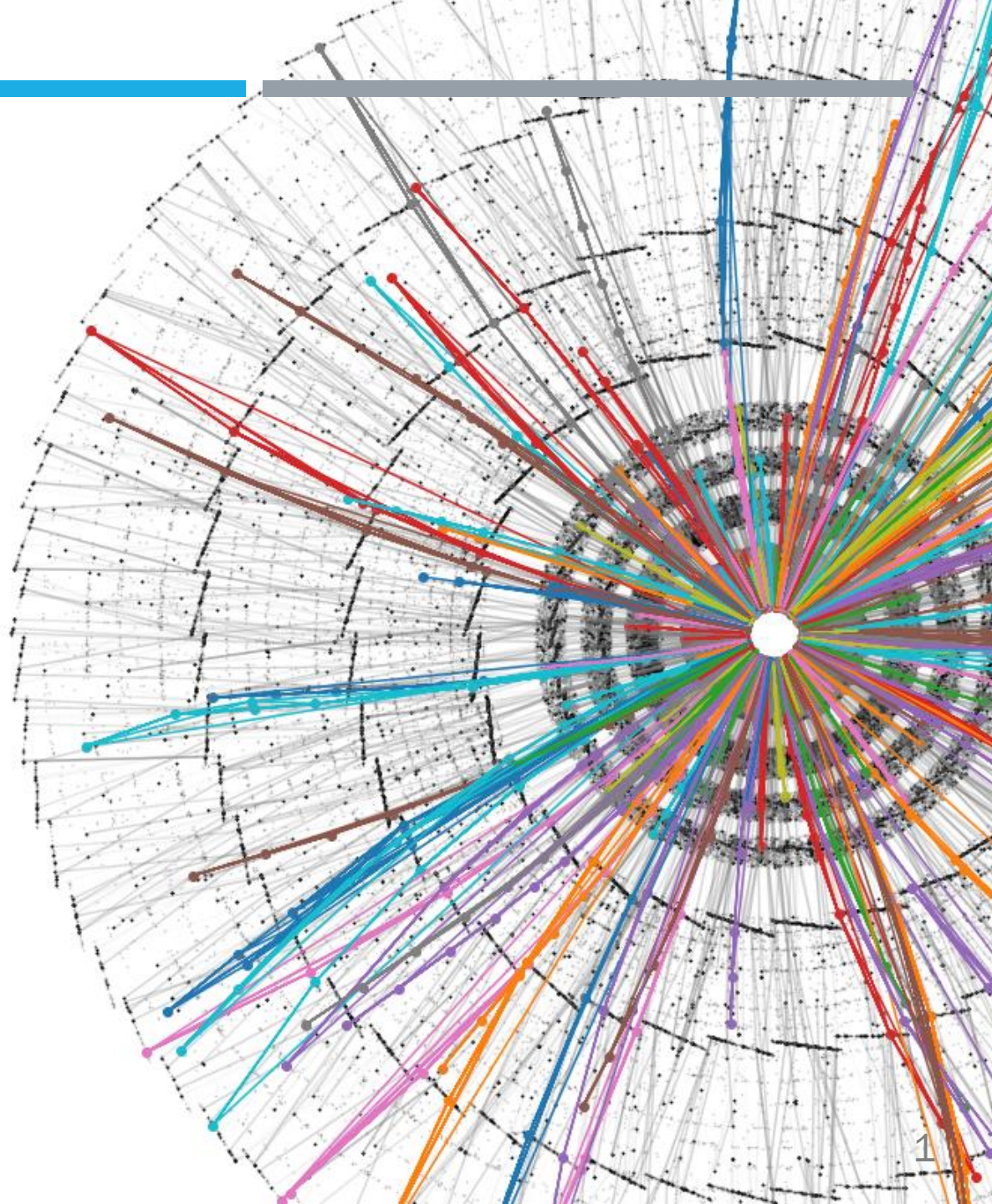
Tracking with Graph Neural Networks

Part 2: Extensions

DANIEL MURNANE
BERKELEY LAB, CERN

HIGHRR LECTURE WEEK, HEIDELBERG UNIVERSITY
SEPTEMBER 13, 2023



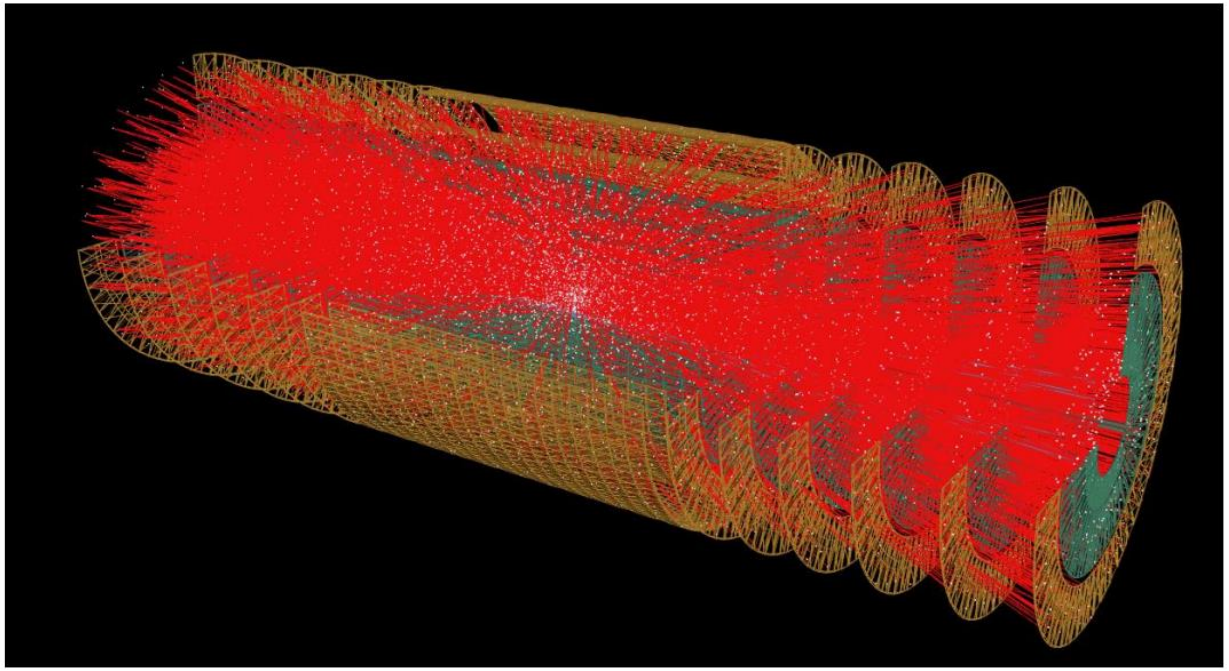
OVERVIEW

- TrackML Competition & Dataset
- TrackML Score & V-Score
- Extending GNN4ITk: Faster, Better, Different
- Faster…
  - Construction Upgrades
  - GNN Upgrades
- Better…
  - Heterogeneity
  - Hierarchy
  - Checkpointing
- Different…
  - Physics-motivated GNNs
  - Object condensation
  - … others?

TRACKML COMPETITION & DATASET

TRACKML CHALLENGE

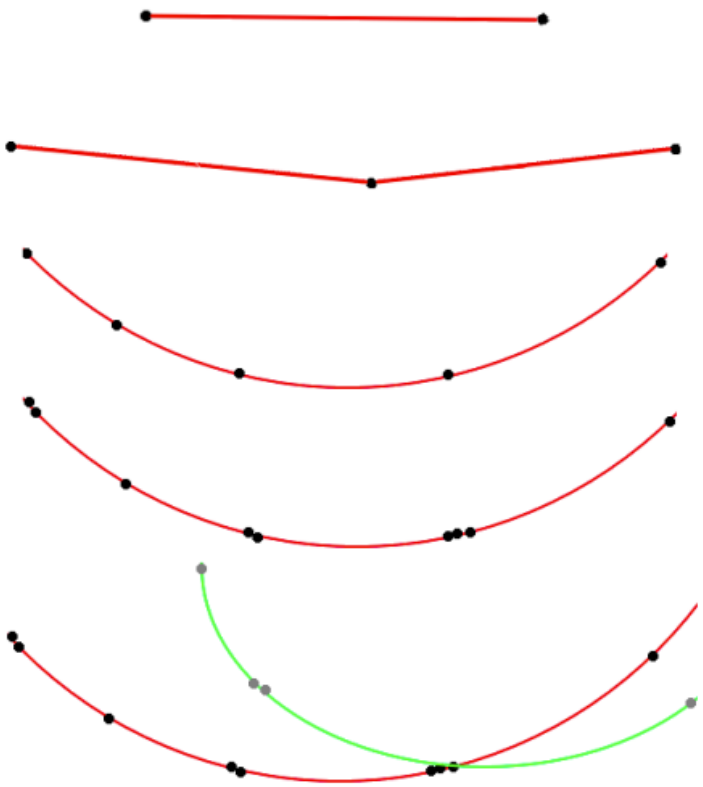
- A Kaggle Competition launched in 2018 for particle tracking with ML
- "Generic detector" was used – ATLAS-like, but removed some of the complications: material effects, secondary particles, much of the noise, shared hits
- Accuracy and throughput phases
- Winners of each:



Arxiv:1904.06778

TRACKML CHALLENGE

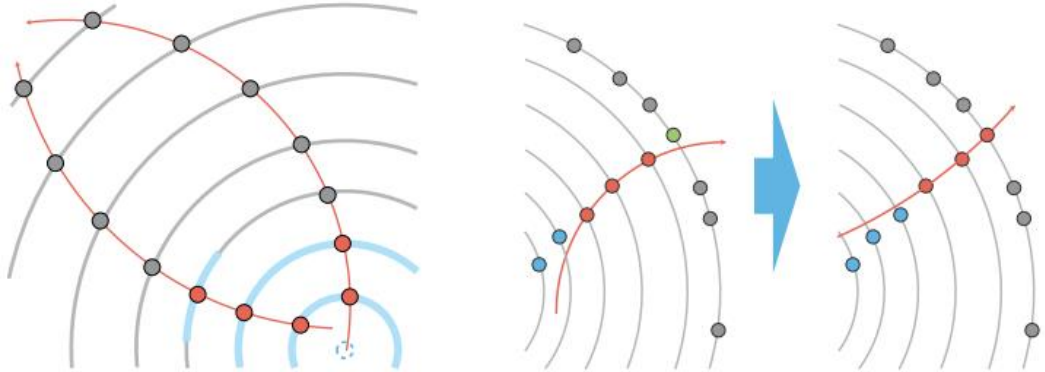
- A Kaggle Competition launched in 2018 for particle tracking with ML
- "Generic detector" was used – ATLAS-like, but removed some of the complications: material effects, secondary particles, much of the noise, shared hits
- Accuracy and throughput phases
- Winners of each:
  - Accuracy: *TopQuarks* – Uses seeds and track following. Conceptually similar to Kalman Filter
  - Throuput:



Arxiv:1904.06778

TRACKML CHALLENGE

- A Kaggle Competition launched in 2018 for particle tracking with ML
- "Generic detector" was used – ATLAS-like, but removed some of the complications: material effects, secondary particles, much of the noise, shared hits
- Accuracy and throughput phases
- Winners of each:
  - Accuracy: *TopQuarks* – Uses seeds and track following. Conceptually similar to Kalman Filter
  - Throuput: *Mikado* – Also uses a similar concept to progress tracking, e.g. Kalman Filter
- What's the takeaway here? It's not straightforward to beat the old ways!



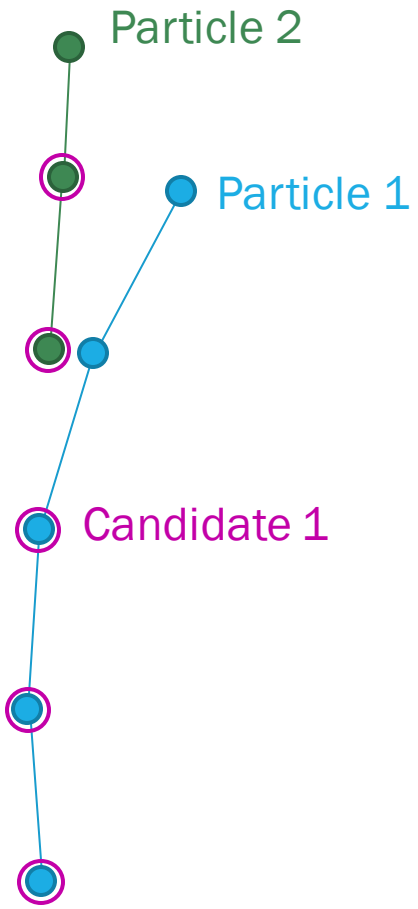
Arxiv:2105.01160

TRACKING METRICS

TRACK MATCHING DEFINITIONS

- $N(P_i, C_j)$ is the number of spacepoints shared by particle i and candidate j
- Particle i is called "matched" if, for some j, $\frac{N(P_i,C_j)}{N(P_i)} > f_{truth}$
- Candidate j is called "matched" if, for some i, $\frac{N(P_i,C_j)}{N(C_j)} > f_{reco}$
- Particle i and candidate j are called "double matched" if, for some i and j, $\frac{N(P_i,C_j)}{N(P_i)} > f_{truth}$ and $\frac{N(P_i,C_j)}{N(C_j)} > f_{reco}$
- $eff = \frac{\sum_i P_i(matching\ condition)}{\sum_i P_i}$, $pur = \frac{\sum_j C_j(matching\ condition)}{\sum_j C_j}$

Standard matching: single-matched particles with $f_{truth} = 0.5$
Strict matching: double-matched particles with $f_{reco} = 1.0$



TRACKML SCORE: WEIGHTED MATCHING

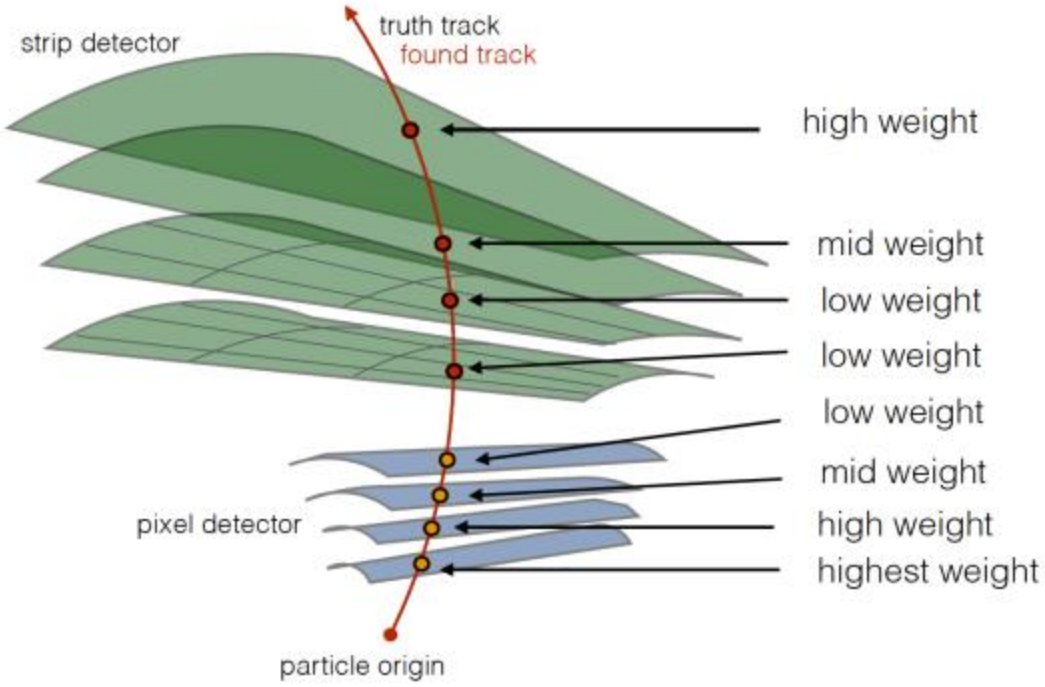
1. Assign an importance to every hit in the event, which all sum to 1
   - Important hits: From long track, innermost and outermost hits, high pt hits
2. A track is correctly "matched" to a particle if:
   - Strictly greater than 50% of hits in the track belong to that particle
   - Strictly greater than 50% of hits in the particle belong to that track

```python
def score_event(truth, submission):
    """Compute the TrackML event score for a single event.

    Parameters
    ----------
    truth : pandas.DataFrame
        Truth information. Must have hit_id, particle_id, and weight columns.
    submission : pandas.DataFrame
        Proposed hit/track association. Must have hit_id and track_id columns.
    """
    tracks = _analyze_tracks(truth, submission)
    purity_rec = numpy.true_divide(tracks['major_nhits'], tracks['nhits'])
    purity_maj = numpy.true_divide(tracks['major_nhits'], tracks['major_particle_nhits'])
    good_track = (0.5 < purity_rec) & (0.5 < purity_maj)
    return tracks['major_weight'][good_track].sum()
```

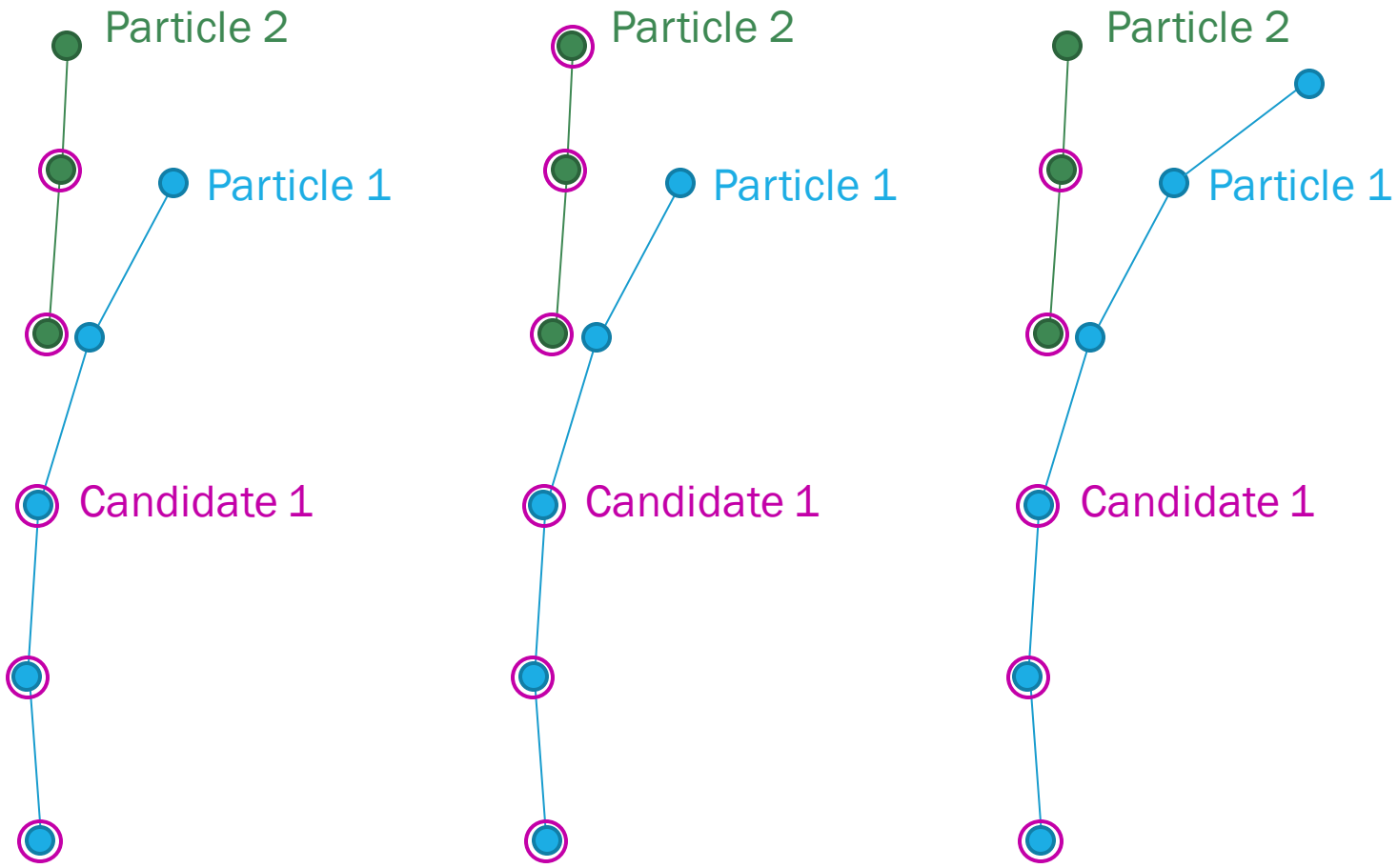
TRACKML SCORE: WEIGHTED MATCHING

1. Assign an importance to every hit in the event, which all sum to 1
   - Important hits: From long track, innermost and outermost hits, high pt hits
2. A track is correctly "matched" to a particle if:
   - Strictly greater than 50% of hits in the track belong to that particle
   - Strictly greater than 50% of hits in the particle belong to that track



TRACKML SCORE: WEIGHTED MATCHING

1. Assign an importance to every hit in the event, which all sum to 1
   - Important hits: From long track, innermost and outermost hits, high pt hits
2. A track is correctly "matched" to a particle if:
   - Strictly greater than 50% of hits in the track belong to that particle
   - Strictly greater than 50% of hits in the particle belong to that track



Candidate 1 matched with Particle 1

No match

No match

TRACKML SCORE: WEIGHTED MATCHING

1. Assign an importance to every hit in the event, which all sum to 1
   - Important hits: From long track, innermost and outermost hits, high pt hits
2. A track is correctly "matched" to a particle if:
   - Strictly greater than 50% of hits in the track belong to that particle
   - Strictly greater than 50% of hits in the particle belong to that track
3. All the weights of the matched hits are summed. A perfect matching of all tracks gives a sum of 1

```python
def score_event(truth, submission):
    """Compute the TrackML event score for a single event.

    Parameters
    ----------
    truth : pandas.DataFrame
        Truth information. Must have hit_id, particle_id, and weight columns.
    submission : pandas.DataFrame
        Proposed hit/track association. Must have hit_id and track_id columns.
    """
    tracks = _analyze_tracks(truth, submission)
    purity_rec = numpy.true_divide(tracks['major_nhits'], tracks['nhits'])
    purity_maj = numpy.true_divide(tracks['major_nhits'], tracks['major_particle_nhits'])
    good_track = (0.5 < purity_rec) & (0.5 < purity_maj)
    return tracks['major_weight'][good_track].sum()
```

THE MATCHING PROBLEM

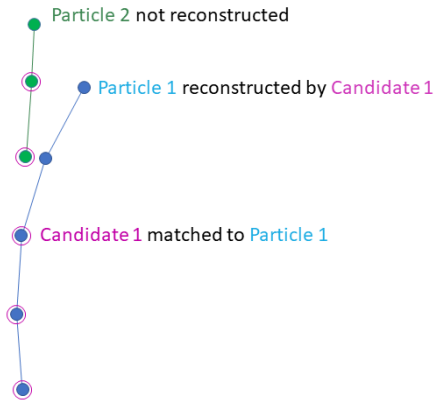
- ATLAS-style? One-way? Two-way?
- What percentage of hits matched?
- Minimum number of hits in track and particle?
- All particles equally important? All hits equally important?
- What about shared hits? Can they be matched?

ATLAS Matching

Description: A particle is ***reconstructed*** if at least one track is matched to it. A track is ***matched*** if over MF% of its hits belong to a single particle.

Performance: $eff = 50\%$
$FR = 0\%$
$DR = 0\%$

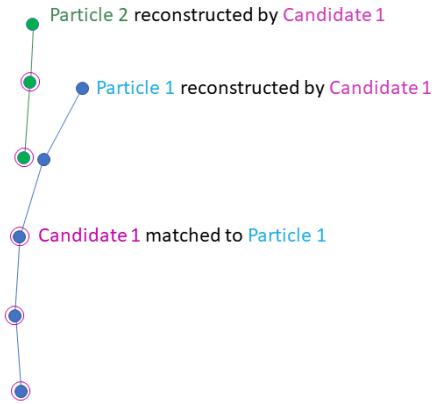
MF%: The given matching fraction
FR: Fake rate = 1 – purity
DR: Duplication rate



One-Way Matching

Description: A particle is ***reconstructed*** if over MF% of its hits belong to a single track. A track is ***matched*** if over MF% of its hits belong to a single particle.

