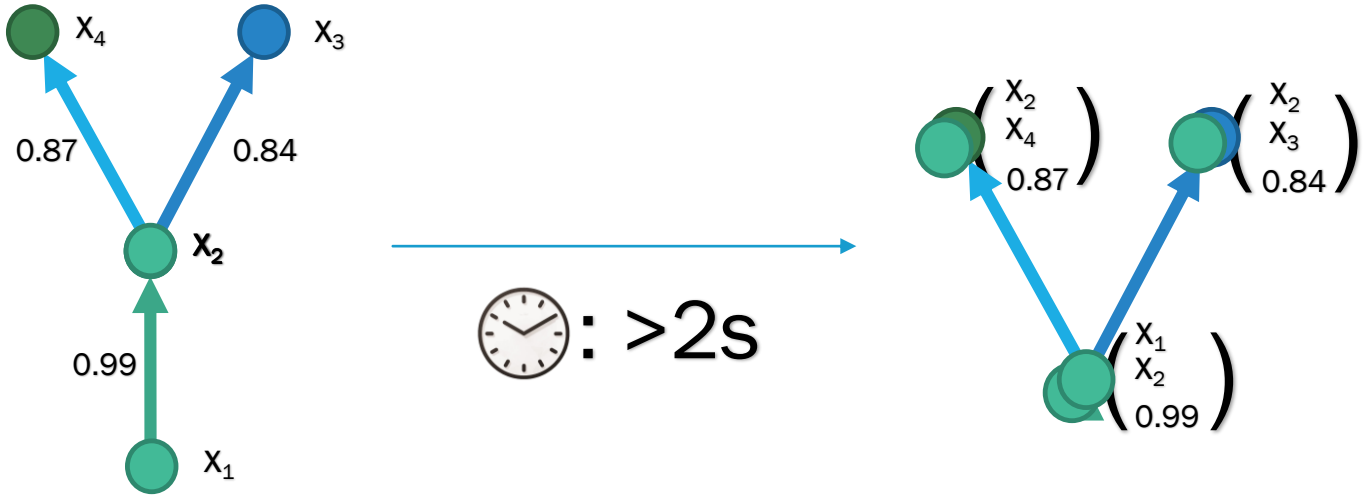
GRAPH INFERENCE PROBLEMS

- Can fit the GNN model on the GPU, and can fit the filtered edges on the GPU, so what's the problem?
- Memory issues: Cannot naively fit the output of the filter onto a GPU **for training**, therefore any simplification to the training regime (e.g. splitting the graph) will impact inference performance



GRAPH INFERENCE PROBLEMS

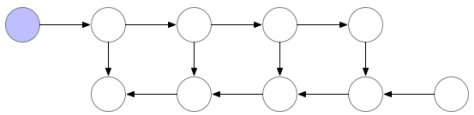
- Can fit the GNN model on the GPU, and can fit the filtered edges on the GPU, so what's the problem?
- Memory issues: Cannot naively fit the output of the filter onto a GPU **for training**, therefore any simplification to the training regime (e.g. splitting the graph) will impact inference performance
- Use both PyTorch and Tensorflow models for development, i.e. heterogenous pipeline switching on-the-fly
- **Triplet inference can boost performance, particularly for seed-making, however…**
- **Doublet-to-triplet conversion with built-in method for O(1.5 million) edges takes over 2 seconds**



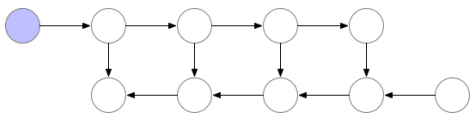
UPGRADES TO GRAPH INFERENCE

- Training memory limitations:
  1. Gradient checkpointing, 22% memory usage
  2. Mixed precision, 50% memory usage
  3. Combined: 11% memory usage, with no impact on accuracy performance
- Weighting by “importance” to improve purity
- Triplet construction feature