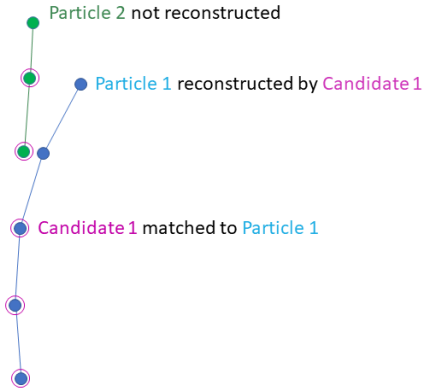
Performance: $eff = 100\%$
$FR = 0\%$
$DR = 0\%$



Two-Way Matching

Description: A particle is ***reconstructed,*** and a track is ***matched***, if over MF% of each of their hits are shared by each other. Therefore, a track is uniquely ***matched*** to the particle it ***reconstructs.***

Performance: $eff = 50\%$
$FR = 0\%$
$DR = 0\%$

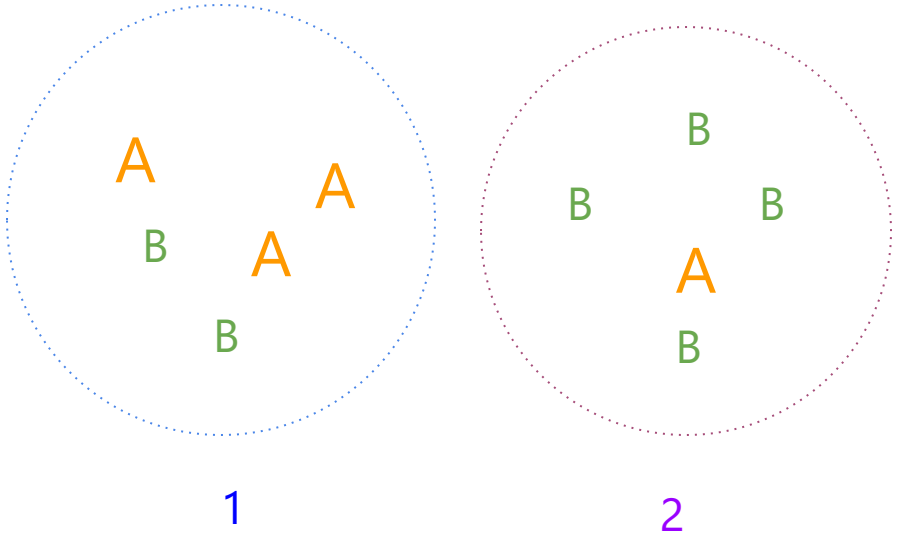


EVALUATING A CLUSTERING

- Assigning a single value to the "goodness of clustering" is non-trivial and non-obvious
- Let's consider an example to see the trade-offs
- Consider a set of objects of type A and B

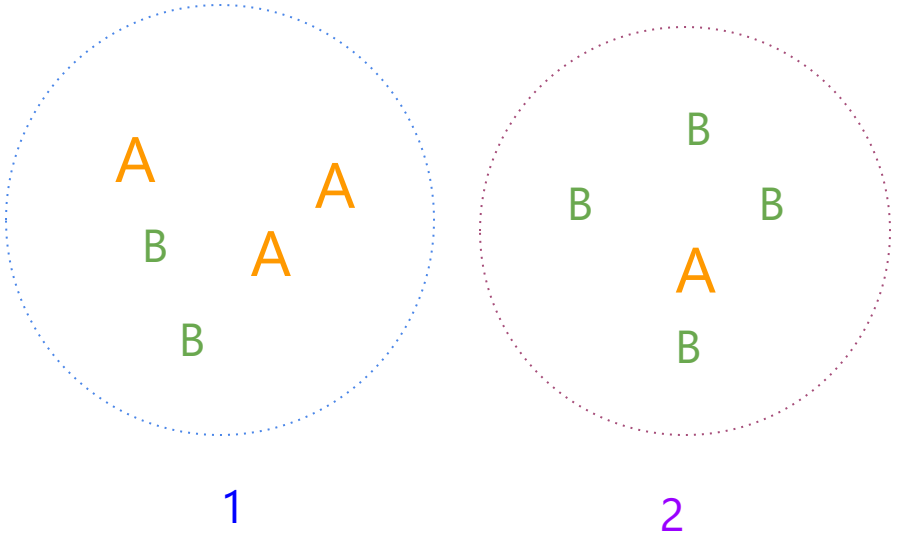
EVALUATING A CLUSTERING

- Assigning a single value to the "goodness of clustering" is non-trivial and non-obvious
- Let's consider an example to see the trade-offs
- Consider a set of objects of type A and B
- Let's cluster them into cluster 1 and cluster 2



EVALUATING A CLUSTERING

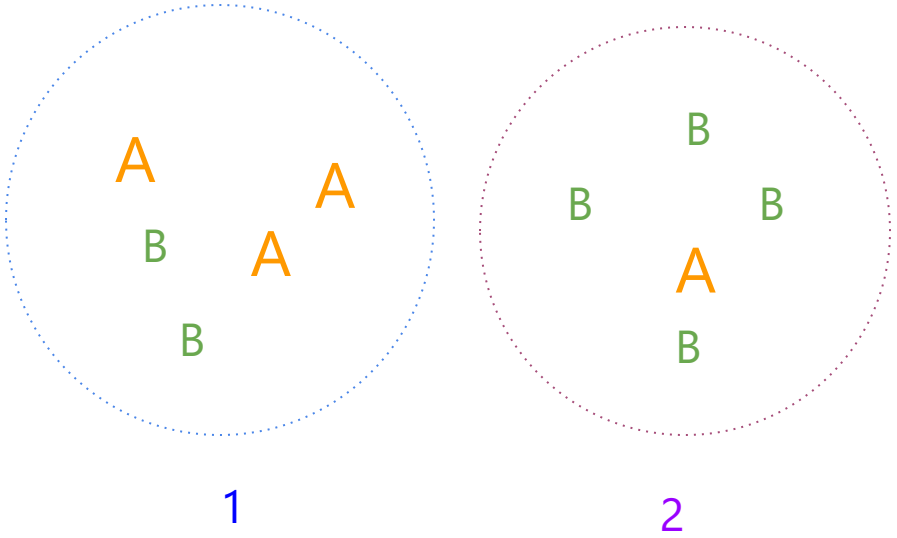
- How should we measure our performance?
- We can start by defining the *entropy* in each cluster



EVALUATING A CLUSTERING

- How should we measure our performance?
- We can start by defining the *entropy* in each cluster
- Entropy: $s = -\sum_i P_i \log(P_i) = -\sum_i \frac{p_i}{N} \log\left(\frac{p_i}{N}\right)$

  where p_i is the number of objects of type i and N is the total number of objects. Then $\frac{p_i}{N}$ is obviously the probability P_i of selecting object i at random.
- This has the nice behavior that the more homogeneous a cluster, as $p_i \to N$, entropy goes to zero

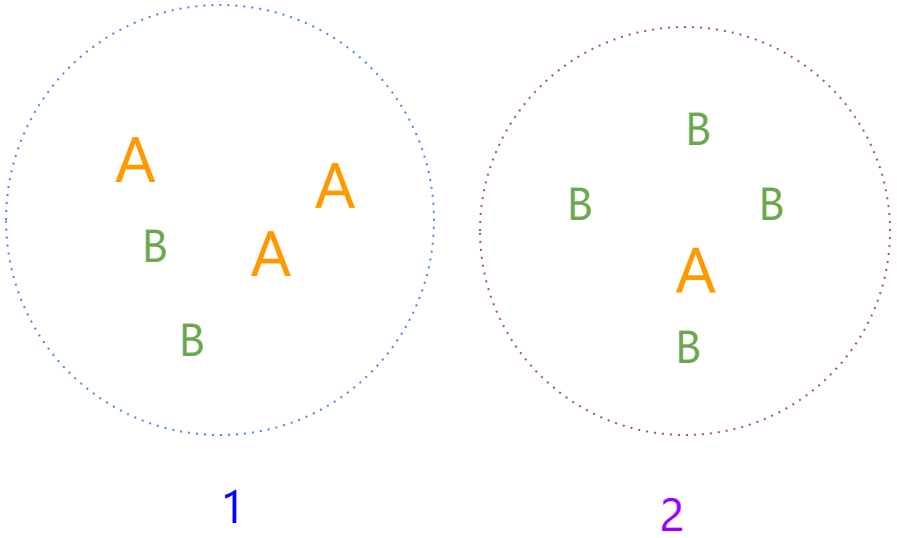


EVALUATING A CLUSTERING

- So, entropy in the two clusters is:

$$s_1 = -(P_A \log(P_A) + P_B \log(P_B))$$
$$= -\left(\frac{3}{5}\log\left(\frac{3}{5}\right) + \frac{2}{5}\log\left(\frac{2}{5}\right)\right)$$
$$= 0.67$$

$$s_2 = -\left(\frac{4}{5}\log\left(\frac{4}{5}\right) + \frac{1}{5}\log\left(\frac{1}{5}\right)\right)$$
$$= 0.5$$



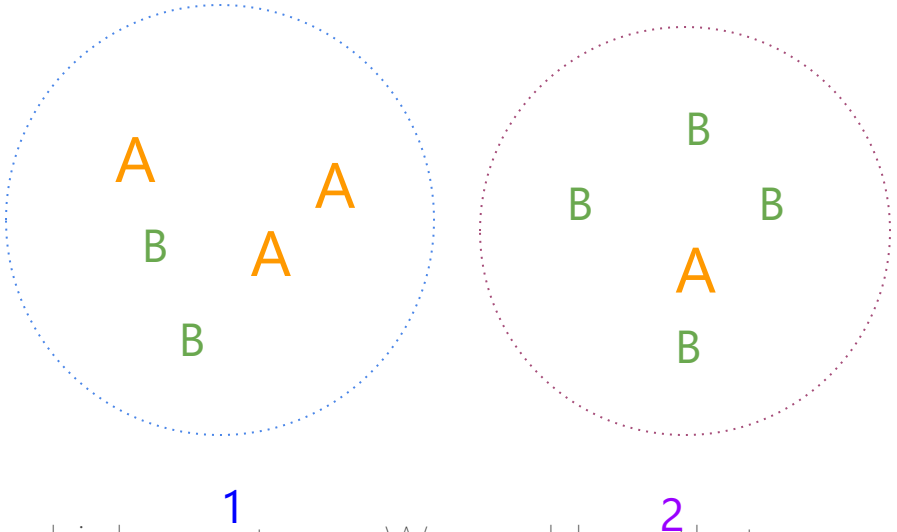
- We can see that the higher ratio of B's to A's in cluster 2 leads to lower entropy – it is more *homogeneous*

EVALUATING A CLUSTERING

- We can also calculate the entropy of each object type, relative to the cluster they have been labelled with

$$s_A = -(P_1 \log(P_1) + P_2 \log(P_2))$$
$$= -\left(\frac{3}{4}\log\left(\frac{3}{4}\right) + \frac{1}{4}\log\left(\frac{1}{4}\right)\right)$$
$$= 0.56$$

$$s_B = -\left(\frac{2}{6}\log\left(\frac{2}{6}\right) + \frac{4}{6}\log\left(\frac{4}{6}\right)\right)$$
$$= 0.73$$



- We can see that the B's are more spread across clusters, so have higher entropy. We could say that, since the A's are better captured by a single cluster, they are more *complete*

HOMOGENEITY, COMPLETENESS AND V-SCORE

V-Measure: A conditional entropy-based external cluster evaluation measure

Andrew Rosenberg and Julia Hirschberg
Department of Computer Science
Columbia University
New York, NY 10027

- We can extend these ideas to capture the homogeneity and completeness across *all* clusters and *all* particles
- The exact derivation is out-of-scope, but you should definitely look into mutual information to understand this properly!
- At the end of the day:
  - Homogeneity is a measure of how well you've kept each cluster to a single particle type
  - Completeness is a measure of how well you've assigned all hits in a particle to a single cluster
- These are the clustering analogy of purity and efficiency
- However, Kaggle allows a single score to capture performance…

HOMOGENEITY, COMPLETENESS AND V-SCORE

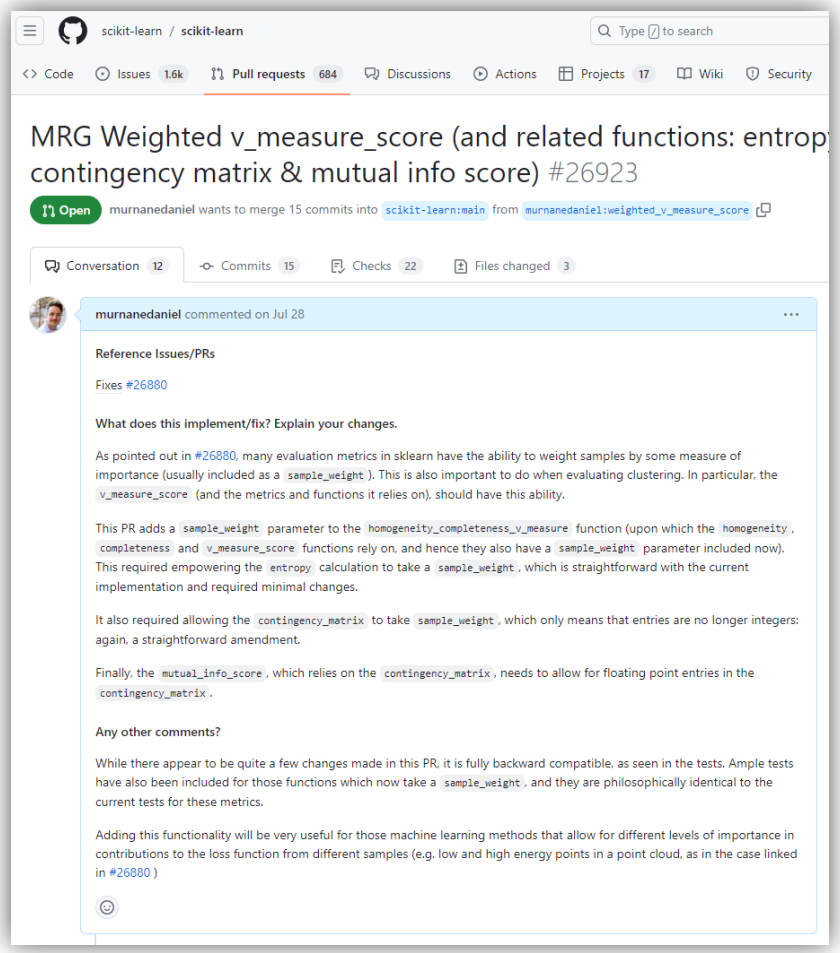
- Enter the V-Score (the "validity score"), and analogy to the F-Score (from efficiency and purity)

$$V = 2\,\frac{H \cdot C}{H + C}$$

- This is the harmonic mean of homogeneity and completeness, and it is zero when either of those measures is zero
- In general, there is a trade-off between H and C, and the V-score allows to smooth over that trade-off

EXTENSION: WEIGHTING THE V-SCORE

- One final point: It is not equally important to cluster all points in a particle
- If a particle leaves several high-energy hits, they should certainly be clustered together
- If two particles have high energy hits, they should certainly *not* be clustered together
- These leads us to create a new V-Score definition: the Weighted V-Score
- The derivation is out-of-scope (the source code will be available at an upcoming version of scikit-learn)
- Can deep dive if you're interested



scikit-learn / scikit-learn

Type / to search

Code | Issues 1.6k | Pull requests 684 | Discussions | Actions | Projects 17 | Wiki | Security

MRG Weighted v_measure_score (and related functions: entropy contingency matrix & mutual info score) #26923

Open murnanedaniel wants to merge 15 commits into scikit-learn:main from murnanedaniel:weighted_v_measure_score

Conversation 12 | Commits 15 | Checks 22 | Files changed 3

murnanedaniel commented on Jul 28

Reference Issues/PRs

Fixes #26880

What does this implement/fix? Explain your changes.

As pointed out in #26880, many evaluation metrics in sklearn have the ability to weight samples by some measure of importance (usually included as a `sample_weight`). This is also important to do when evaluating clustering. In particular, the `v_measure_score` (and the metrics and functions it relies on), should have this ability.

This PR adds a `sample_weight` parameter to the `homogeneity_completeness_v_measure` function (upon which the `homogeneity`, `completeness` and `v_measure_score` functions rely on, and hence they also have a `sample_weight` parameter included now). This required empowering the `entropy` calculation to take a `sample_weight`, which is straightforward with the current implementation and required minimal changes.

It also required allowing the `contingency_matrix` to take `sample_weight`, which only means that entries are no longer integers: again, a straightforward amendment.

Finally, the `mutual_info_score`, which relies on the `contingency_matrix`, needs to allow for floating point entries in the `contingency_matrix`.

Any other comments?

While there appear to be quite a few changes made in this PR, it is fully backward compatible, as seen in the tests. Ample tests have also been included for those functions which now take a `sample_weight`, and they are philosophically identical to the current tests for these metrics.

Adding this functionality will be very useful for those machine learning methods that allow for different levels of importance in contributions to the loss function from different samples (e.g. low and high energy points in a point cloud, as in the case linked in #26880)

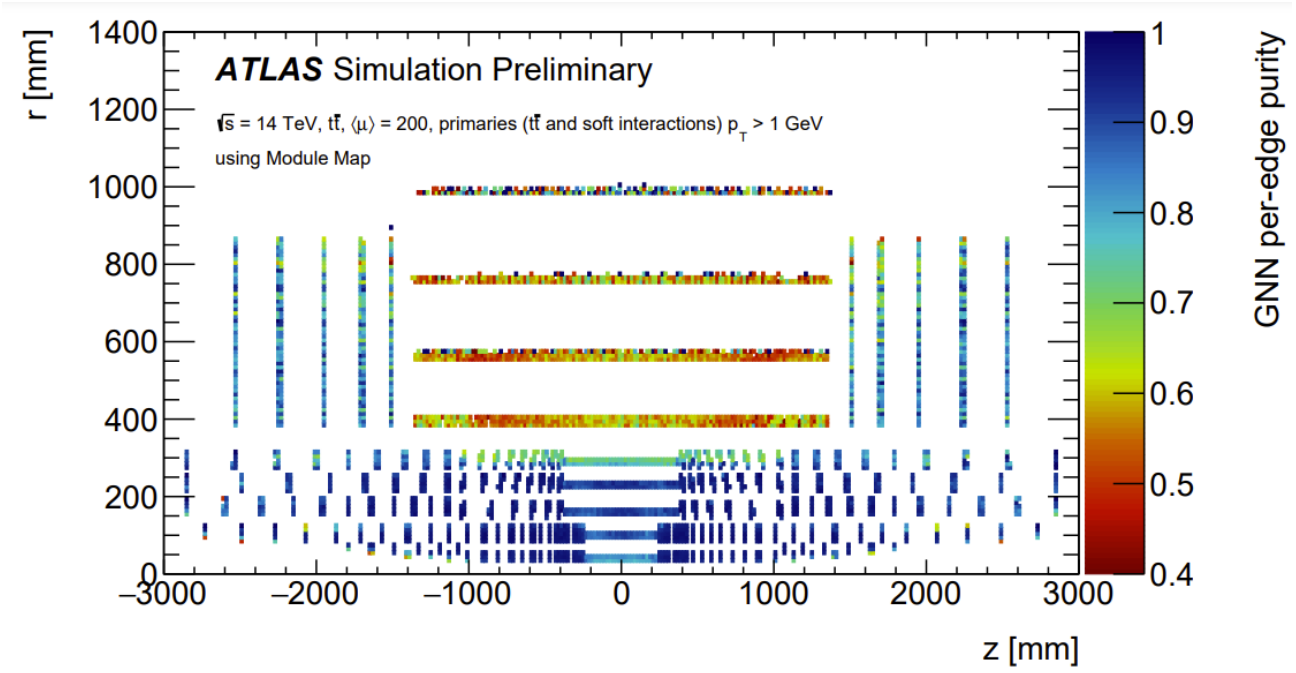
THROUGHPUT & LATENCY

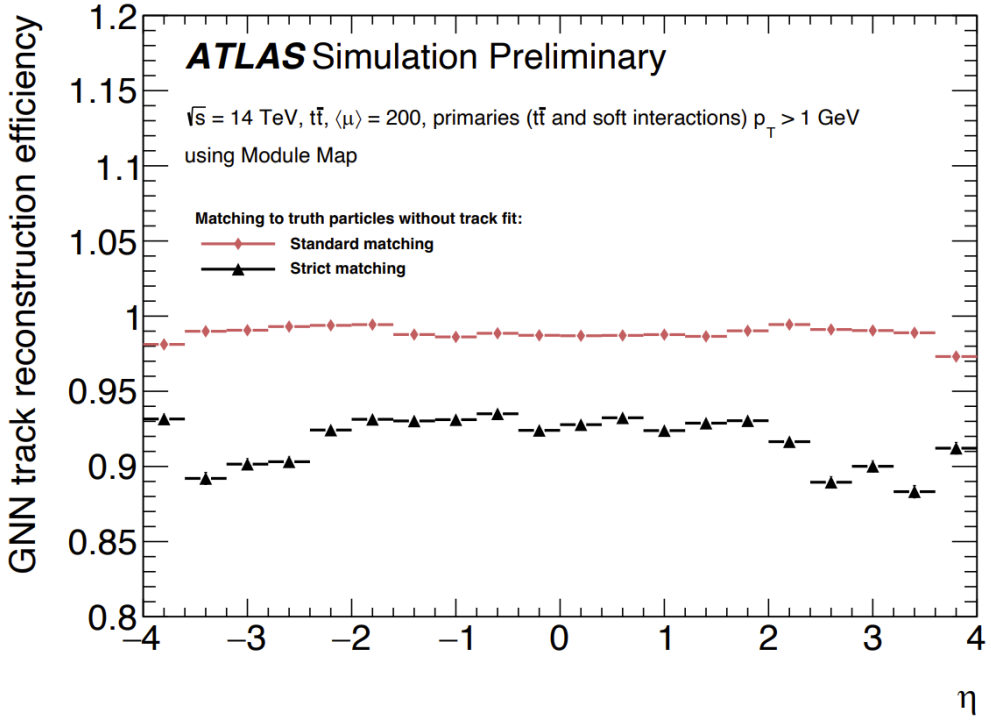
- What is the goal?
- Once moved to “offline tracking” essentially infinite time to reconstruct (although compute budget is limited)
- In ATLAS HL-LHC trigger (aka “Event Filter”), have O(microseconds) time to reconstruct, maybe with some dip in efficiency
- In some experiments (e.g. LHCb), aim to trigger on (essentially) all events, and perform on-the-fly full event reconstruction. In that case, target O(milliseconds) reconstruction with high accuracy

SHORTCOMINGS OF GNN4ITK

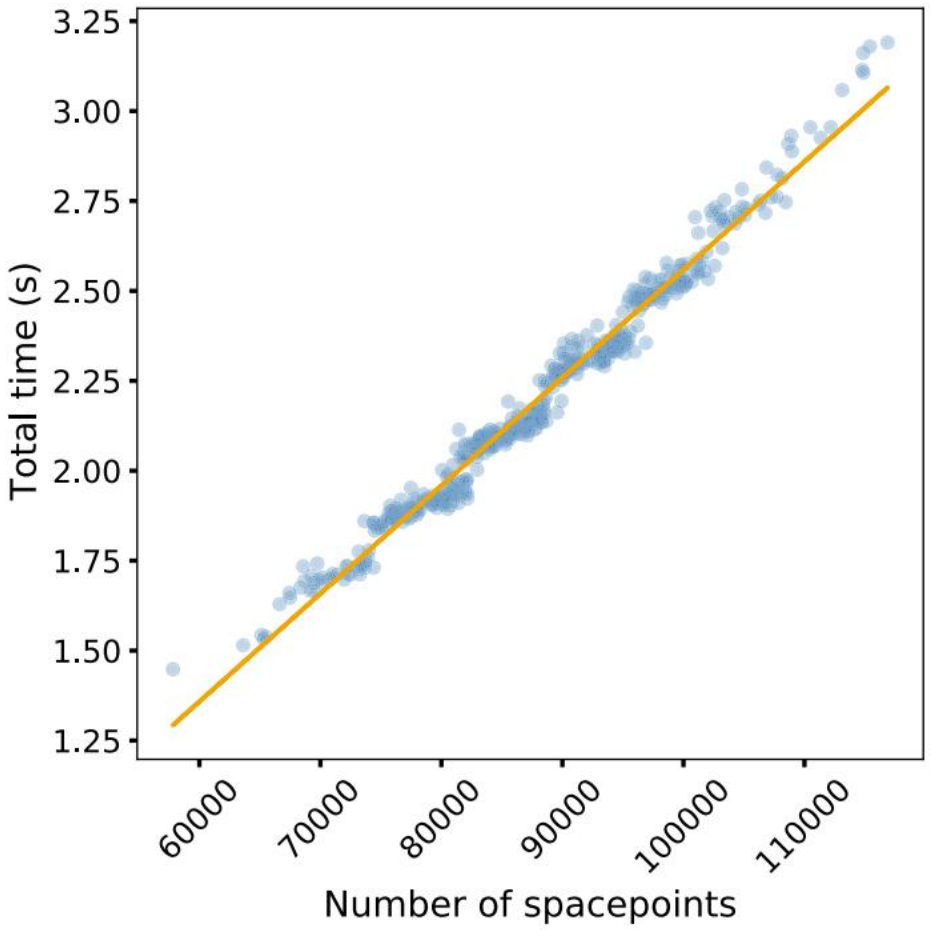
ACCURACY SHORTCOMINGS

- Not perfect tracking efficiency
- Poor performance in barrel strip modules





THROUGHPUT SHORTCOMINGS

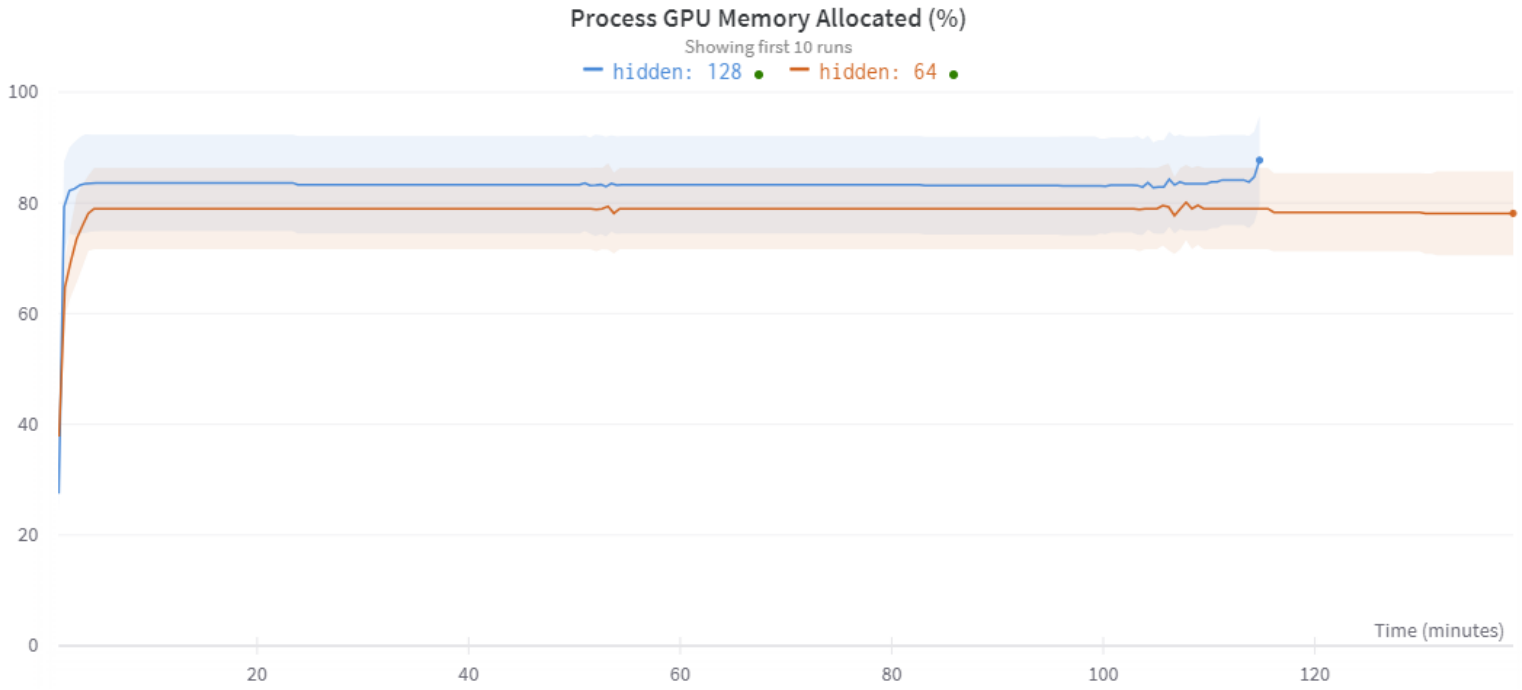


	Baseline	Faiss	cuGraph	AMP	FRNN
Data Loading	0.0022 ± 0.0003	0.0021 ± 0.0003	0.0023 ± 0.0003	0.0022 ± 0.0003	0.0022 ± 0.0003
Embedding	0.02 ± 0.003	0.02 ± 0.003	0.02 ± 0.003	0.0067 ± 0.0007	0.0067 ± 0.0007
Build Edges	12 ± 2.64	0.54 ± 0.07	0.53 ± 0.07	0.53 ± 0.07	0.04 ± 0.01
Filtering	0.7 ± 0.15	0.7 ± 0.15	0.7 ± 0.15	0.37 ± 0.08	0.37 ± 0.08
GNN	0.17 ± 0.03	0.17 ± 0.03	0.17 ± 0.03	0.17 ± 0.03	0.17 ± 0.03
Labeling	2.2 ± 0.3	2.1 ± 0.3	0.11 ± 0.01	0.09 ± 0.008	0.09 ± 0.008
Total time	$15 \pm 3.$	3.6 ± 0.6	1.6 ± 0.3	1.2 ± 0.2	0.7 ± 0.1

- Physics is important, but GNNs shine in scaling behavior
- When development began, graph-based pipeline started required 15 sec for TrackML
- Implemented custom Fixed Radius Nearest Neighbor (FRNN) algo., cuGraph Connected Components algo., and Mixed Precision inference
- Now have sub-second TrackML inference on 16Gb V100 GPU
- Inference time scales approximately linearly across size of event, in TrackML

TRAINING COST SHORTCOMINGS

- Even with the largest available GPUs (80Gb A100), still max out the memory with a relatively "small" GNN - 100k-1m parameters



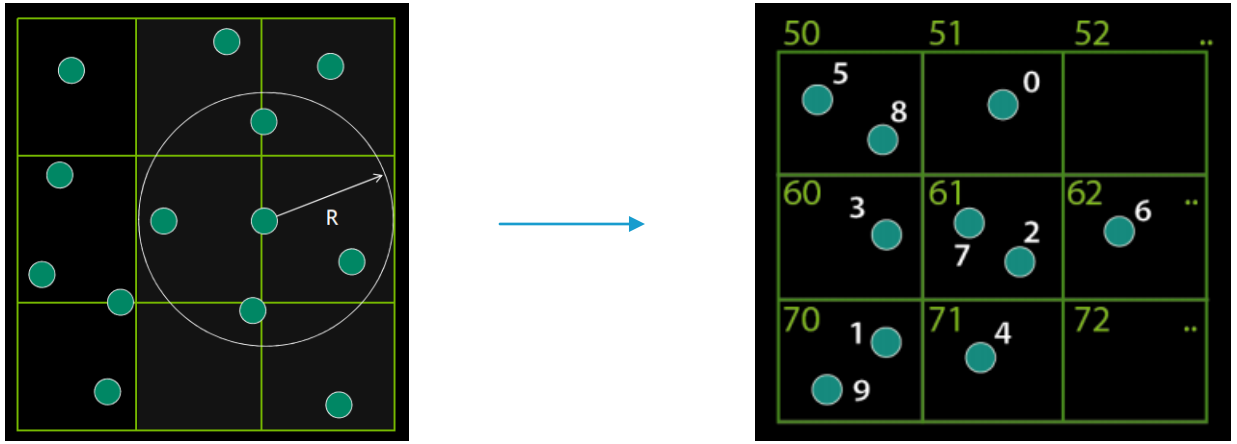
- What about if we want to go from spacepoints to clusters (300k nodes to 400k nodes), or to the next higher luminosity detector, or we want to train a very large GNN?

FASTER GNN TRACKING

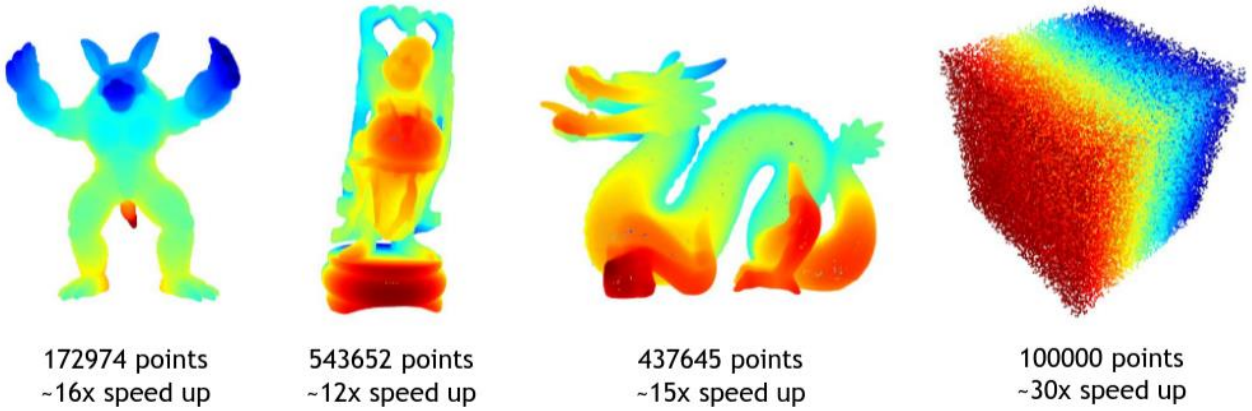
FAST GRAPH CONSTRUCTION

- Nearest neighbor search is a **bottleneck** of the graph construction stage
- FAISS finding K=500 for N=100,000 ~ 700ms
- KNN is overkill – we don't need explicit list of K sorted neighbours
- Built custom library on **Fixed Radius Nearest Neighbour (FRNN)** search algorithm
- Cell-by-cell grid search is *much* faster: [*The complexity of finding fixed-radius near neighbors*. Bentley, et al 1977]

Fast fixed-radius nearest neighbors: Interactive Million-particle Fluids, Hoetzlein (NVIDIA), 2014



- Fixed Radius NN Search vs Pytorch3D's KNN



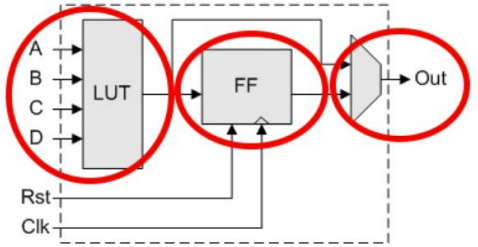
Accelerating NN Search on CUDA for Learning Point Clouds, Xue 2020

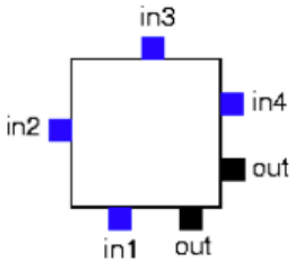
FASTER SEGMENTATION

- Many graph operations can be parallelised, and therefore are well-suited to GPU implementation
- Connected components is one algorithm, which can be parallelised
- Scipy has CPU version, which loops over each node with "Depth-first Search"
- CuGraph searches many "frontiers" simultaneously

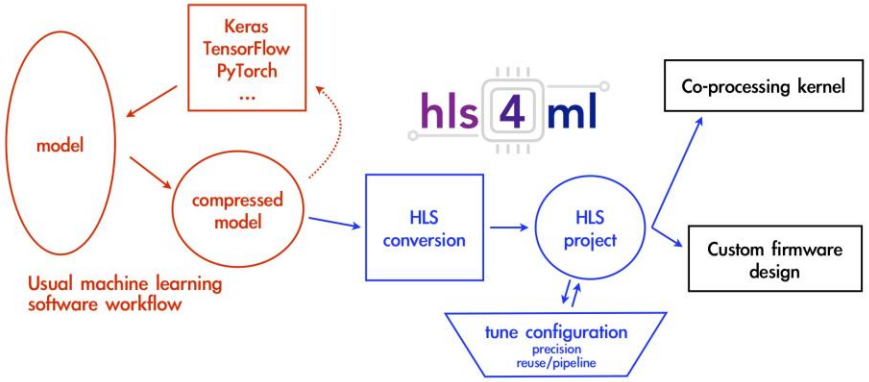
	Baseline	Faiss	cuGraph	AMP	FRNN
Data Loading	0.0022 ± 0.0003	0.0021 ± 0.0003	0.0023 ± 0.0003	0.0022 ± 0.0003	0.0022 ± 0.0003
Embedding	0.02 ± 0.003	0.02 ± 0.003	0.02 ± 0.003	0.0067 ± 0.0007	0.0067 ± 0.0007
Build Edges	12 ± 2.64	0.54 ± 0.07	0.53 ± 0.07	0.53 ± 0.07	0.04 ± 0.01
Filtering	0.7 ± 0.15	0.7 ± 0.15	0.7 ± 0.15	0.37 ± 0.08	0.37 ± 0.08
GNN	0.17 ± 0.03	0.17 ± 0.03	0.17 ± 0.03	0.17 ± 0.03	0.17 ± 0.03
Labeling	2.2 ± 0.3	2.1 ± 0.3	0.11 ± 0.01	0.09 ± 0.008	0.09 ± 0.008
Total time	$15 \pm 3.$	3.6 ± 0.6	1.6 ± 0.3	1.2 ± 0.2	0.7 ± 0.1

FASTER HARDWARE

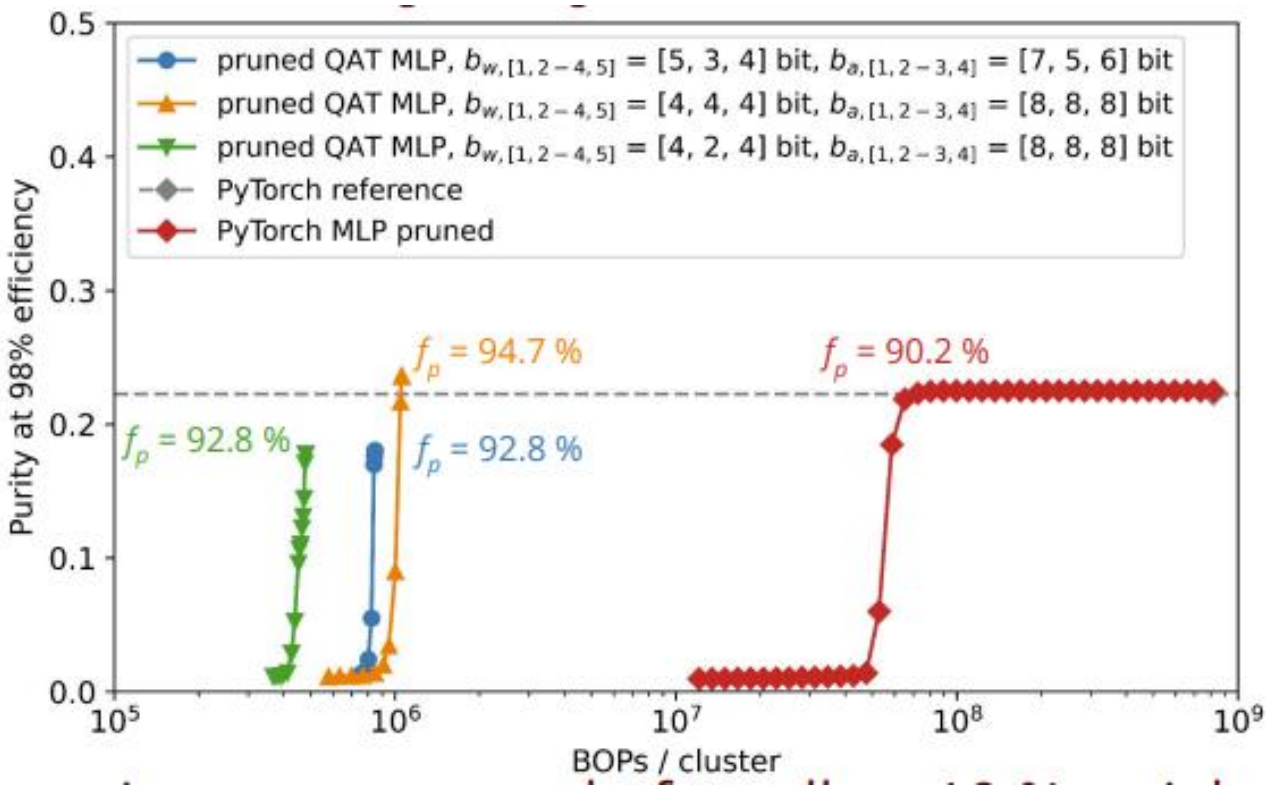




- GPUs are used **by default** in our ML pipeline for training and inference
- But FPGAs are a very low-latency option
- Field Programmable Gate Array are able to compile a program to hardware, using Logic Elements and IO – essentially Look-up Tables (LUTs) that can capture any 4-input Boolean operators
- Typically need to write functions from scratch, but HLS4ML is an effort to automatically compile Python ML frameworks to HLS (High Level Synthesis) language, then to the hardware language



PRUNING



- Hard to beat GPUs for big matrix multiplication – can be very efficiently multi-threaded
- But large models typically only have a small subset of "important" weights (c.f. Lottery Ticket Hypothesis)
- We can simply set those weights ~0 to exactly 0, but on GPU one still needs to run the full matrix multiplication
- On FPGA, since the multiplication is in series, we can *skip* those 0 entries, and get a speed-up!

QUANTIZATION

- Similar to pruning, since everything is done manually on FPGA, we can choose how much precision we use to speed up
- Can simply reduce precision of weights and operations after training
- However there is *significant* improvement in performance using "Quantisation-aware Training" (QAT)

Quantization-aware Training (QAT)

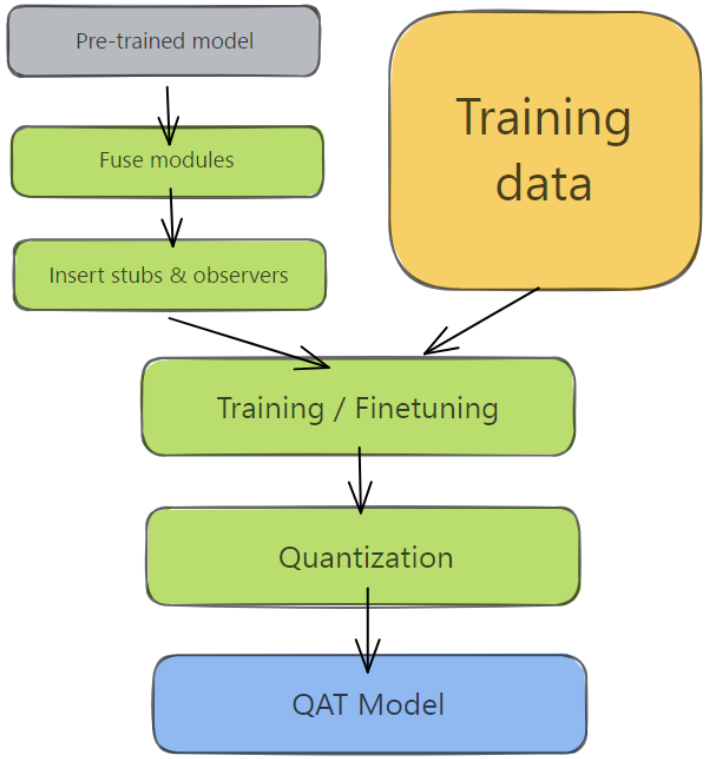
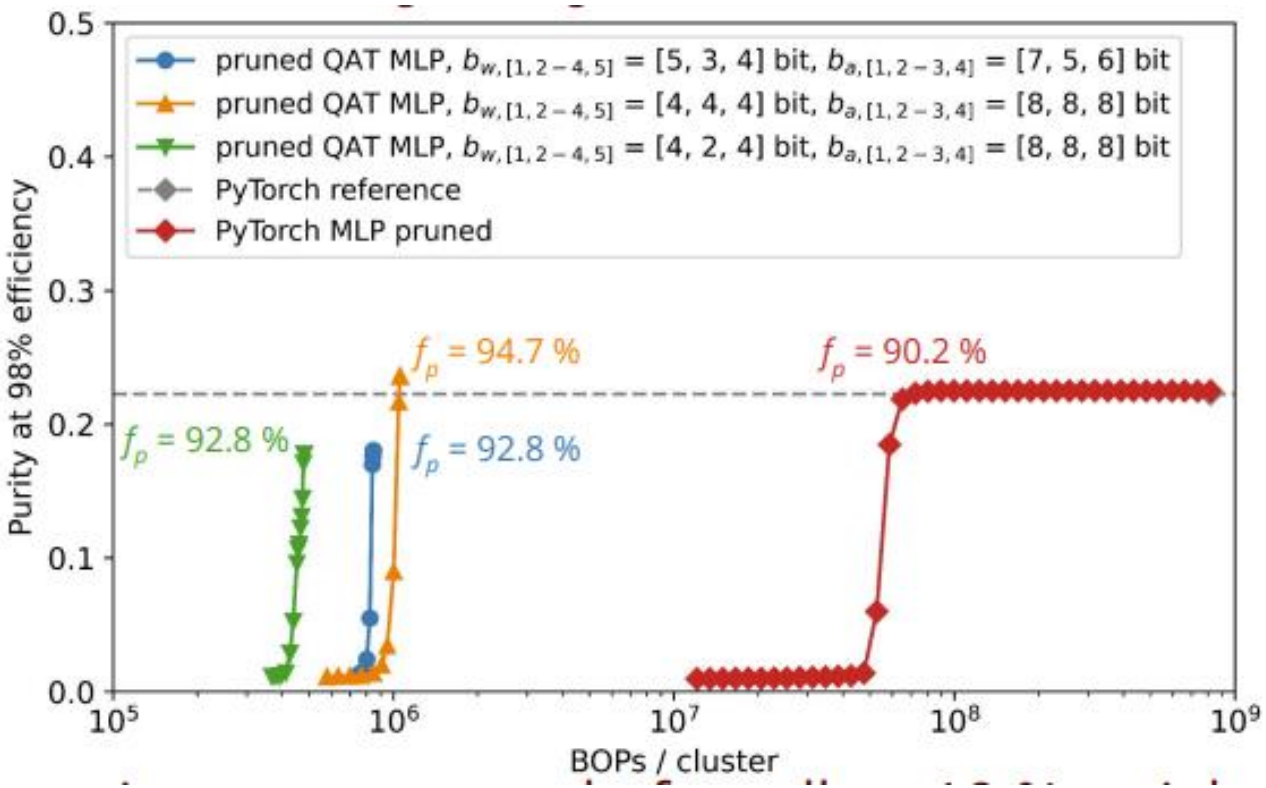


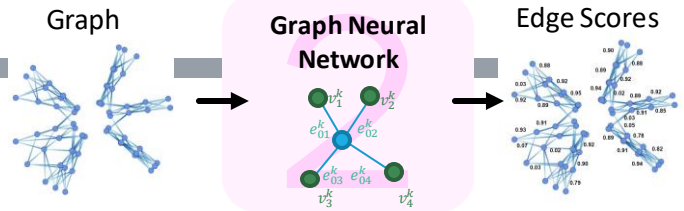
Fig 5. Steps in Quantization-Aware Training

QUANTIZATION

- Similar to pruning, since everything is done manually on FPGA, we can choose how much precision we use to speed up
- Can simply reduce precision of weights and operations after training
- However there is *significant* improvement in performance using "Quantisation-aware Training" (QAT)



MORE ACCURATE GNN TRACKING

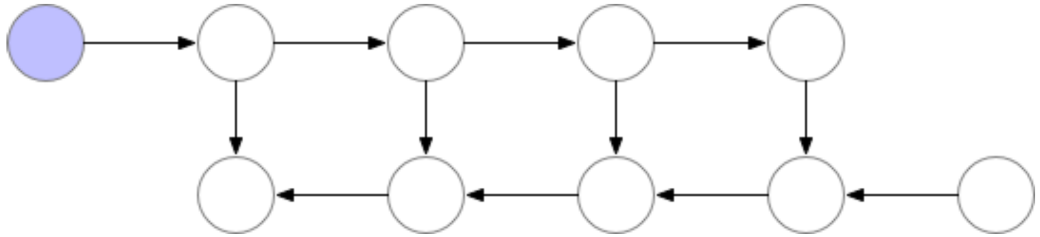


CHECKPOINTING

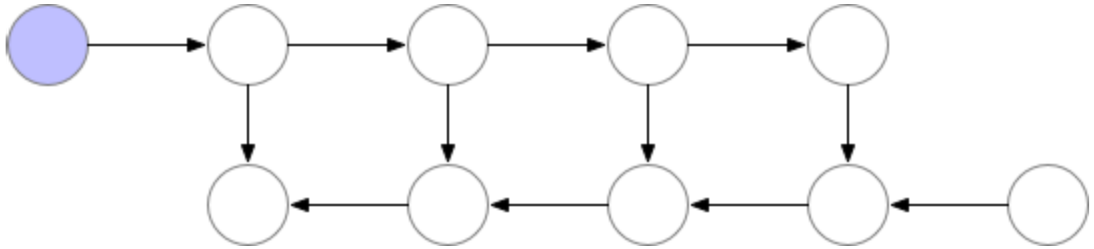
- Graph construction leads to very large graphs O(1m) edges, cannot fit training on A100 GPU with 32Gb memory
- Should not split the graphs up (leads to lower GNN accuracy)
- Solution A: Were previously using a compromising form of "gradient checkpointing" – reduced memory by 4x
- Now using maximal checkpointing, reduce memory further by 2x – just fits on A100