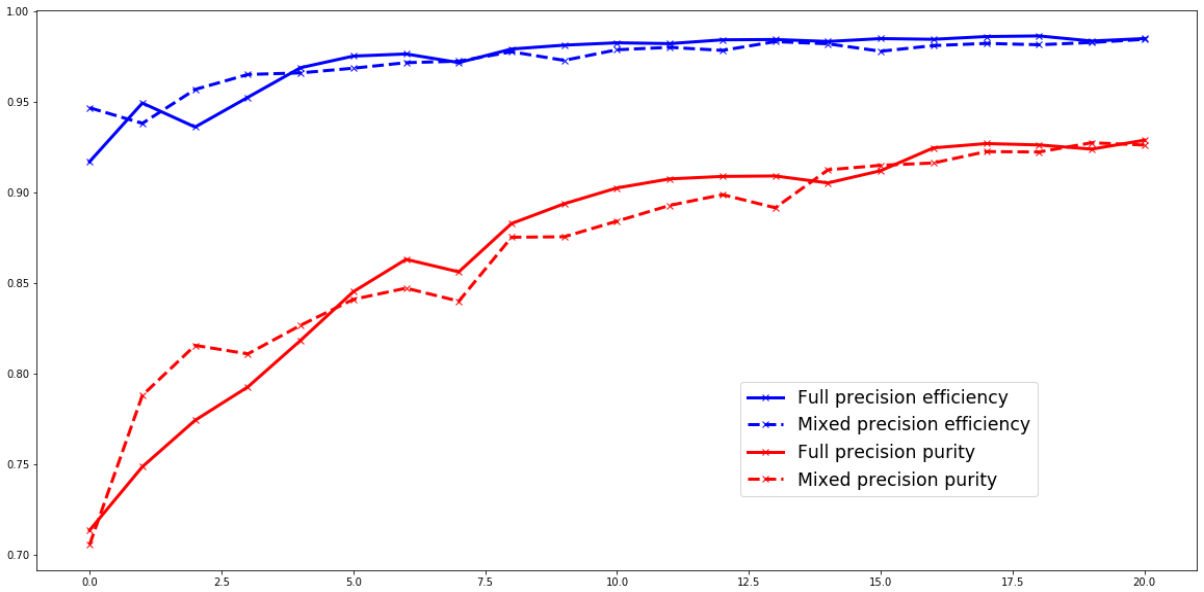
No checkpointing



Maximal checkpointing



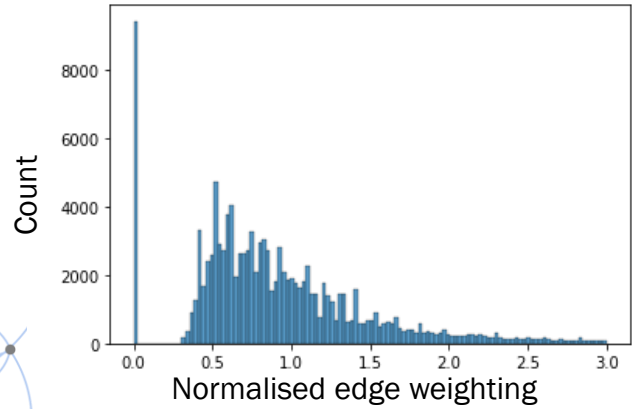
[Chen, et al., arxiv:1604.06174]

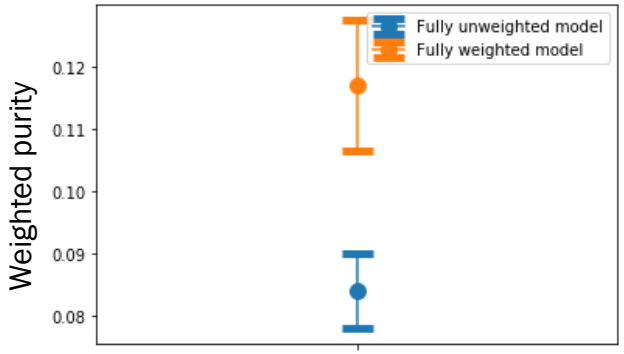


TrkX

UPGRADES TO GRAPH INFERENCE

- Training memory limitations:
  1. Gradient checkpointing, 22% memory usage
  2. Mixed precision, 50% memory usage
  3. Combined: 11% memory usage, with no impact on accuracy performance
- **Weighting by "importance" to improve purity, 75% memory usage**
- Triplet construction feature





Edge weighting

- Weight up first and last hits
- Weight up hits from long and straight tracks
- Secondary ($|z| > 16.5cm, r > 2mm$) particles weight to zero
- Edge weight defined by hit weights:

$$w_{ij} = \begin{cases} \frac{1}{2}(s_i + s_j) * p, & e_{ij} \text{ is True} \\ 1, & e_{ij} \text{ is False} \end{cases}$$

  TrackML score s_i of ith hit, p tuning
- Use (normalised) TrackML weighting, but can apply this criteria to ATLAS
- Allows for smooth performance interpolation across p_T

UPGRADES TO GRAPH INFERENCE

- Training memory limitations:
  1. Gradient checkpointing, 22% memory usage
  2. Mixed precision, 50% memory usage
  3. Combined: 11% memory usage, with no impact on accuracy performance
- Weighting by “importance” to improve purity
- **Triplet construction sparse method, 2s → 10ms**
- **Triplet GNN allows high-performance seed-making**