Edge Labeling

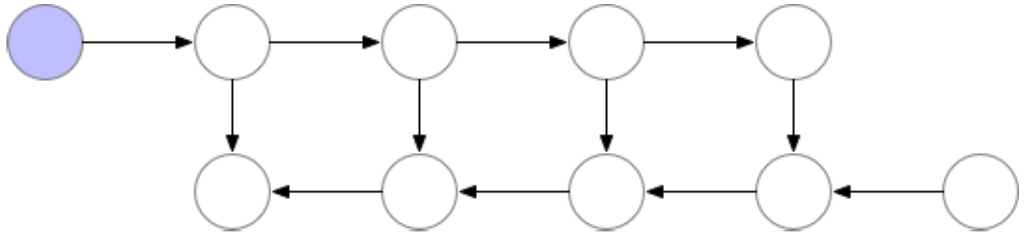
No checkpointing

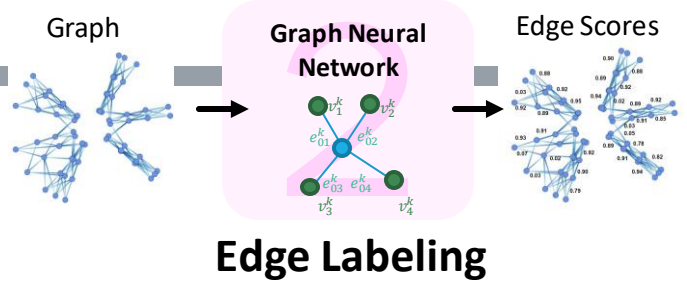


Partial checkpointing



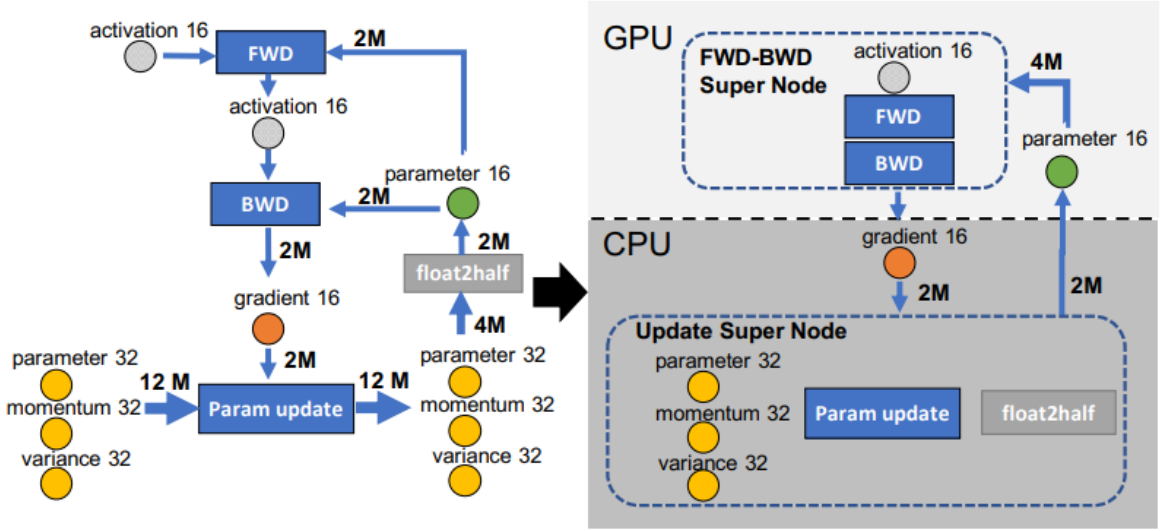
Maximal checkpointing





TRAINING SOLUTIONS

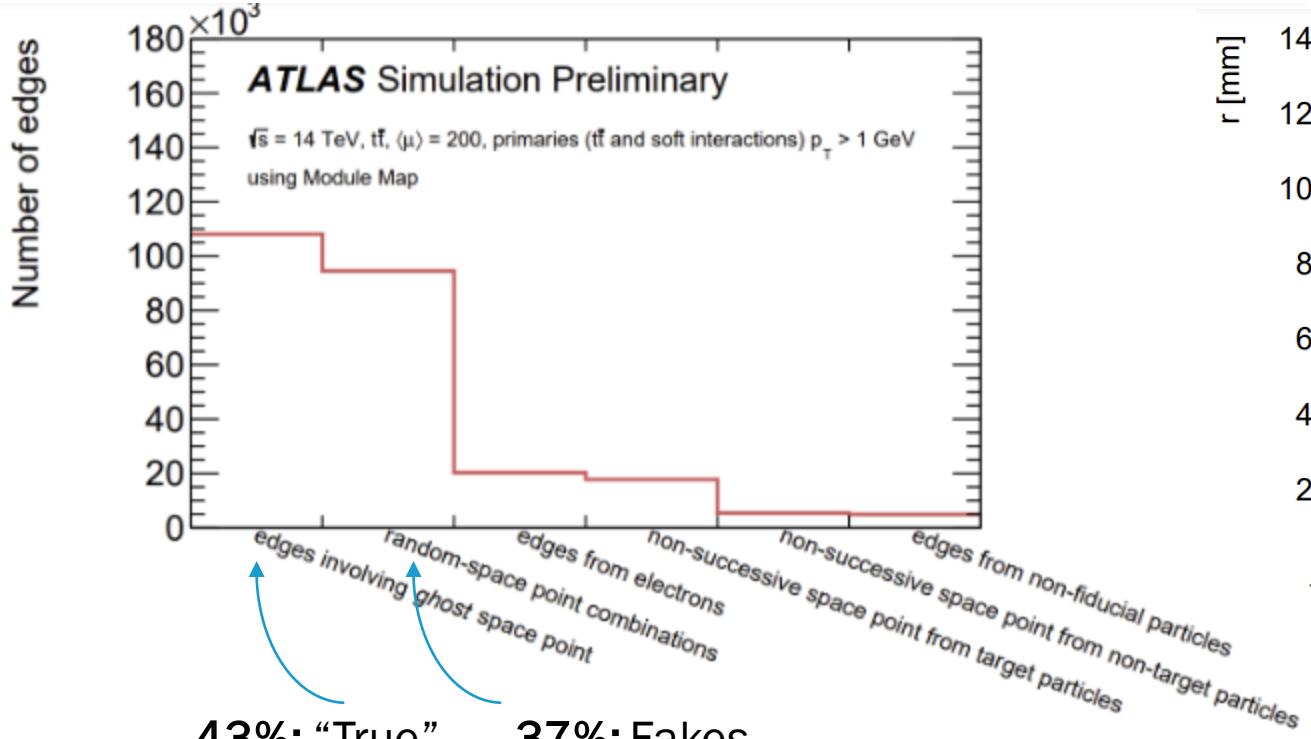
- Solution B: Model offloading
- Each layer of GNN placed on GPU for forward and backward pass, but held on CPU otherwise
- Works well with TensorFlow, enabling training of O(1m) edge graphs
- Unable to integrate with Pytorch pipeline



ZeRO-Offload: Democratizing Billion-Scale Model Training
arXiv: 2101.06840

BARREL STRIP MISCLASSIFICATION

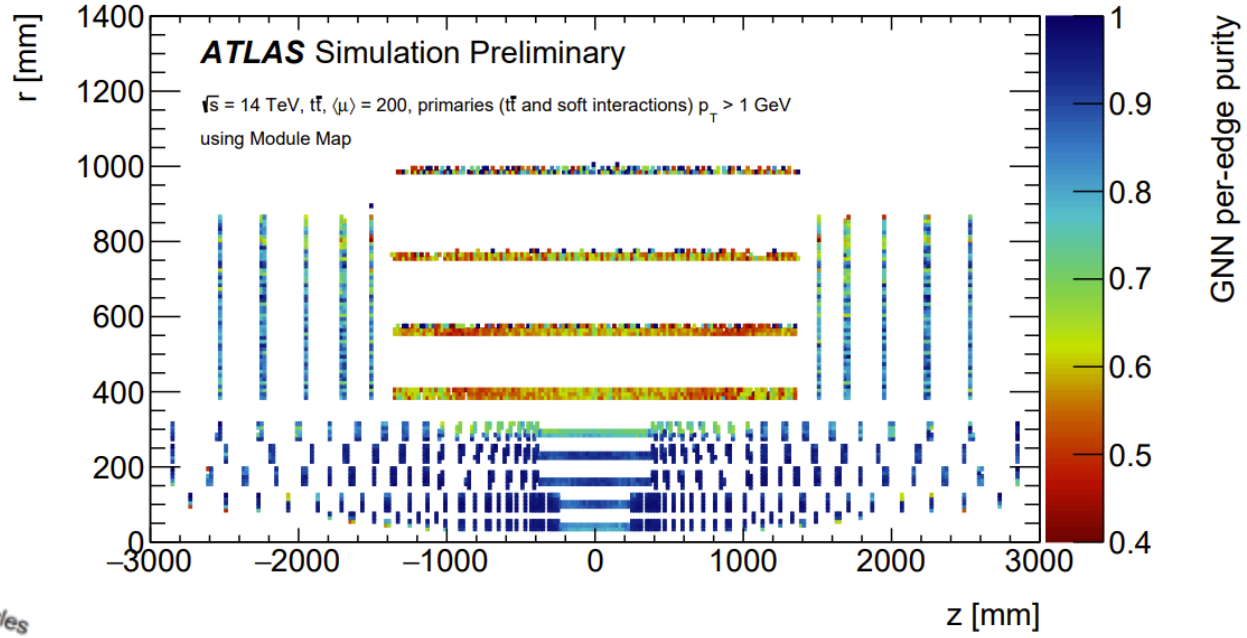
Nature of false positive edges



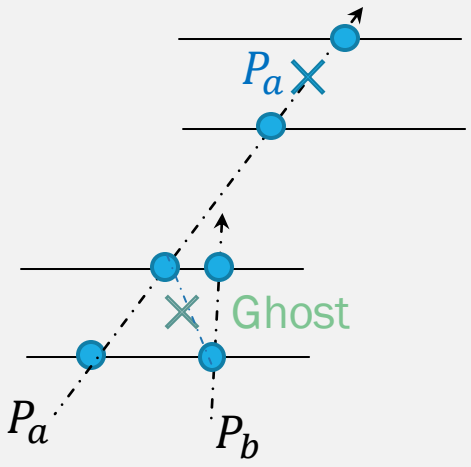
43%: "True" ghosts

37%: Fakes

Location of false positive edges



BARREL STRIP MISCLASSIFICATION



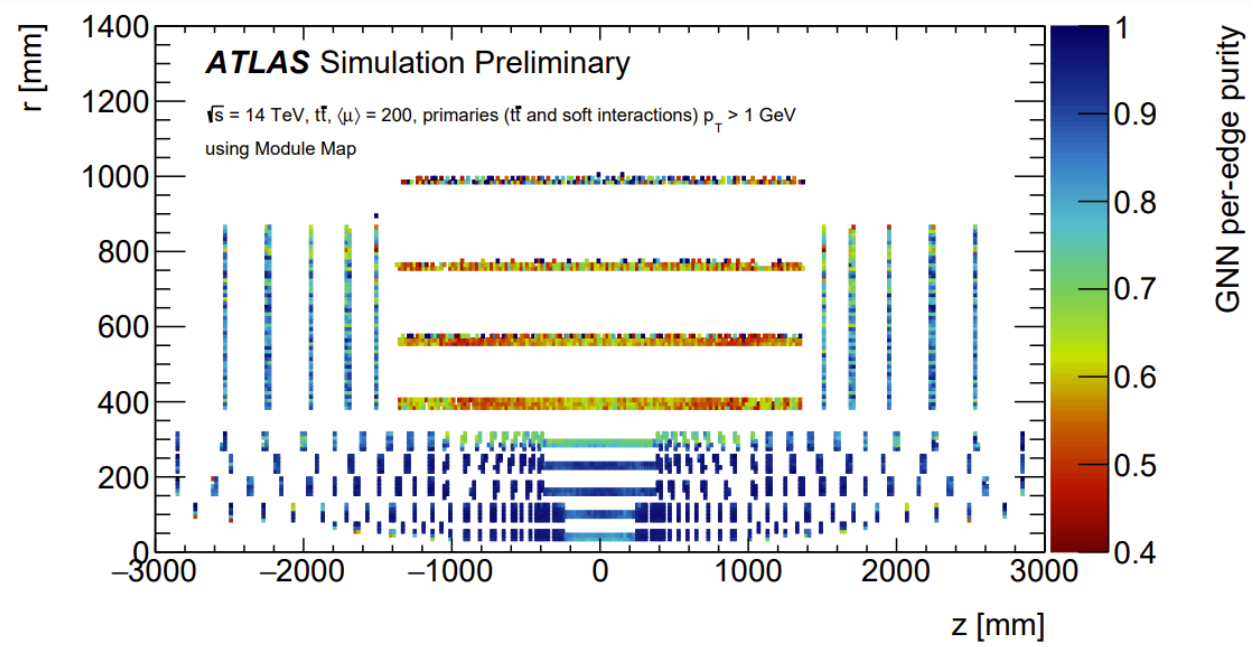
"True" ghost edges: 43%

Edges between SP from particle A, and a ghost SP of clusters from particle A and particle B. I.e. The GNN is "right", the construction is "wrong"

Fake edges: 37%

Edges between SP from particle A and particle B. i.e. The GNN is "wrong"

Location of false positive edges



STRIP MODULES: GHOSTS AND Z-RESOLUTION

- Since spacepoints are constructed from pairs of clusters in the strip, could mis-construct and form a ghost
- These ghosts can be cleaned up in later stages of the reconstruction chain
- *However*, even for correctly matched clusters, there remains low z-resolution
- Consider this example
- Easily confuses GNN!
- Could fix by including underlying cluster information somehow… (e.g. heterogeneous node features)

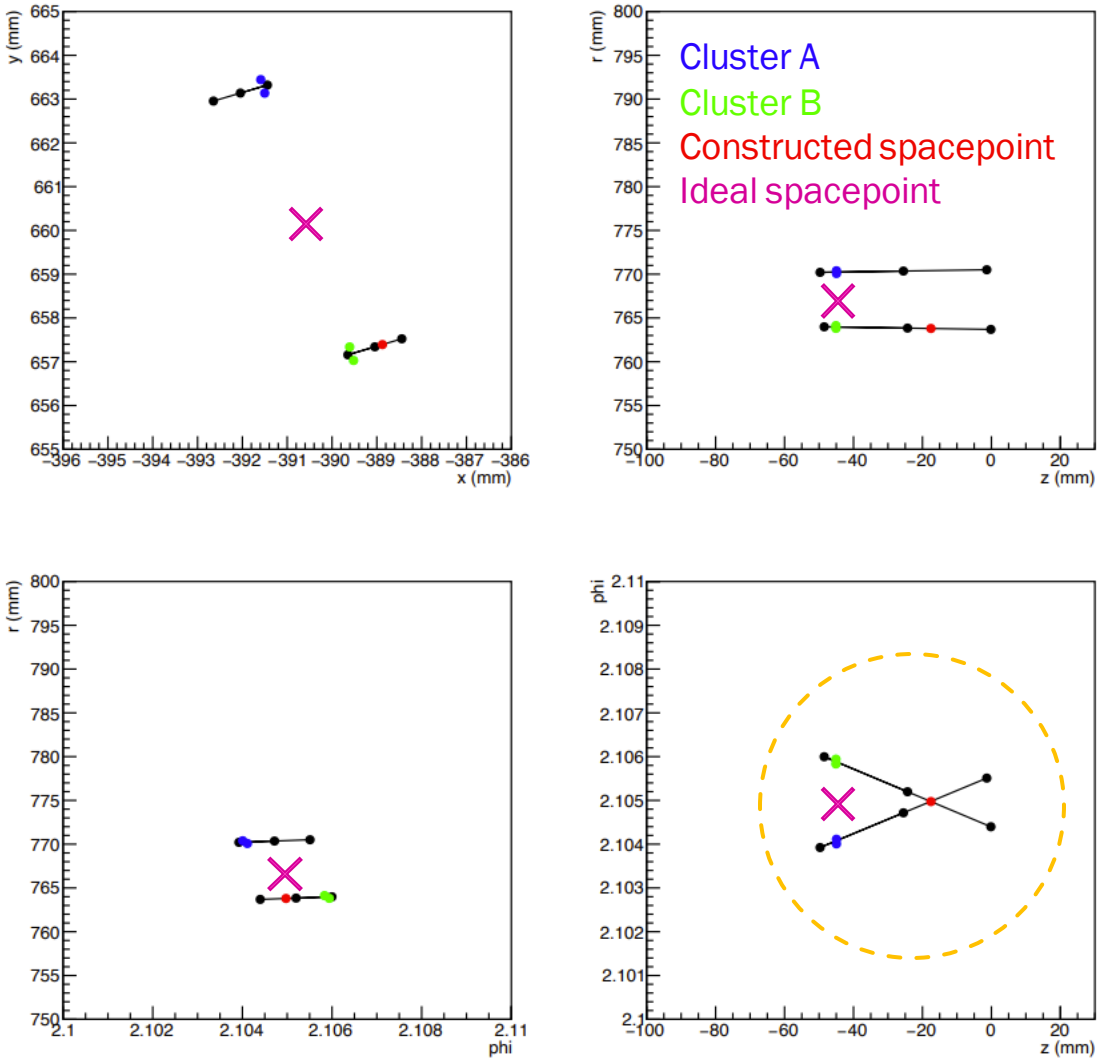
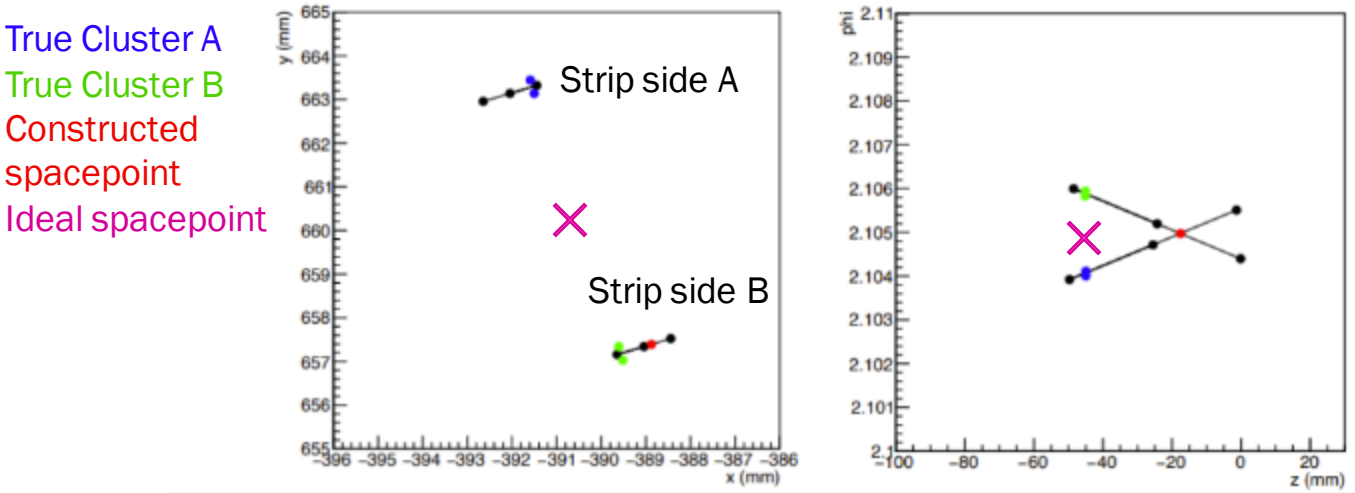
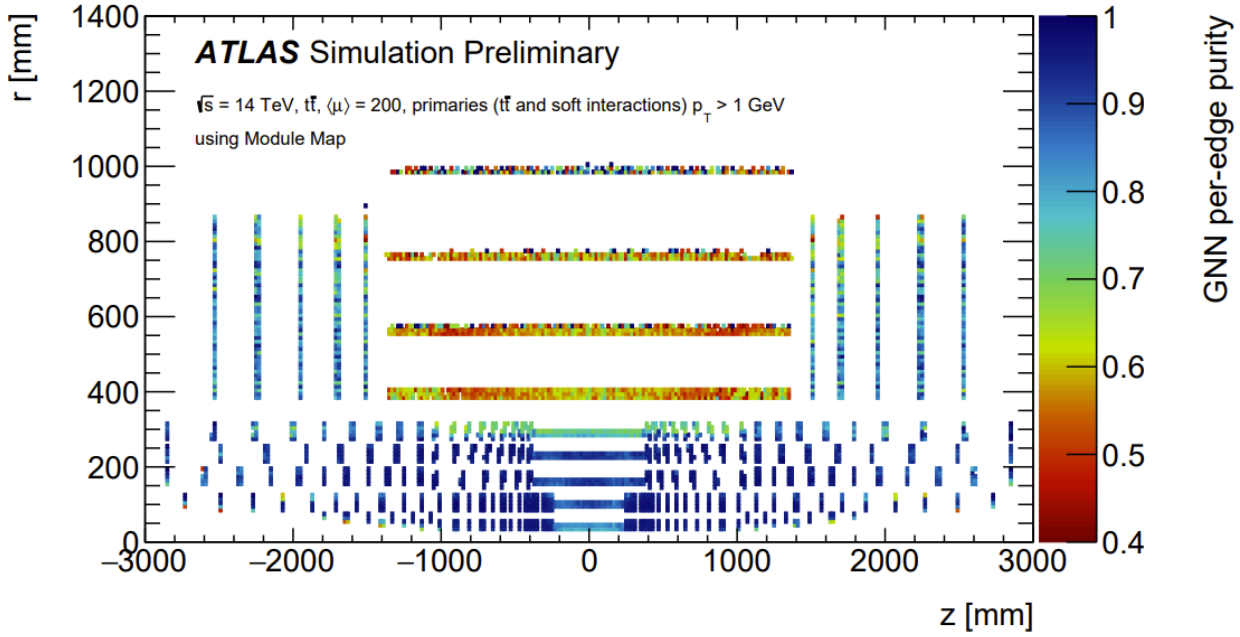


Image courtesy of Jan Stark – thanks!

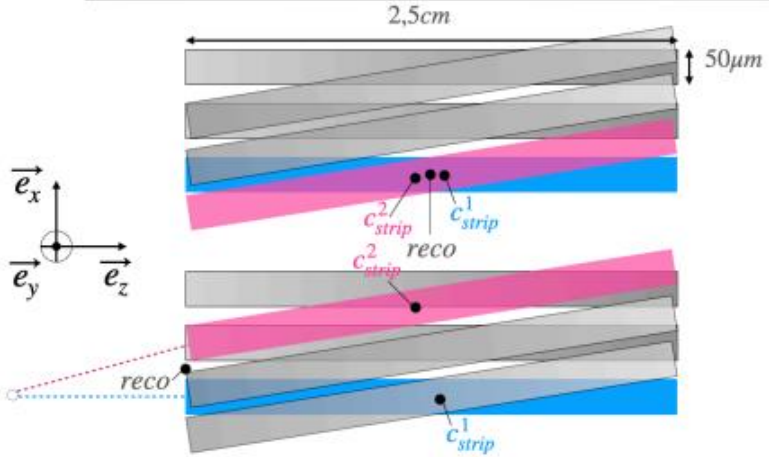
CURRENT PIPELINE PERFORMANCE

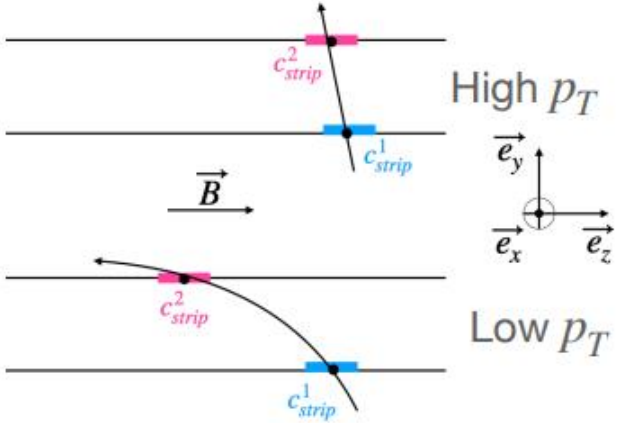
- Consider GNN performance on edge classification across pseudorapidity η
- Drop in performance at low η – what is special about this region?
- Low performance in barrel strips, where spacepoints are built from two strip clusters
- Spacepoint position may be far from "ideal" position – i.e. midpoint between ground truth clusters
- How can we attach these two sets of cluster features? Pixel spacepoints only have one set of cluster features…





STRIP low spatial resolution and Heterogenous Data

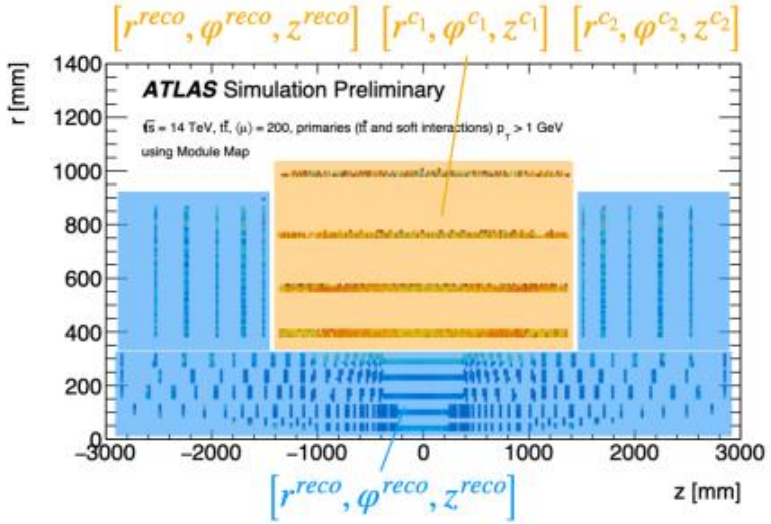




$$r^{reco}, \varphi^{reco}, z^{reco} = f_{strip}(c^1_{strip}, c^2_{strip})$$

The mechanism that led to the poor purity in CTD2022 results is understood: straight line approximation used in the ATLAS space point reconstruction leads to poor z resolution ($O(cm)$!) at low p_T

Impossible to exactly reconstruct space point position without knowing curvature of the track !



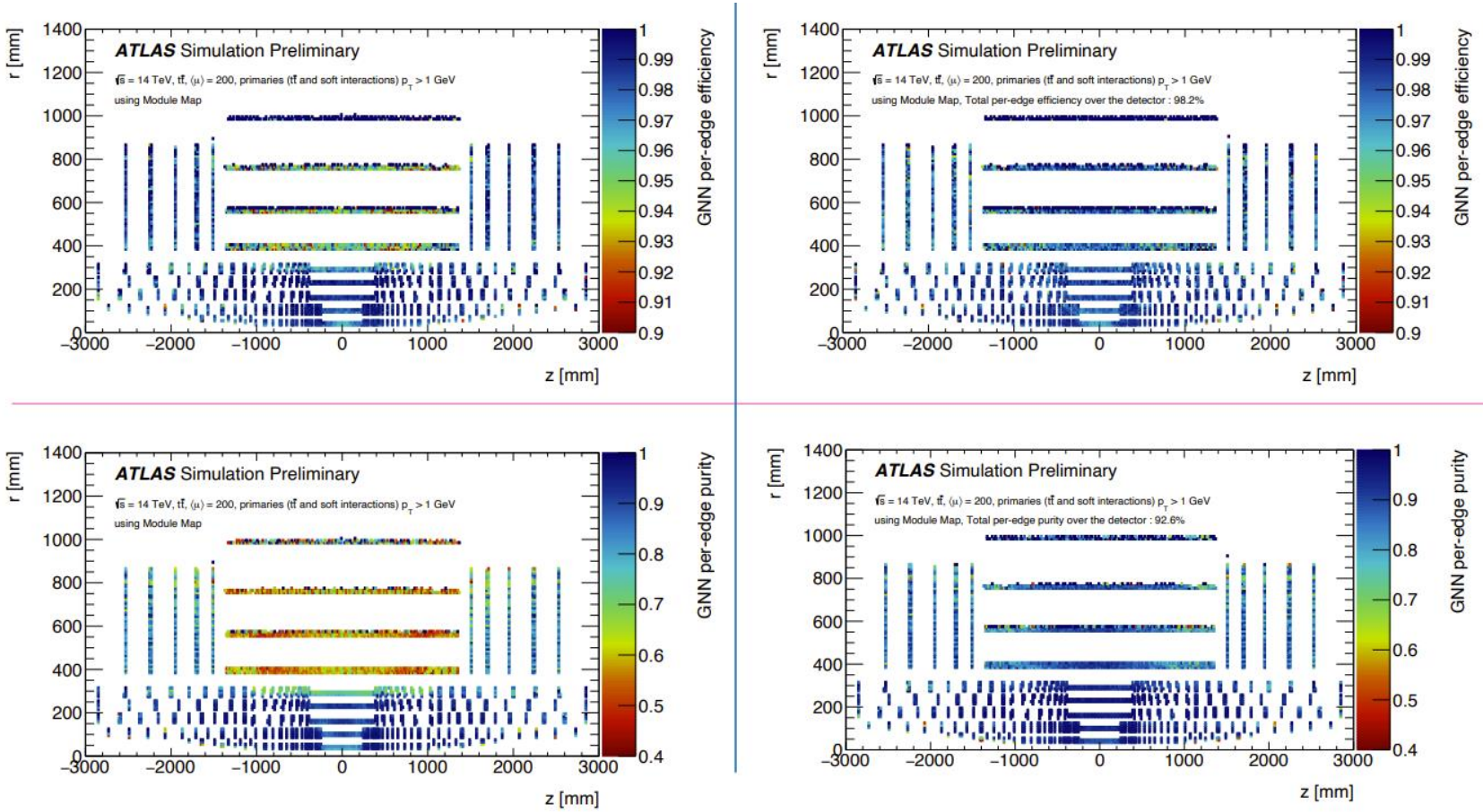
Key Idea: Give as node features in STRIP BARREL the **STRIP clusters data**
Hopefully the GNN will be able to learn a **better representation of the Space Point**

GNN combines space points into tracks, i.e. has access to curvature !

IMPROVEMENT FROM INCLUDING CLUSTER INFORMATION

Only spacepoint information

Spacepoint+cluster information



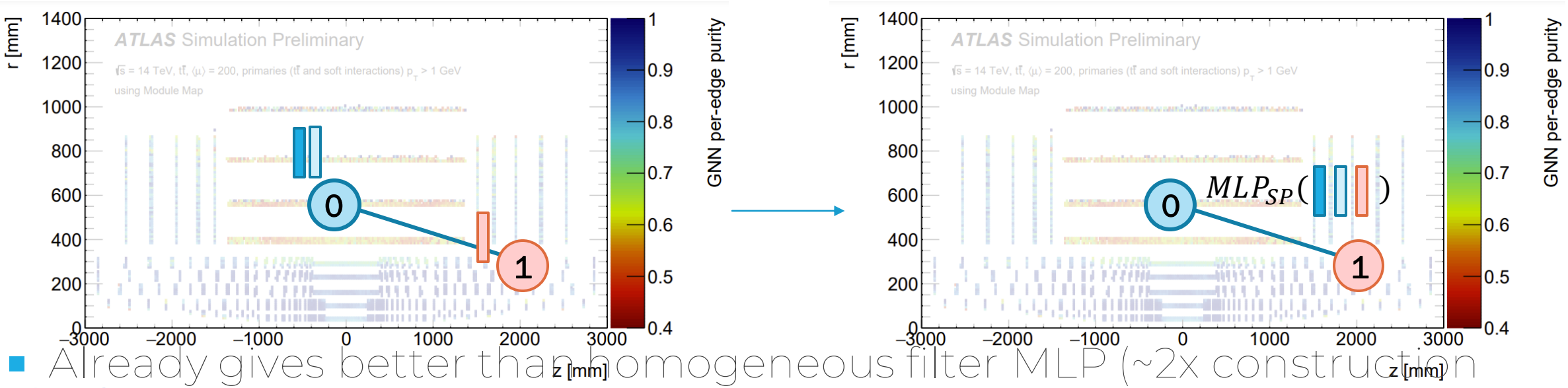
ONGOING WORK: HETEROGENEOUS NODE FEATURES

- Motivated by inconsistent performance across detector:
- Currently each node in graph uses same input feature set – spacepoint $\mathbf{s} = (r, \phi, z)$
- We could imagine using cluster-level information, e.g. position and shape of energy deposit
- *But:* this is not consistent across detector. Need different node and edge networks depending on detector region

ONGOING WORK: HETEROGENEOUS NODE FEATURES

- To get intuition, consider simple filter MLP applied to two pixel nodes:

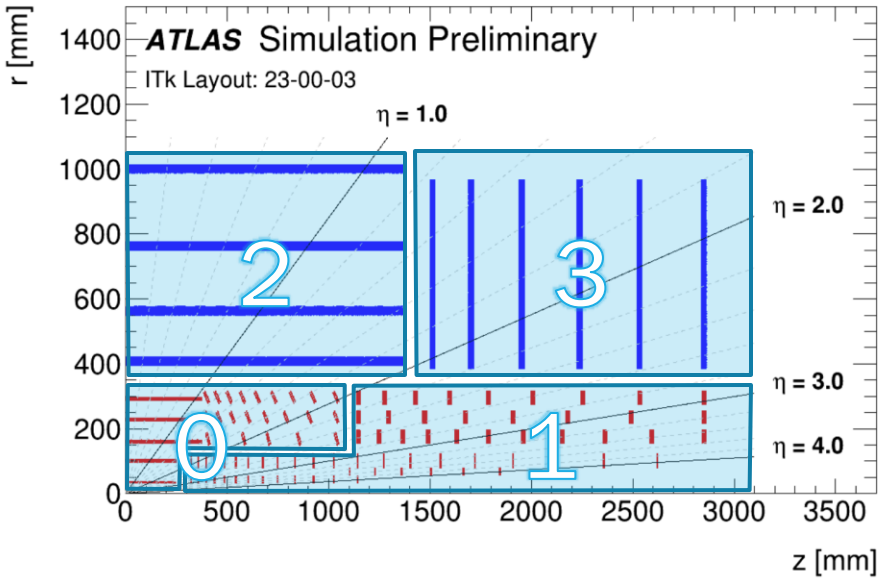
- To apply a filter MLP to a pixel (single cluster) and strip (double cluster) node combination, need a *different* MLP:

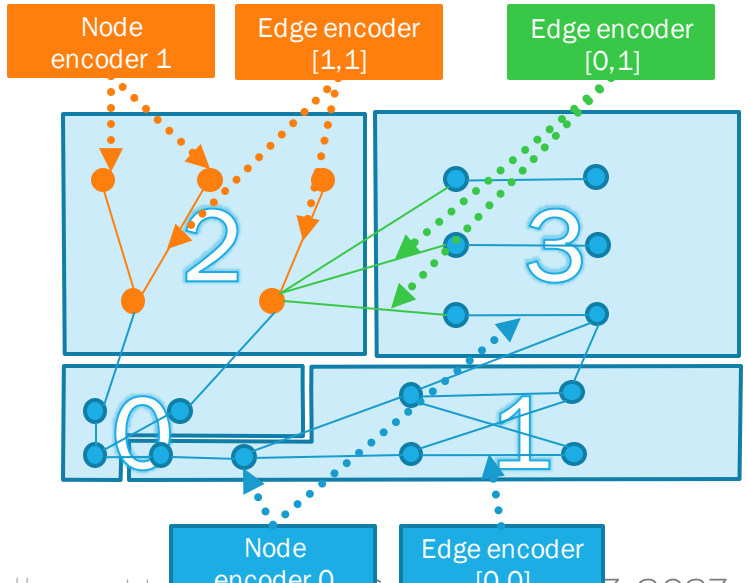


- Already gives better than homogeneous filter MLP (~2x construction purity)

ONGOING WORK: HETEROGENEOUS GRAPH NEURAL NETWORK

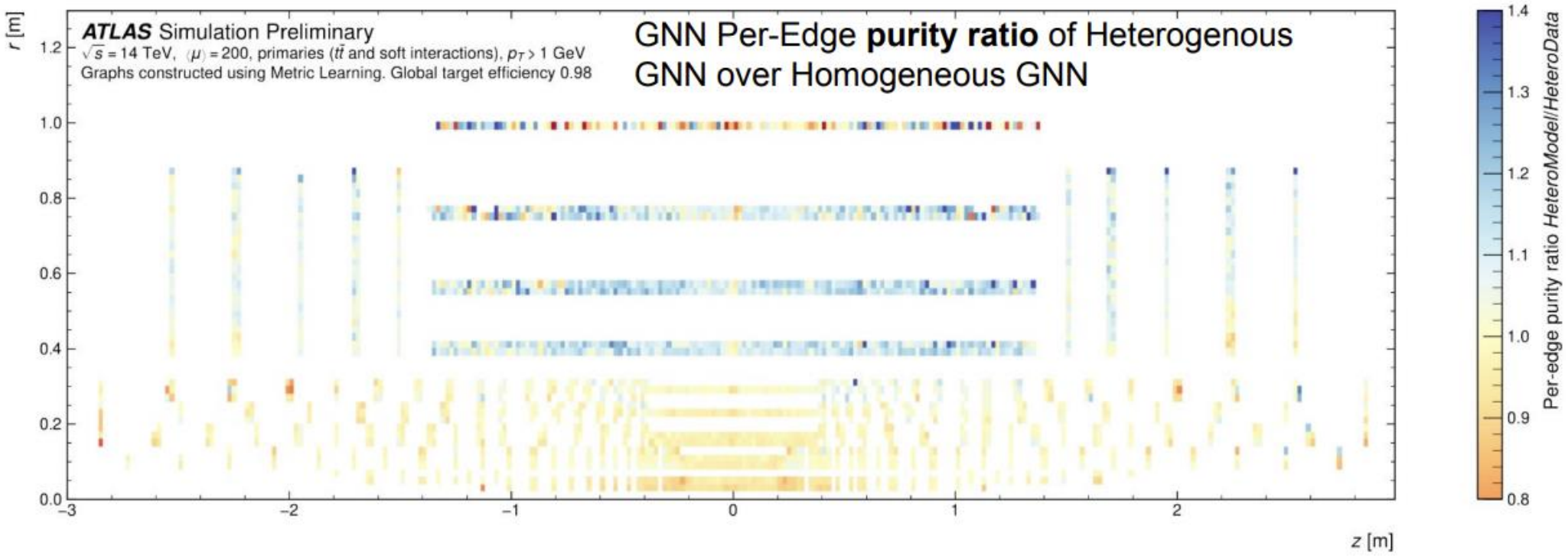
- Exact same logic applies to GNN networks
- For a four-region heterogeneous GNN, we have four node encoders/networks (N_0, N_1, N_2, N_3) and ten edge encoders/networks ($E_{00}, E_{01}, E_{02}, E_{03}, E_{11}, \ldots, E_{34}, E_{44}$)
- Thus, is a larger model and takes longer to train
- But reduces GNN inefficiency and fake rate by approximately half





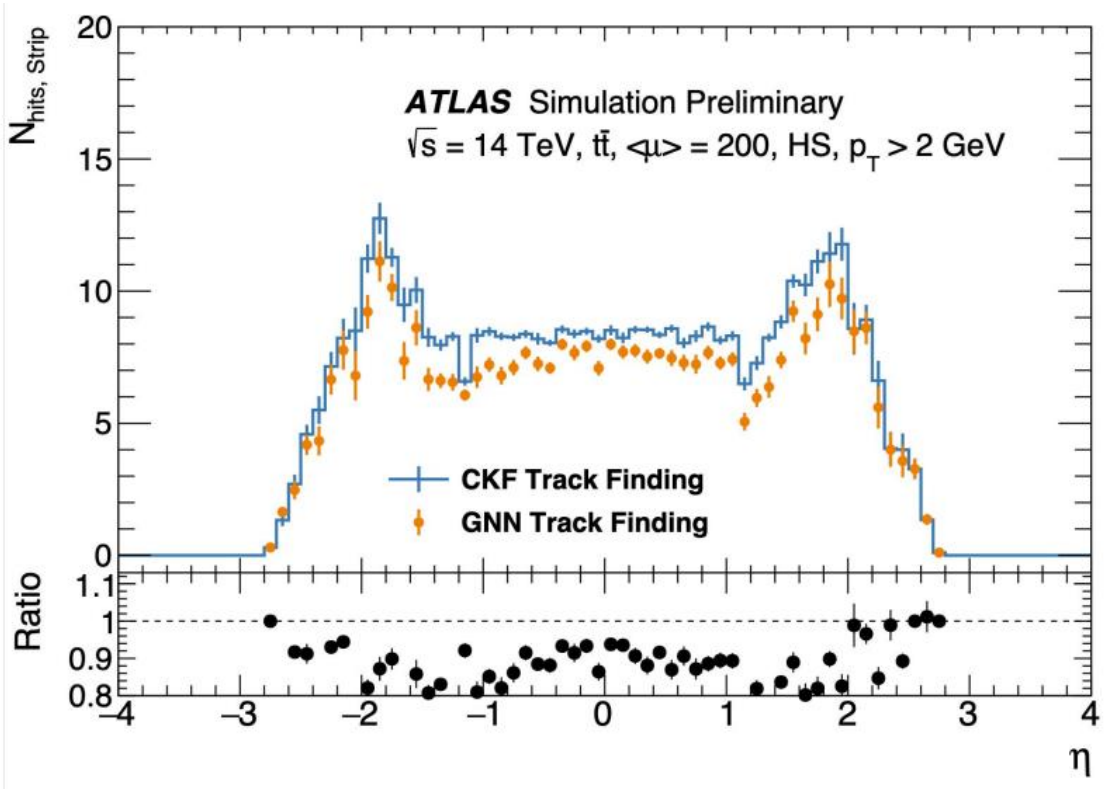
HETEROGENEOUS GNN PERFORMANCE

- The average total purity is 94% for both models
- Adding model heterogeneity results in up to 11% improvement in GNN per-edge purity in the Strip barrel region, with ~1% loss in the Pixel subsystem



HETEROGENEITY & THE MISSING HITS…

- Now we can see why we are missing hits: there are orphan clusters in the strip that are never constructed into spacepoints
- Would be great to have GNN that could handle both orphaned clusters and spacepoints…

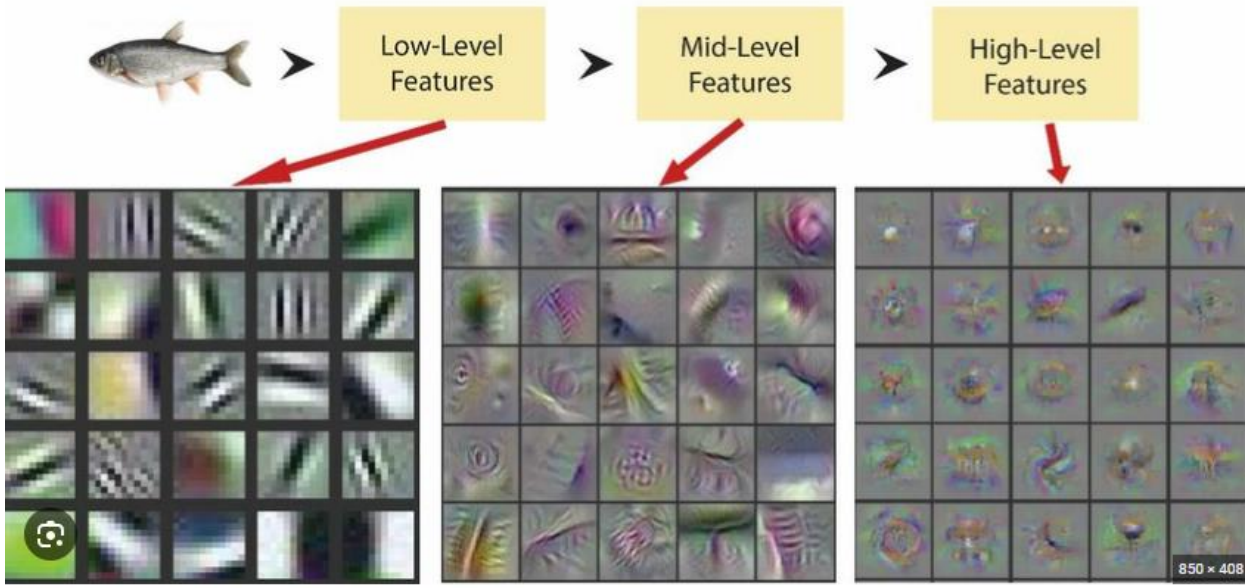


DIFFERENT APPROACHES TO GNN TRACKING

HIERARCHY

HIERARCHY

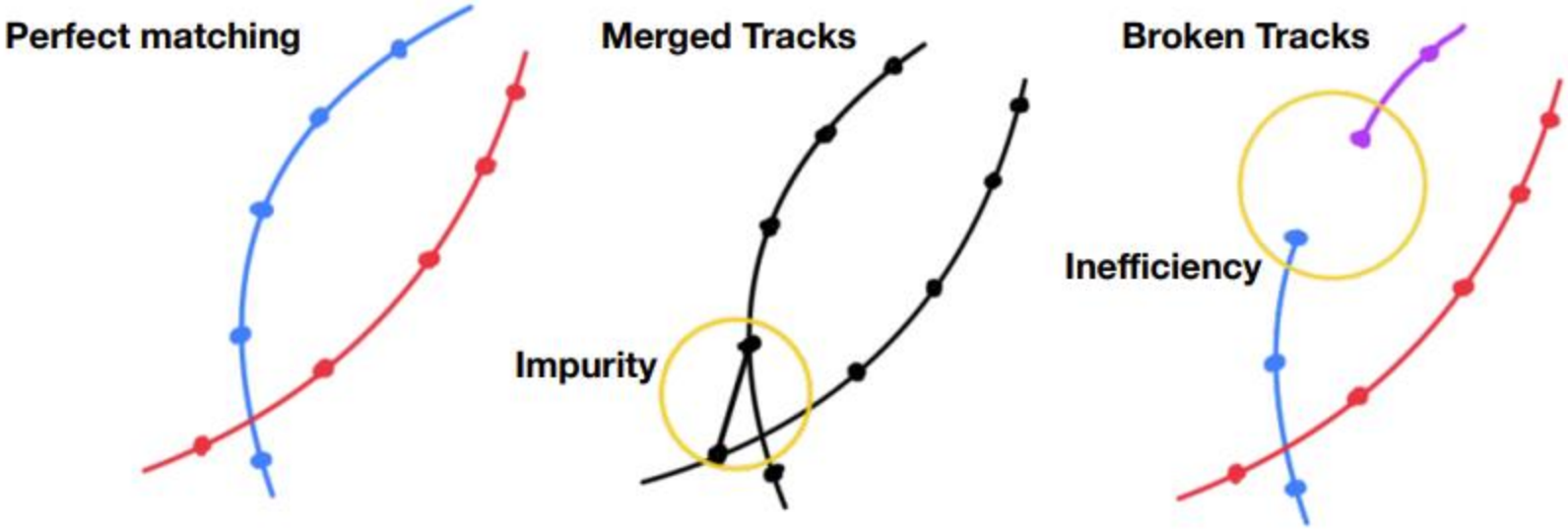
- So far, our whole pipeline has been fairly "vanilla" (but it still took a *lot* of R&D to get this all to work!)
- For example, every object in the graph is a spacepoint (or in the case of the heterogeneous GNN, either a strip spacepoint or pixel spacepoint)
- But there are other granularities in the system, e.g. "track-like" objects



- A hierarchical graph neural network is inspired by the different granularities of filter in a convolutional neural network

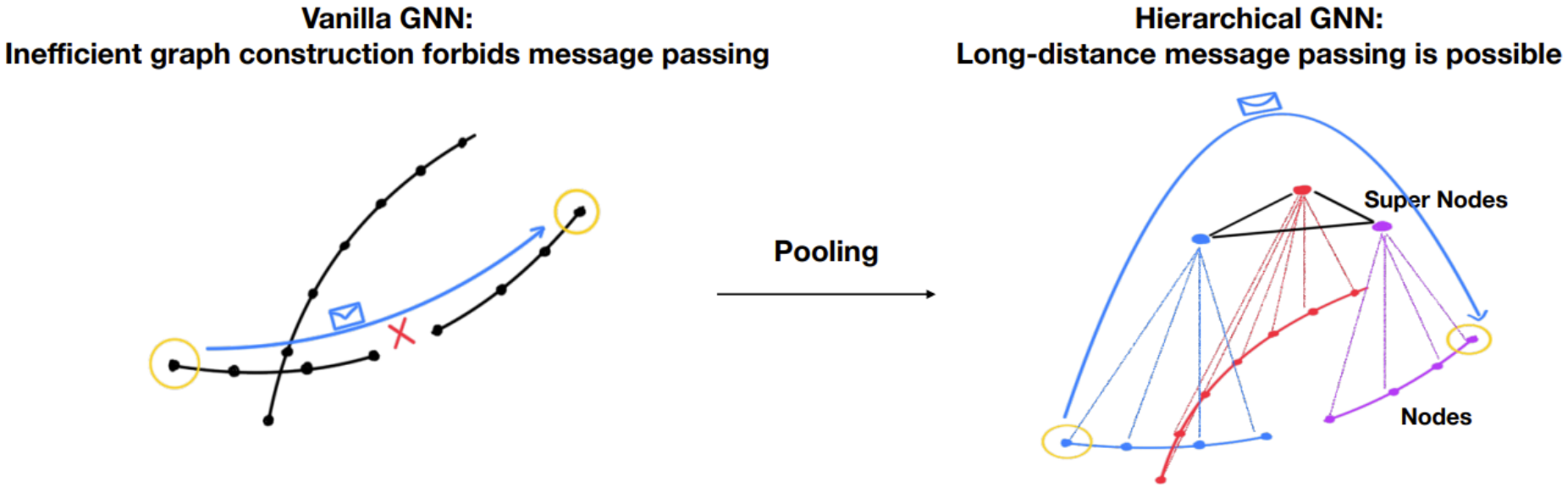
HIERARCHY

- Consider that in the GNN4ITk pipeline if a track is broken (a missing edge), there is no way to recover it



HIERARCHY

- Consider that in the GNN4ITk pipeline if a track is broken (a missing edge), there is no way to recover it
- However, if we could "pool" hits together into track-like supernodes, then we could reconnect them at some other granularity



HIERARCHY

- Consider that in the GNN4ITk pipeline if a track is broken (a missing edge), there is no way to recover it
- However, if we could "pool" hits together into track-like supernodes, then we could reconnect them at some other granularity
- Can deep dive into this later if there's interest. For now:

Tracking Goal	Feature	DiffPool	SAGPool	EdgePool	GMPool (ours)
Subquadratic scaling	Sparse	✗	✓	✓	✓
End-to-end trainable	Differentiable	✓	✓	✓	✓
Variable event size	Adaptive number of clusters	✗	✗	✓	✓
Many hits to many particles relationship	Soft assignment	✓	✗	✗	✓

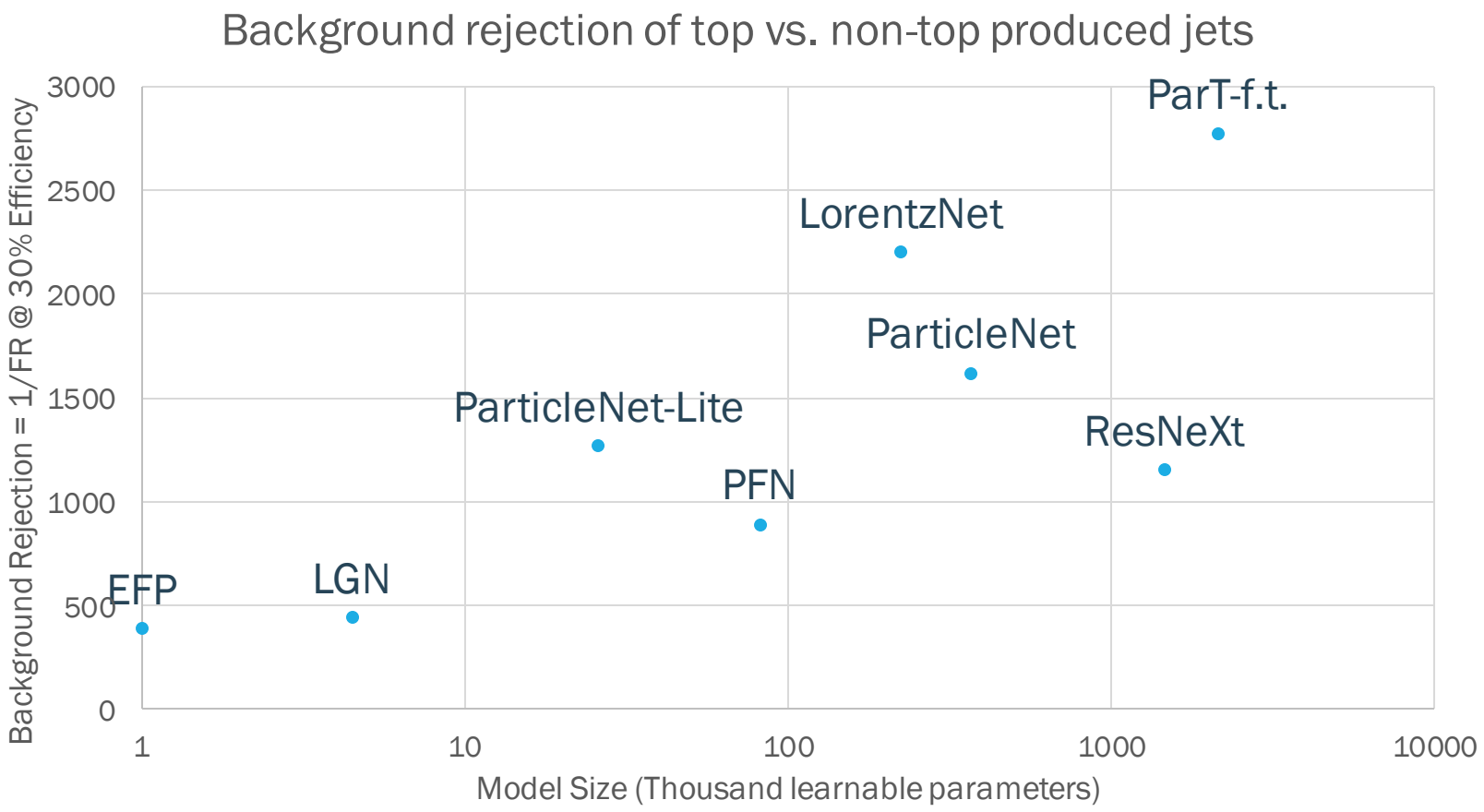
Models	E-GNN	E-HGNN	BC-HGNN	EC-GNN	Truth-CC
Efficiency	94.61%	95.60%	**97.86%**	96.35%	97.75%
Fake Rate	47.31%	47.45%	**36.71%**	55.58 %	57.67%
Time (sec.)	2.17	2.64	1.07	**0.22**	0.07

SYMMETRY

INCLUDING SYMMETRIES IN ML

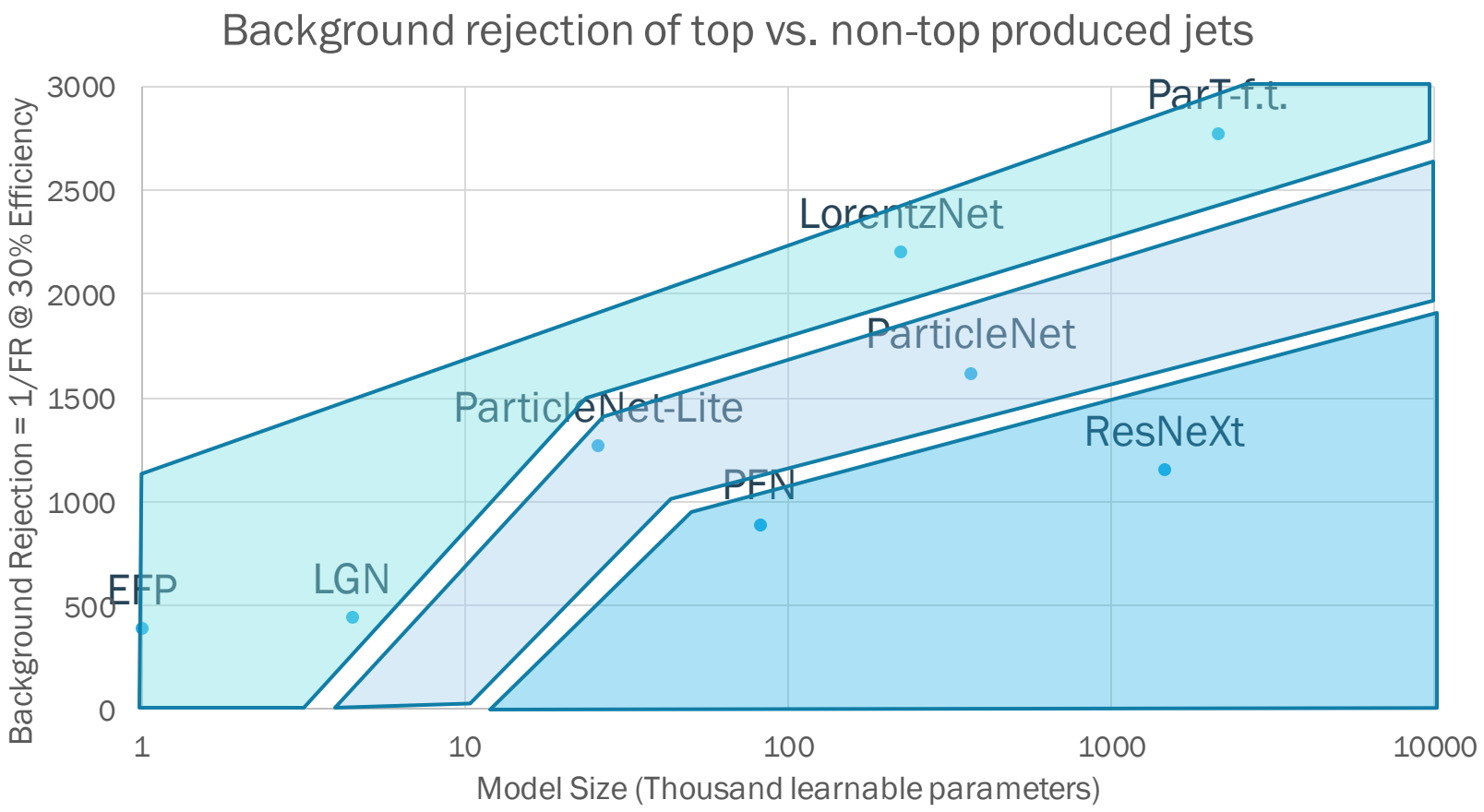
- The message passing in the GNN is "unconstrained" – any features can go in, and because of MLP Universal Approximator Theorem, *any* function may be learned that operates on input features
- However, we can use our physics intuition to reduce the search space of this learned function
- We know that there are symmetries in the geometry of some systems, which can be "built into" the GNN (and almost any other ML architecture)

WHY EQUI-GNN: NAIVELY IMPROVING MODEL PERFORMANCE



Would love to add LundNet-5 and JEDI-net to this plot, but don't have apples-to-apples rejection rate

WHY EQUI-GNN: NAIVELY IMPROVING MODEL PERFORMANCE



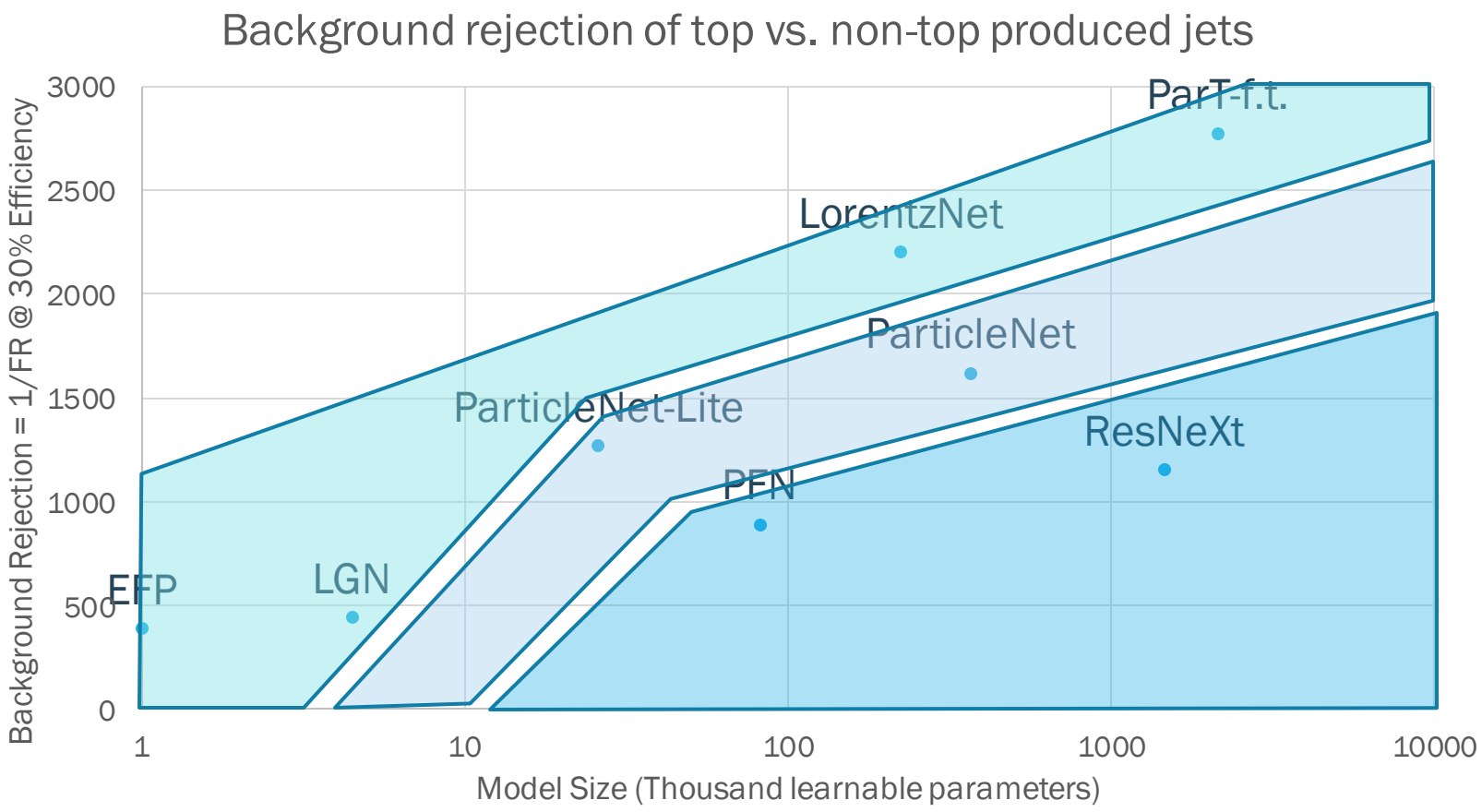
PHYSICS-MOTIVATED ML (SYMMETRY, DATA)

RELATIONAL ML (GRAPHS)

NON-RELATIONAL ML (SETS, IMAGES)

Would love to add LundNet-5 and JEDI-net to this plot, but don't have apples-to-apples rejection rate

WHY EQUI-GNN: NAIVELY IMPROVING MODEL PERFORMANCE



- Given a particular ML structure (a.k.a relational bias), diminishing returns on simply increasing model size
- Graph-structured appears to be as general as one can get structurally
- GNN-based models seem to perform best at large size
- Physics-based models seem to perform best at small size
- Motivates us to constrain graph-structured ML with physics knowledge

Would love to add LundNet-5 and JEDI-net to this plot, but don't have apples-to-apples rejection rate

KINDS OF PHYSICS KNOWLEDGE

- A variety of knowledge about the physics case can be included in the algorithm
- Quantum field theory: Feynman diagram structure (EFP)
- QCD: Decay processes in the Lund plane (LundNet)
- Permutation invariance of the jet constituents (PFN, ParticleNet)
- QCD + permutation invariance: Lund features with GNN (ParT: ParticleTransformer)
- 2D translation invariance in the calorimeter (ResNeXt)
- Special relativity: Frame-invariance under Lorentz transformations (LorentzNet, VecNet, Covariant ParT, …)

Good summary of theory-based tagging in Kasieczka, et al.

QFT Symmetries

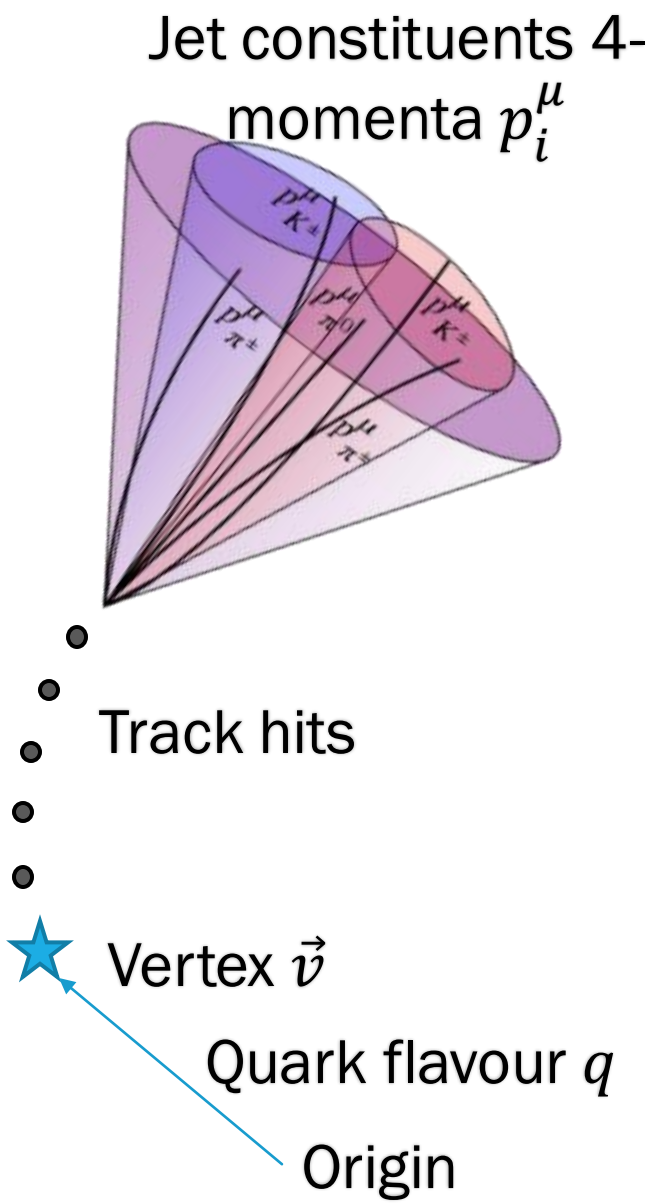
Spacetime Symmetries

Physics-informed Features

Data Augmentation

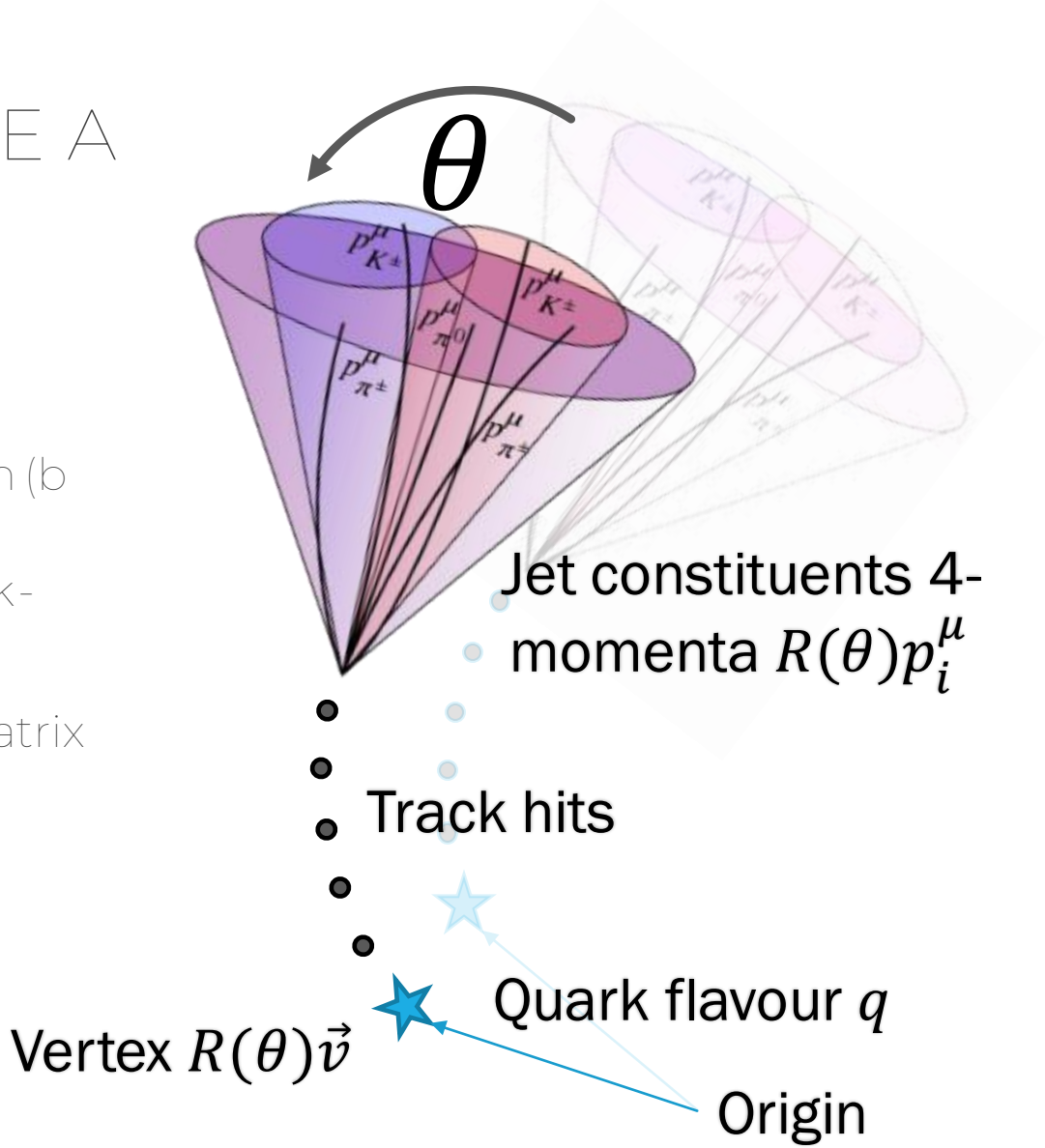
WHAT DOES IT MEAN TO INCLUDE A SYMMETRY?

- Consider jet flavor tagging
- Uses a GNN for predicting the source of jet production (b quark, c quark, tau jet or a lighter particle), as well as auxiliary predictions: track production vertex and track-pair vertex compatibility [ATL-PHYS-PUB-2022-027]
- Consider rotating the jet by angle ϕ, using rotation matrix $R(\theta)$



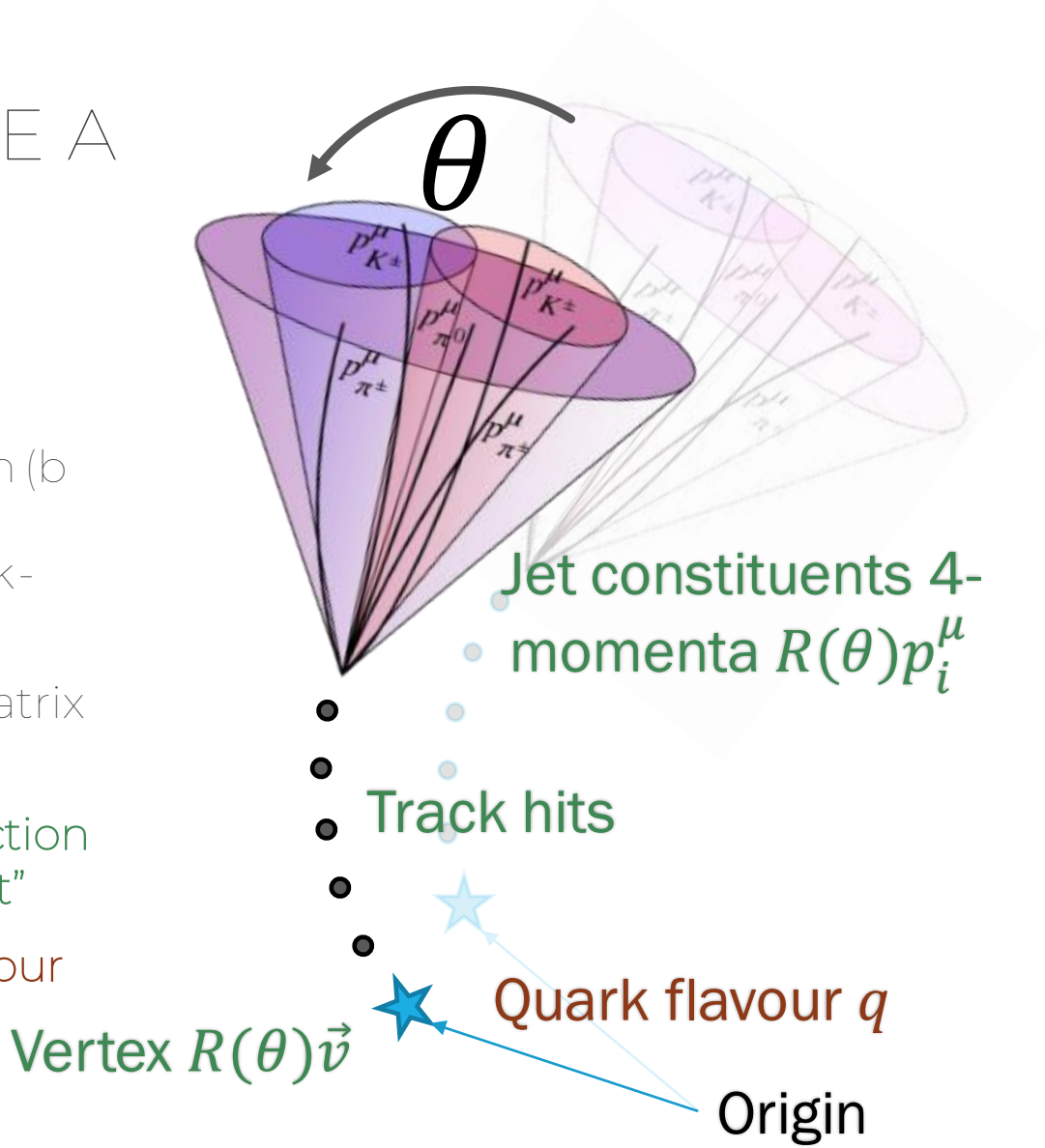
WHAT DOES IT MEAN TO INCLUDE A SYMMETRY?

- Consider jet flavor tagging
- Uses a GNN for predicting the source of jet production (b quark, c quark, tau jet or a lighter particle), as well as auxiliary predictions: track production vertex and track-pair vertex compatibility [ATL-PHYS-PUB-2022-027]
- Consider rotating the jet by angle ϕ, using rotation matrix $R(\theta)$



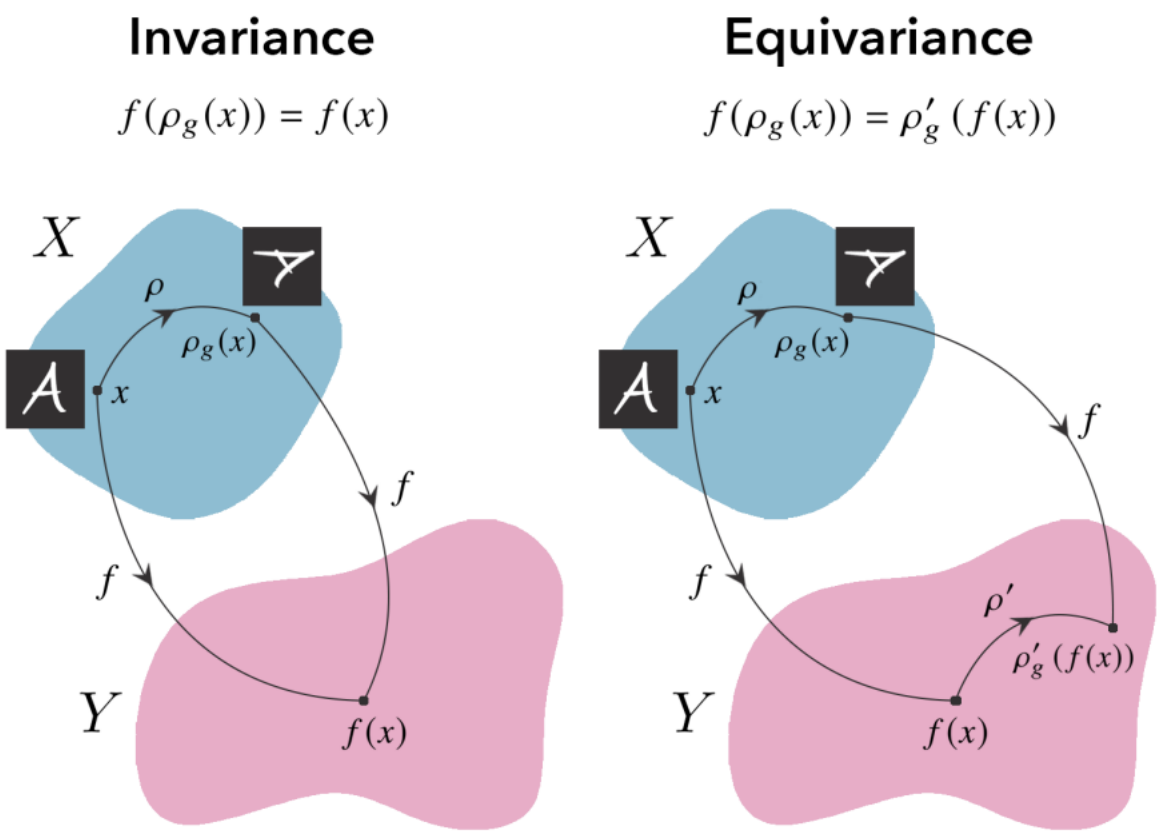
WHAT DOES IT MEAN TO INCLUDE A SYMMETRY?

- Consider jet flavor tagging
- Uses a GNN for predicting the source of jet production (b quark, c quark, tau jet or a lighter particle), as well as auxiliary predictions: track production vertex and track-pair vertex compatibility [ATL-PHYS-PUB-2022-027]
- Consider rotating the jet by angle ϕ, using rotation matrix $R(\theta)$
- Some predictions (and input features) like the production vertex will rotate with the transformation: "equivariant"
- Some predictions (and input features) like the jet flavour should not be affected: "invariant"



INVARIANCE VS. EQUIVARIANCE

- For some neural network f and some input feature x
- For a group element $g \in G$ transformation ρ_g
- Invariant network leaves output unaffected $f\left(\rho_g(x)\right) = f(x)$
- Equivariant (under G) network gives an output that is also transformed by $g \in G$
- May be same representation ρ_g or another representation ρ'_g

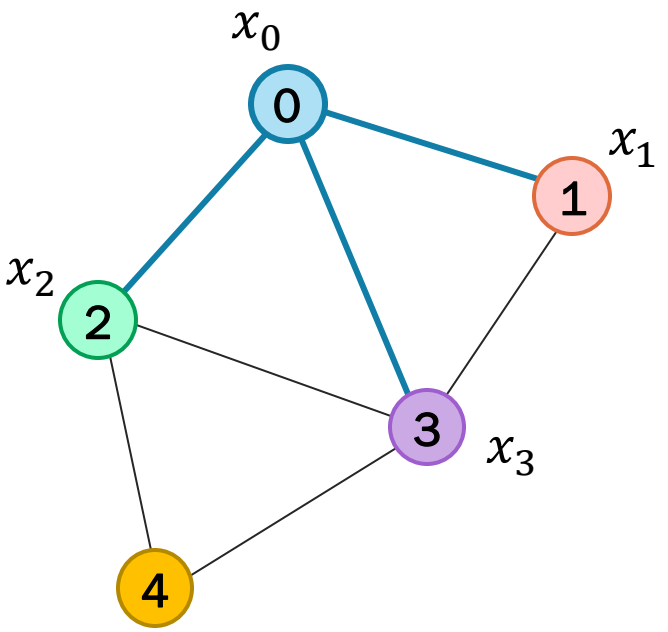


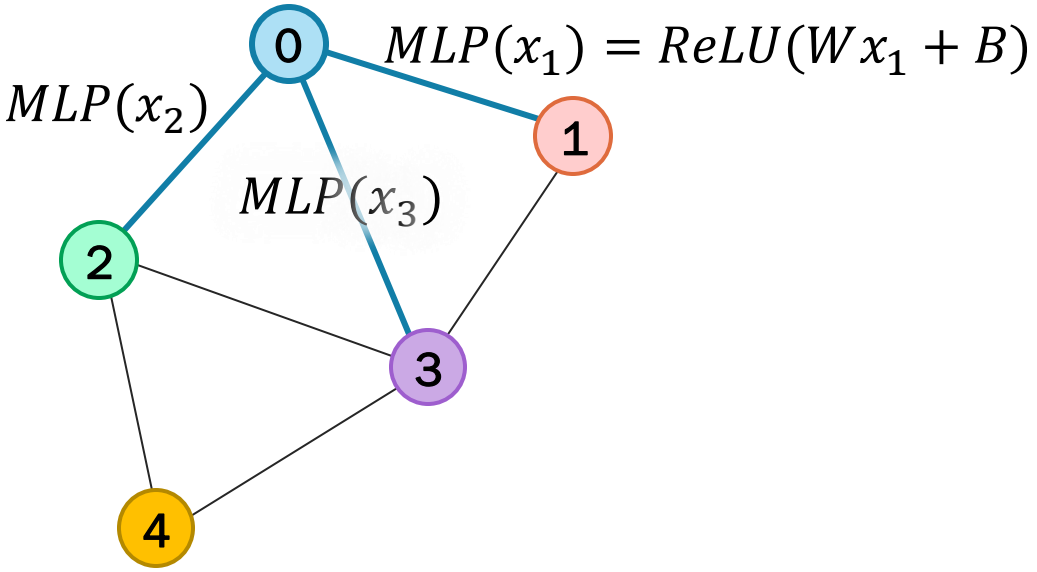
Lovely plot from Mariel Pettee: *Symmetry Group Equivariant Architectures for Physics* – Snowmass White Paper

WHAT DOES IT MEAN TO INCLUDE A SYMMETRY?

- Consider a point cloud, with behavior that you expect to be invariant under E3 symmetry – 3 dimensional Euclidean (rotational and translational) transformations
- Observe how a transformation R propagates in some arbitrary GNN convolution:

$$x_0' = \sum_i MLP(x_i)$$

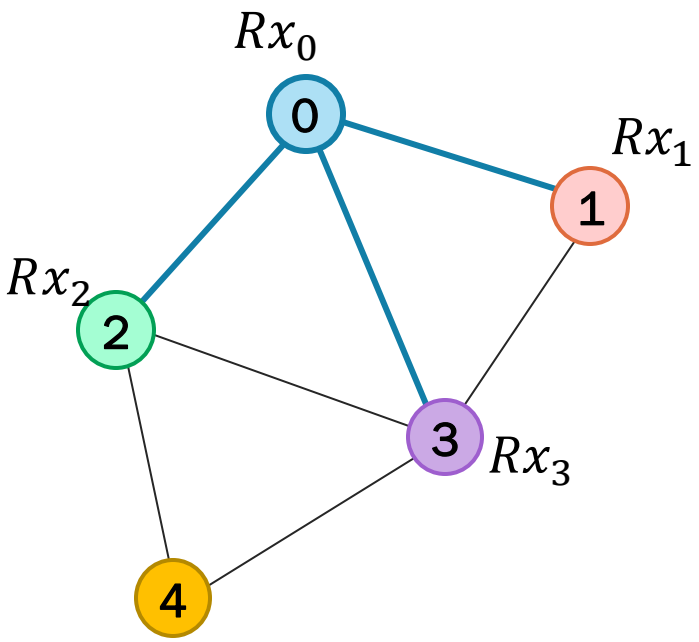


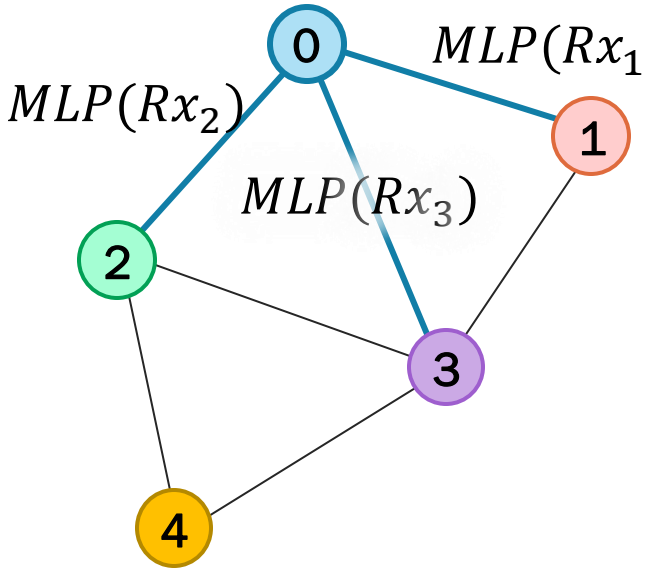


WHAT DOES IT MEAN TO INCLUDE A SYMMETRY?

- Consider a point cloud, with behavior that you expect to be invariant under E3 symmetry – 3 dimensional Euclidean (rotational and translational) transformations
- Observe how a transformation R propagates in some arbitrary GNN convolution:

$$x_0' = \sum_i MLP(Rx_i) = ??$$

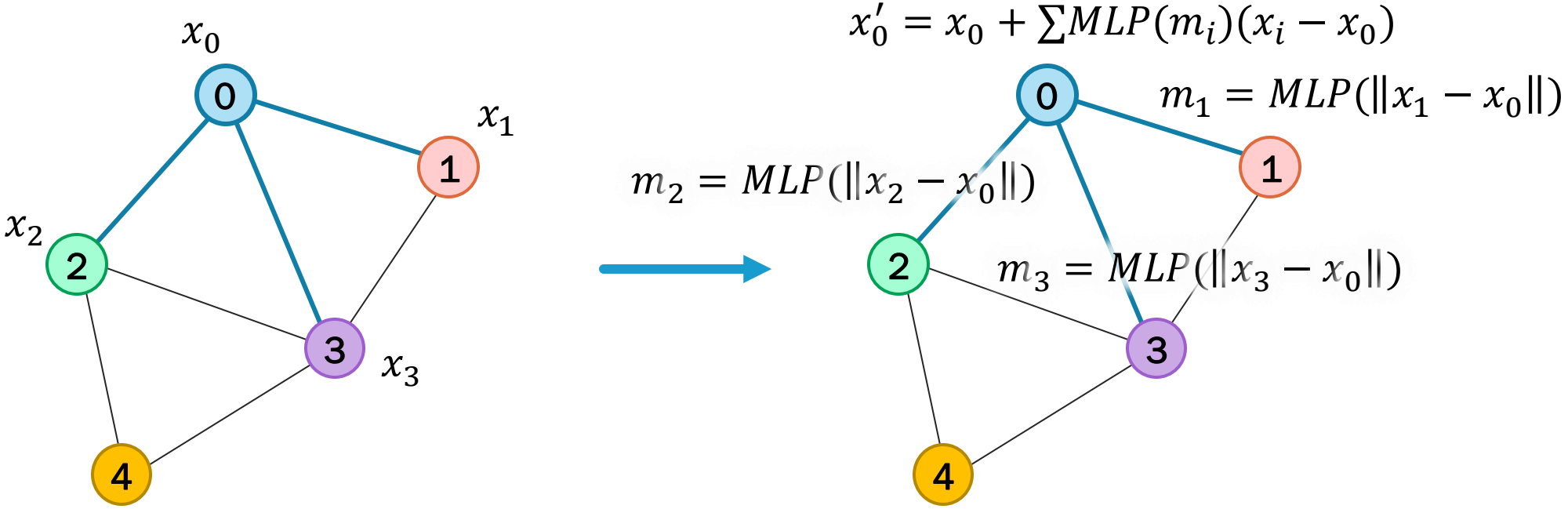




WHAT DOES IT MEAN TO INCLUDE A SYMMETRY?

- Consider a point cloud, with behavior that you expect to be invariant under E3 symmetry – 3 dimensional Euclidean (rotational and translational) transformations
- Observe how a transformation R propagates in some arbitrary GNN convolution
- To preserve E3 symmetry, we must choose a specific kind of message passing function:

$$x_0' = x_0 + \sum MLP(m_i)(x_i - x_0)$$

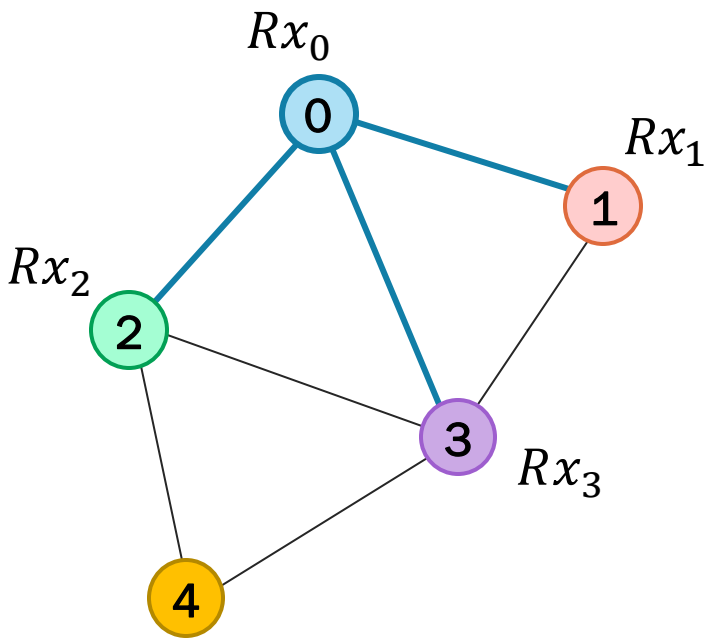


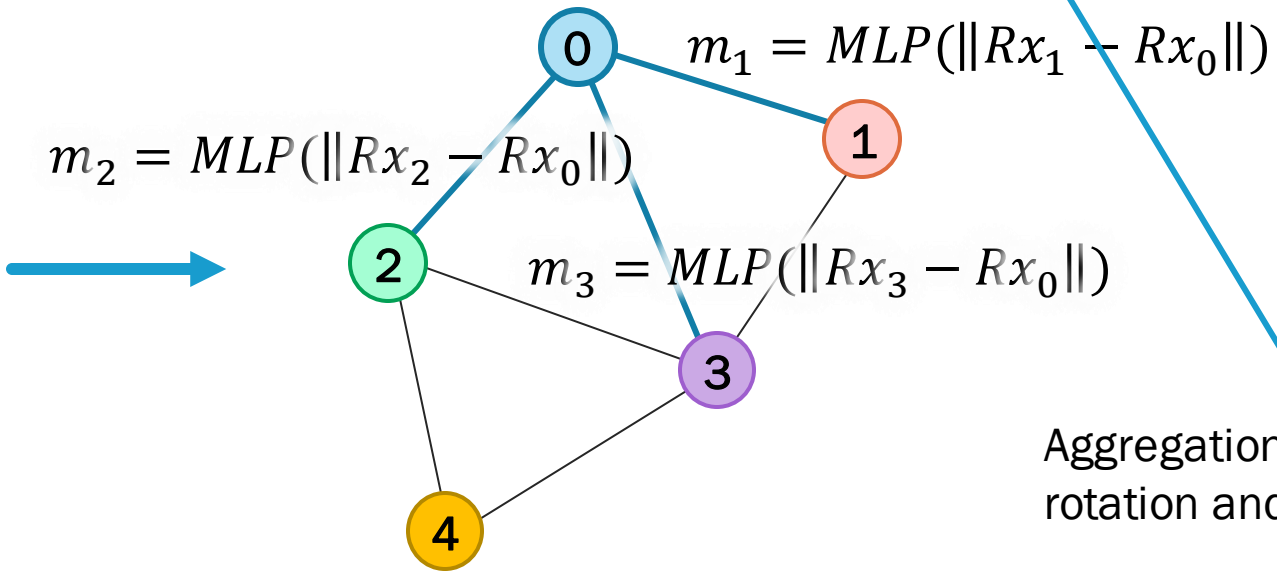
WHAT DOES IT MEAN TO INCLUDE A SYMMETRY?

$$\|Rx_3 - Rx_0\|^2 = (Rx_3 - Rx_0)^T(Rx_3 - Rx_0)$$
$$= (x_3 - x_0)^T R^T R(x_3 - x_0)$$
$$= \|x_3 - x_0\|^2$$

Message passing invariant to rotation and translation

$$Rx_0 + \sum MLP(m_i)(Rx_i - Rx_0) = Rx_0'$$



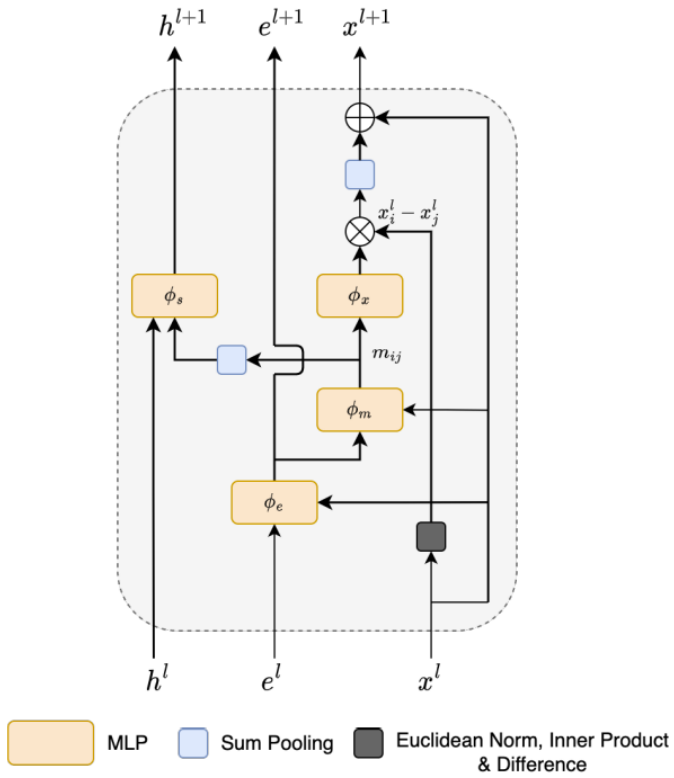


Aggregation equivariant to rotation and translation

E(n) Equivariant Graph Neural Networks, Satorras, et al

SO(2)-EQUIVARIANT GNN FOR TRACKING

- We expect collisions in the LHC to be rotationally symmetric around the beamline
- We also expect them to obey Lorentz symmetry for boosts along the beamline, but to capture this you need **four-vectors** (time is not available in TrackML or, reliably, ITk)
- We can constrain our tracking GNN to preserve SO(2) equivariance
- That is: nodes have three inputs, organized into equivariant features $[x, y]$ and invariant features $[z, charge, \ldots]$; then all intermediate node hidden features are also either equivariant or invariant
- Output edge classification is then *invariant* to rotations around $x - y$

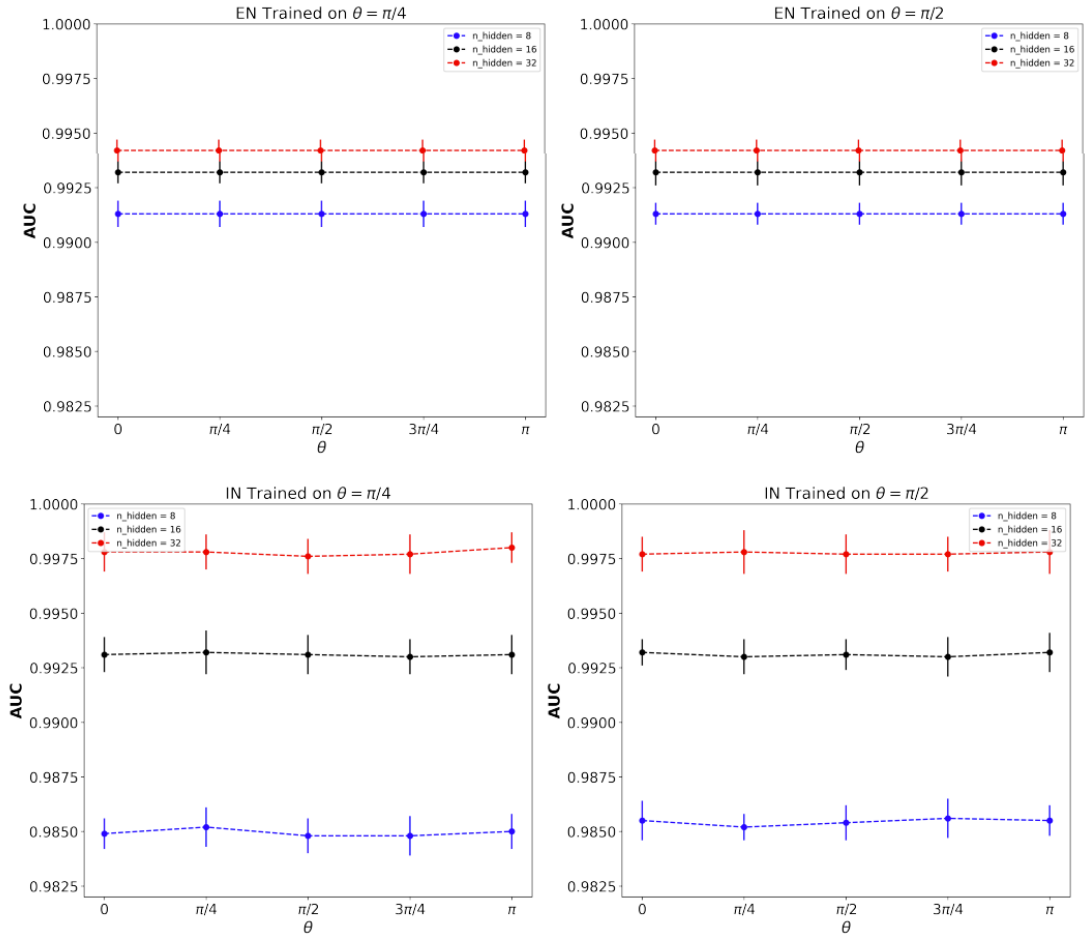


Euclidean Equivariant Block

SO(2)-EQUIVARIANT GNN FOR TRACKING

- This works, *to a degree*
- Get good performance for very small models
- At some point, an unconstrained model outperforms
- Interestingly, even small unconstrained models learn the symmetry

N_{hidden}	Model	Params	AUC	Efficiency	Purity
8	EuclidNet	967	$\mathbf{0.9913 \pm 0.004}$	$\mathbf{0.9459 \pm 0.022}$	$\mathbf{0.7955 \pm 0.040}$
	InteractionNet	1432	0.9849 ± 0.006	0.9314 ± 0.021	0.7319 ± 0.052
16	EuclidNet	2580	0.9932 ± 0.003	0.9530 ± 0.014	$\mathbf{0.8194 \pm 0.033}$
	InteractionNet	4392	0.9932 ± 0.004	$\mathbf{0.9575 \pm 0.019}$	0.8168 ± 0.073
32	EuclidNet	4448	0.9941 ± 0.003	0.9547 ± 0.019	0.9264 ± 0.023
	InteractionNet	6448	$\mathbf{0.9978 \pm 0.003}$	$\mathbf{0.9785 \pm 0.022}$	$\mathbf{0.9945 \pm 0.043}$

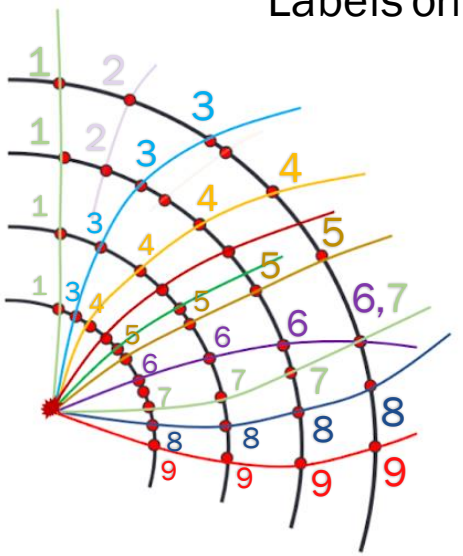


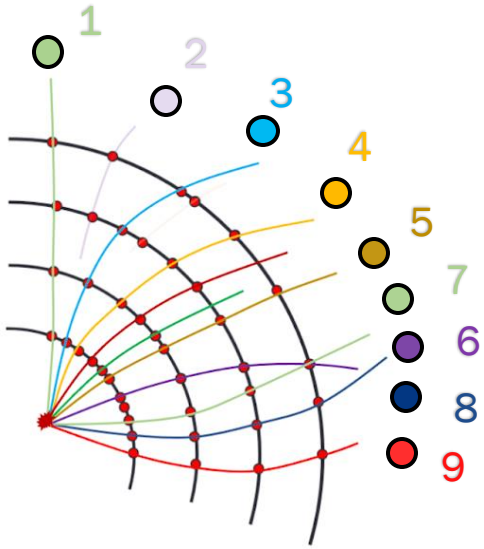
TRACKING AS OBJECT DETECTION

THE TRACKING PROBLEM

- Protons collide in center of detector, "shattering" into thousands of particles
- The *charged* particles travel in curved **tracks** through detector's magnetic field (Lorentz force)
- A track is defined by the **hits** left as energy deposits in the detector material, when the particle interacts with material
- In this study, we use the TrackML Dataset [link], with variable-sized subsets of tracks selected
- The goal of track reconstruction: Given set of hits from particles in a detector, assign label(s) to each hit.

Labels on hits





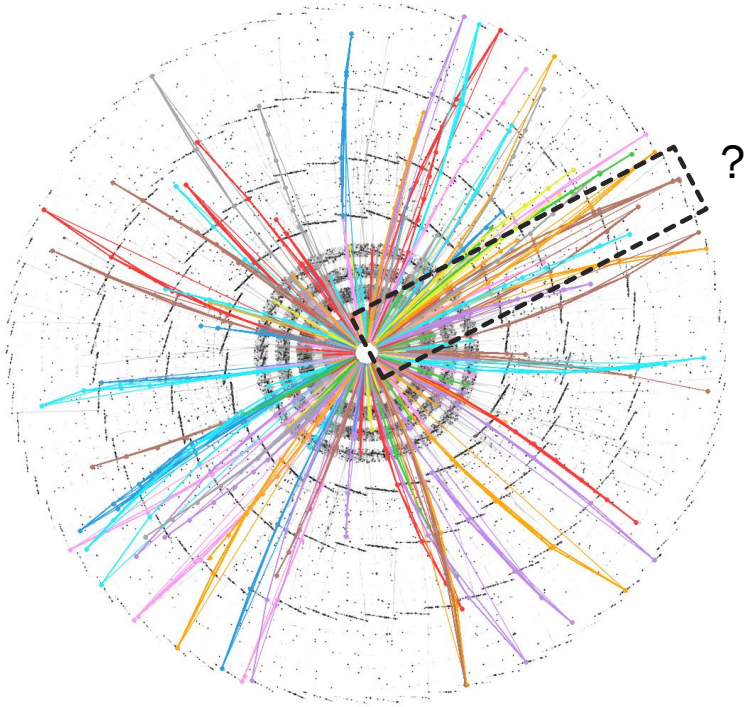
Hits to labels

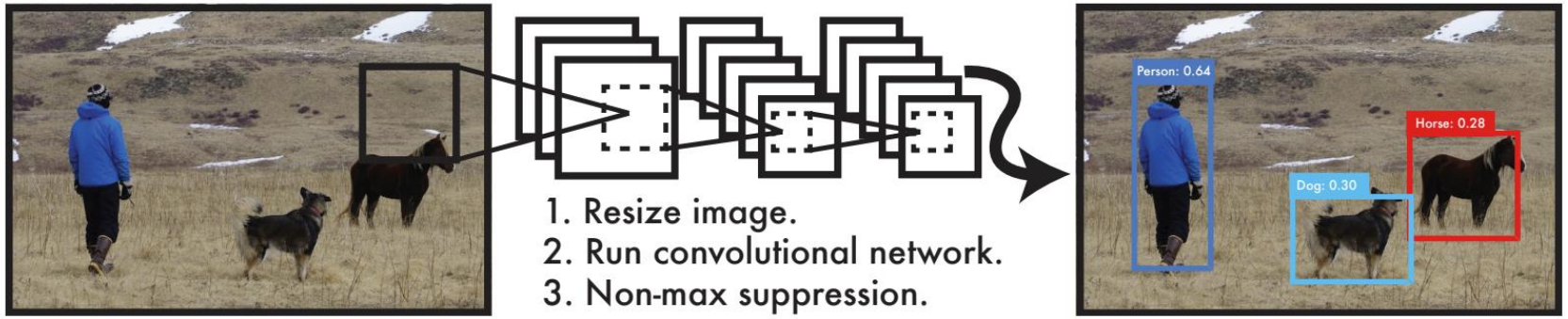
Can reframe the problem of assigning ***label → hits***

1. Assume the existence of some uniquely labelled "*representative point*" in each track object
2. Then our task is to assign ***hits → representative point***

TRACKING AS OBJECT DETECTION

- A well-studied problem in computer vision: Given an image, can we identify all discrete objects of interest and predict information about them?
- Popular approach is to draw a bounding box as the representative label
- Can't directly use this approach for tracking: tracks are not localized in 3D space





The "You Only Look Once" (YOLO) approach to detection: draw a bounding box and predict the object in a single step.

Redmond et al, arXiv: 1506.02640

OBJECT DETECTION AS METRIC LEARNING

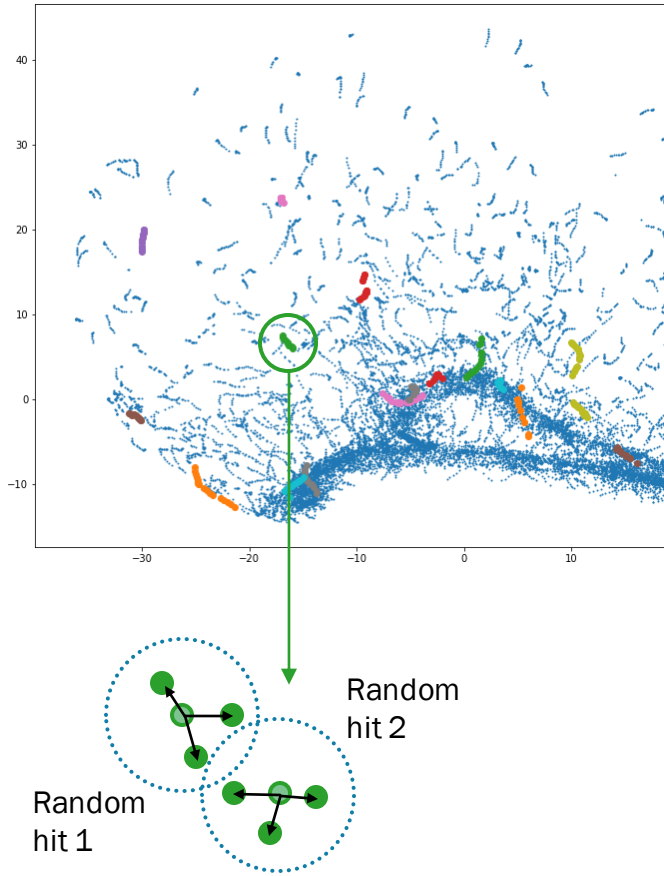
- We consider a "naïve" solution to the object detection problem
- Using a transformer or graph neural network (GNN), embed each hit x_i in a latent space $\mathcal{U}(\mathrm{x_i})$
- Use a hinge loss to encourage hits from the same particle ($y_{ij} = 1$) to be close, hits from different particles ($y_{ij} = 0$) to be distant:

$$L = \begin{cases} \Delta_{\mathrm{ij}}, & when\ \ y_{ij} = 1 \\ \max(0, 1 - \Delta_{\mathrm{ij}}), & when\ \ y_{ij} = 0 \end{cases}$$

To create *representative points*, we use a "greedy condensation" approach. For all points:

1. Randomly select a point
2. Find all neighbors (within radius R)
3. If none of the neighbors are already a representative, then convert the point to a representative, and attach all neighbors to that representative

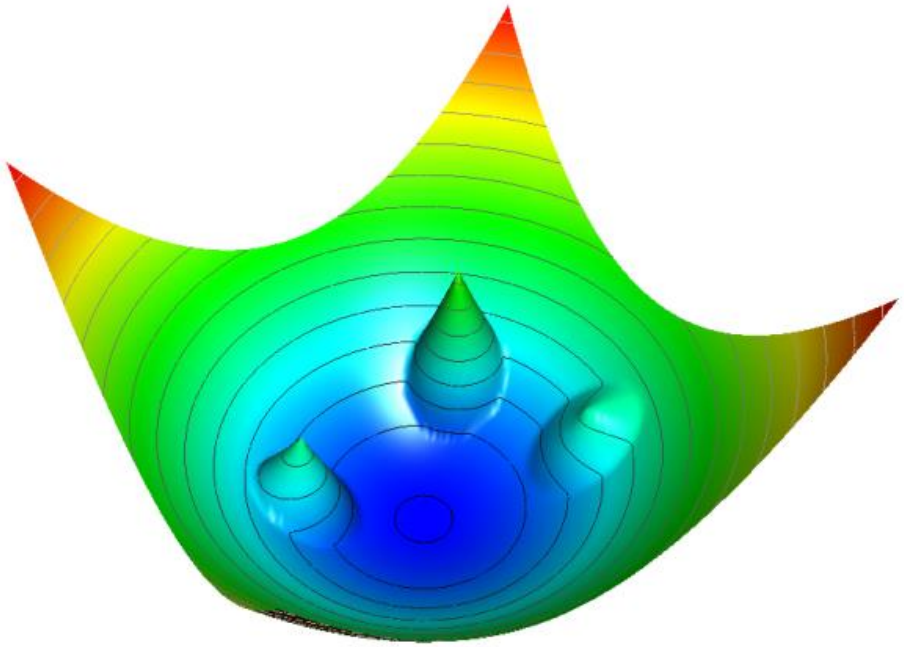
Latent $\mathbb{R}^{12}$ projected to $\mathbb{R}^2$



Works quite well, but some points are clearly better candidates for representative than others. Can we learn which points are good representative points?

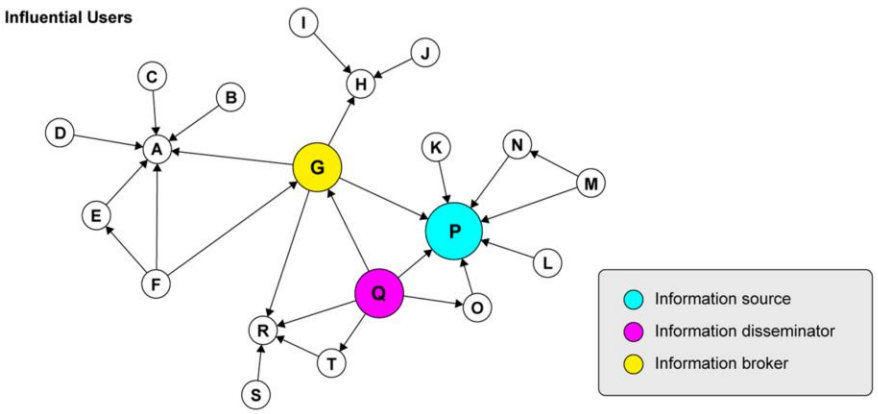
OBJECT CONDENSATION: LEARNING REPRESENTATIVE POINTS

- Idea from particle flow reconstruction: *Object condensation: one-stage grid-free multi-object reconstruction in physics detectors, graph, and image data*, Kiesler 2020 [link]
- Simultaneously learn an embedding similarity space and a condensation score for each hit, where a higher score is a more "attractive" point charge in similarity space
- All hits with learned condensation score β above some threshold are considered candidates for representation points, then we apply greedy condensation to the representatives sorted by β
- Shortcomings:
  - Having this "hard cut" charge threshold requires fine-tuning
  - Inference requires sorting likely condensation points and sequentially considering each condensation point based on all previous condensation points
  - Training (as a simplification) only considers *maximum-scoring* condensation point in each class, which neglects global optima



The potential function of members of the same class relative to the representation point of that class (*Kiesler 2020*)

DESIRED LOSS FUNCTION BEHAVIOUR: A TWITTER INSPIRATION



Kim & Valente 2020, COVID-19 Health Communication Networks on Twitter: Identifying Sources, Disseminators, and Brokers

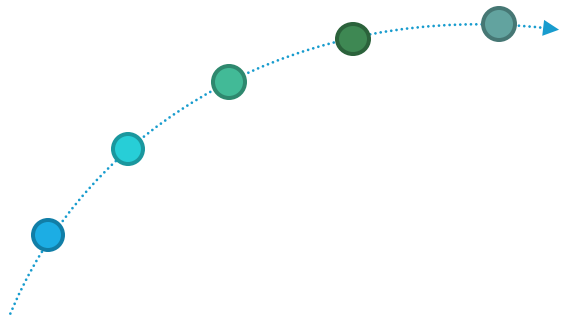
- Idea: We can represent a social network as a directed graph of influence flow
- *Recuero et al, 2019*, and *Kim & Valente 2020* used network analysis to identify several types of user based on in-degree and out-degree of information flow
- Let's simplify: All members of network can be users (receive information from incoming edge) *and* influencers (send information to outgoing edge)
- We can build a directed graph by learning for each member of the point cloud two embeddings in the *same space*: a user-embedding and an influencer-embedding

Goal 1
We would like users of each class to crowd around exactly one influencer that represents their class

Goal 2
We want influencers to be distant from each other

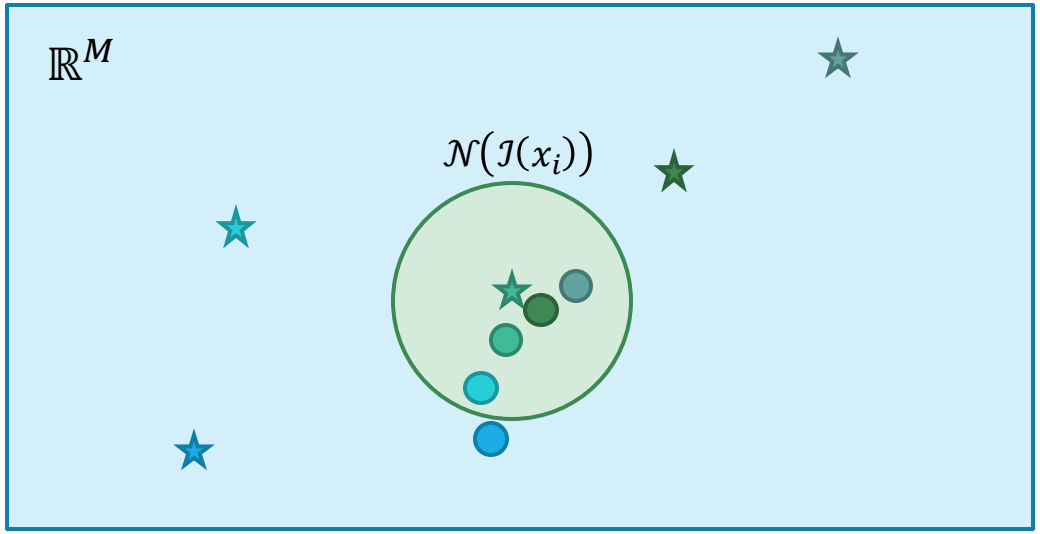
DESIRED LOSS FUNCTION BEHAVIOUR

- Given each of N points x_i in track T_a embedded into $\mathbb{R}^M$ with two models: a user-embedding $\mathcal{U}$ and an influencer-embedding $\mathcal{I}$
- We want a minimum in the loss when *all* hits $x_i \in T_a$ have $\mathcal{U}(x_i)$ inside neighbourhood $\mathcal{N}(\mathcal{I}(x_i))$ for at least one influencer (and preferably *only* one influencer)



T_a, $N = 5$

$\mathcal{U}(x_i), \mathcal{I}(x_i)$



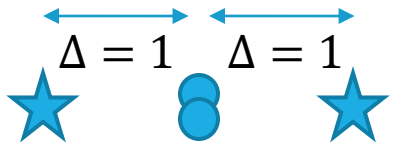
In this case, 4 out of 5 users are in the neighbourhood of an influencer

- Position of **user-embeddings**
- Position of **influencer-embeddings**

DESIRED LOSS FUNCTION BEHAVIOUR

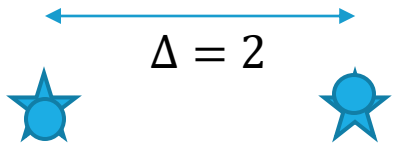
- Given each of N points x_i in track T_a embedded into $\mathbb{R}^M$ with two models: a user-embedding $\mathcal{U}$ and an influencer-embedding $\mathcal{I}$
- We want a minimum in the loss when *all* hits $x_i \in T_a$ have $\mathcal{U}(x_i)$ inside neighbourhood $\mathcal{N}(\mathcal{I}(x_i))$ for at least one influencer (and preferably *only* one influencer)
- We can achieve this by taking $L_u(T_a) = \sqrt[N]{\prod_j \frac{1}{N} \sum_i \left|\mathcal{U}(x_i) - \mathcal{I}(x_j)\right|^2}$
- Consider loss L in simple example of two points in three different cases:

● Position of **user-embeddings**
★ Position of **influencer-embeddings**



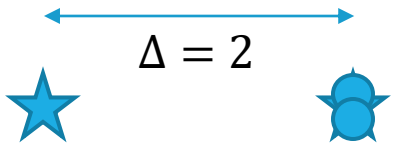
$L \propto (\Delta_{00}^2 + \Delta_{01}^2) \times (\Delta_{10}^2 + \Delta_{11}^2) = 4$

Case A



$L \propto (\Delta_{00}^2 + \Delta_{01}^2) \times (\Delta_{10}^2 + \Delta_{11}^2) = 16$

Case B



$L \propto (\Delta_{00}^2 + \Delta_{01}^2) \times (\Delta_{10}^2 + \Delta_{11}^2) = 0$

Case C

Note: Noise is given a class label NaN and handled like all other data points

THE INFLUENCER LOSS

- The attractive Influencer-User loss is actually the geometric mean across influencers of the arithmetic mean across users of the distance between each positive pair across all n tracks, so we can rewrite it for numerical stability:

$$L_{\mathcal{U}}^{+}(T_a) = \exp\left(\frac{1}{N}\sum_j \ln\left(\frac{1}{N}\sum_i \Delta_{ij}^2\right)\right), \qquad L_{\mathcal{U}}^{+} = \frac{1}{n}\sum_a T_a, \qquad y_{ij} = 1$$

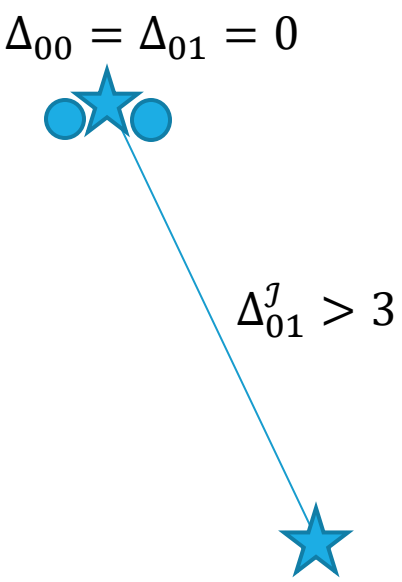
- We include a repulsive Influencer-User hinge loss to punish users condensing towards an influencer from a different class:

$$L_{\mathcal{U}}^{-} = \mathrm{mean}_{ij}\left(\max\left(0, 1 - \Delta_{ij}\right)\right), \qquad y_{ij} = 0$$

- And finally, we encourage influencers being a distance of at least $\boldsymbol{\Delta}^{\mathcal{I}}$ from each other, to avoid users being "overrepresented" by multiple influencers:

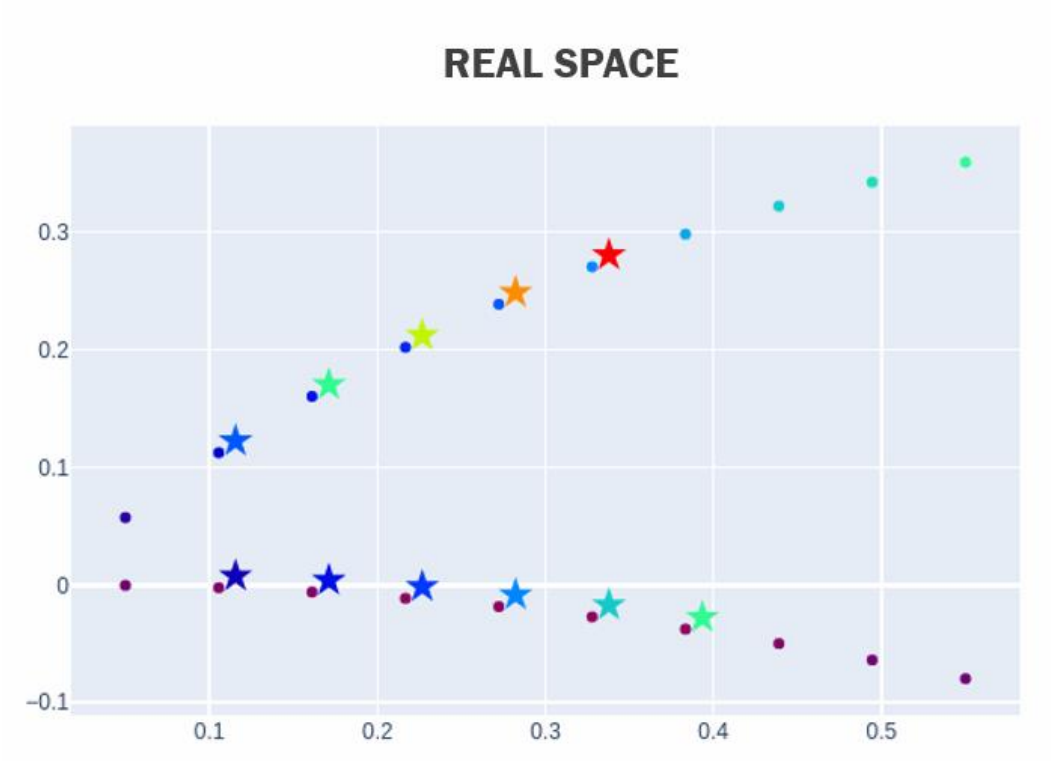
$$L_{\mathcal{I}} = \mathrm{mean}_{ij}\left(\max\left(0, \boldsymbol{\Delta}^{\mathcal{I}} - \Delta_{ij}^{\mathcal{I}}\right)\right), \qquad y_{ij} = 0$$

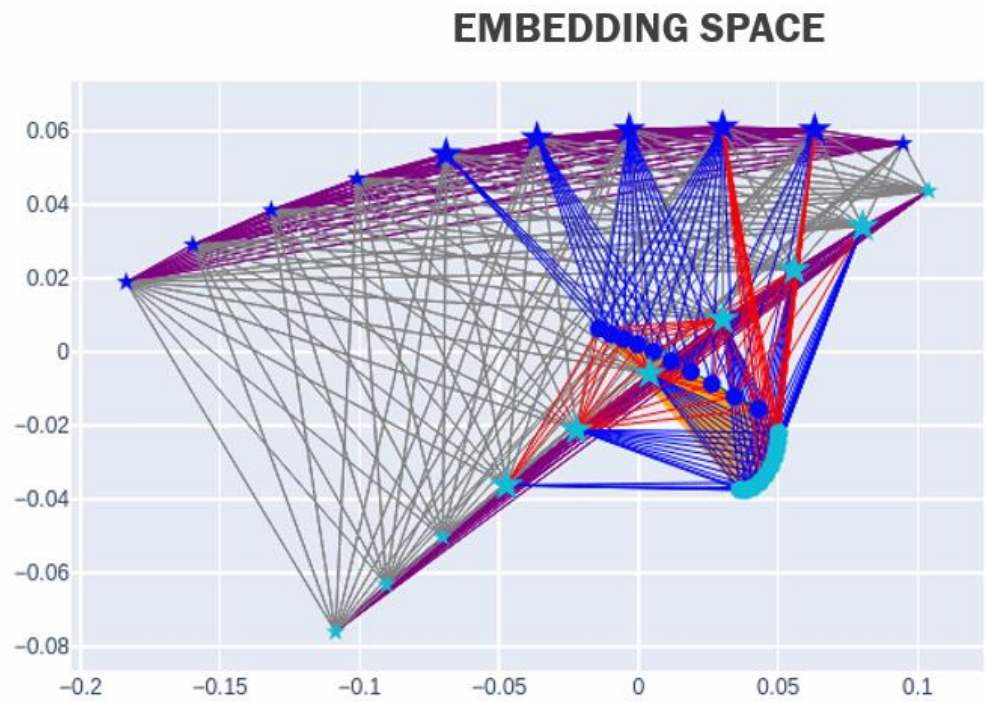
- We take this combination as the Influencer Loss $\boldsymbol{L} = \boldsymbol{L}_{\mathcal{U}}^{+} + \boldsymbol{a}\boldsymbol{L}_{\mathcal{U}}^{-} + \boldsymbol{b}\boldsymbol{L}_{\mathcal{I}}$, where the weights $\boldsymbol{a}$ and $\boldsymbol{b}$ can be used to tune the efficiency-purity rate and the efficiency-duplicate rate, respectively



The total Influencer Loss is at a minimum in this case

A TRAINING MONTAGE

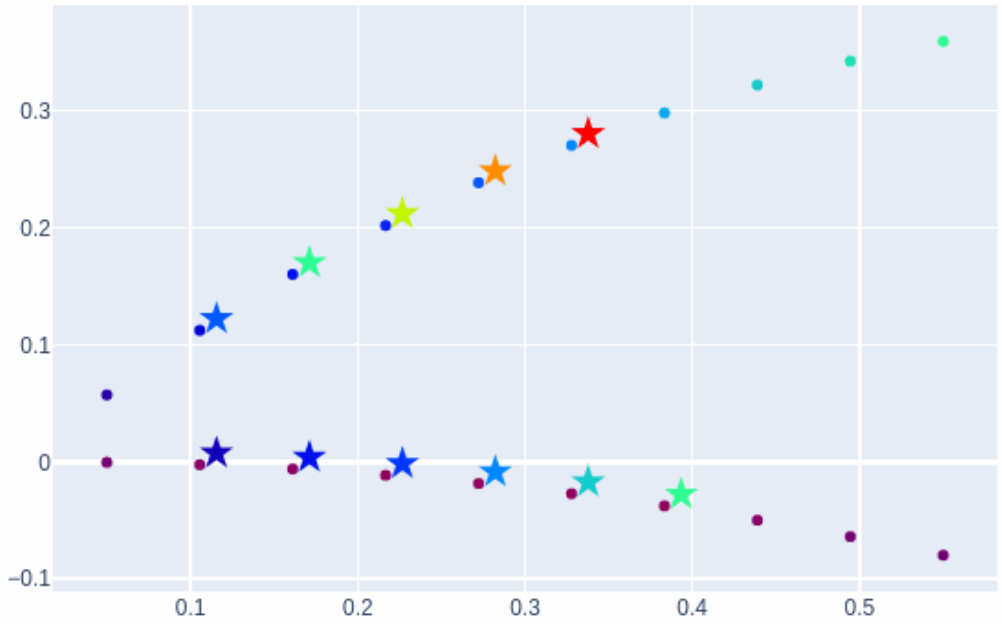




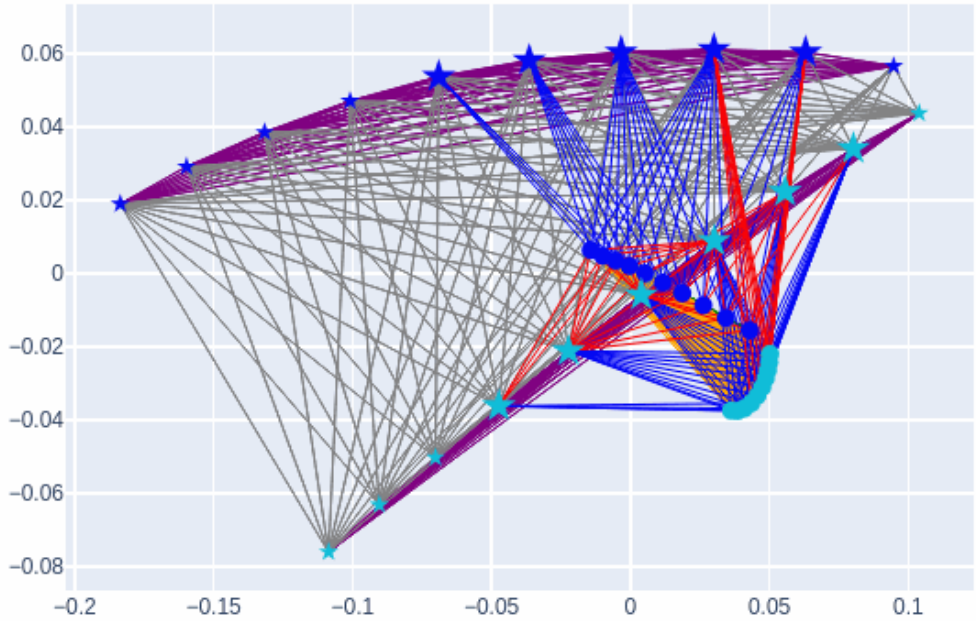
- We can see the Influencer Loss working on two tracks above, across training epochs
- In **Real Space**, we show only Users (circles) and Influencers (stars) when they are associated with an Influencer or User (respectively)
- The color in **Real Space** is a projection in 1D of the location in **Embedding Space**
- In **Embedding Space**, we should edges created, and connected Influencers are large stars, unconnected Influencers are small stars

A TRAINING MONTAGE

REAL SPACE



EMBEDDING SPACE

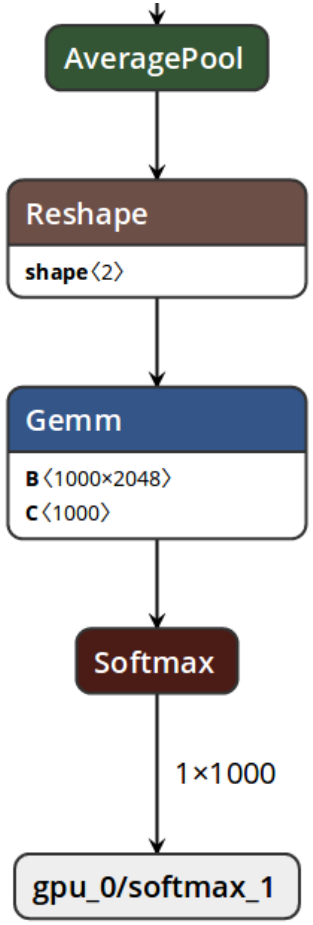


- We can see the Influencer Loss working on two tracks above, across training epochs
- In **Real Space**, we show only Users (circles) and Influencers (stars) when they are associated with an Influencer or User (respectively)
- The color in **Real Space** is a projection in 1D of the location in **Embedding Space**
- In **Embedding Space**, we should edges created, and connected Influencers are large stars, unconnected Influencers are small stars

GNN TRACKING IN PRODUCTION

CONVERSION TO ONNX

- Onnx is the gold standard for portability of ML
- Takes any framework (Tensorflow, Pytorch, Jax)
- Represents as a computational graph
- However, graph neural network operations have always been lacking in Onnx
- The latest version of Pytorch operations and Onnx libraries *supports GNN conversion!*



CONVERSION TO C++

- This can be done with Onnx, with C++ library of OnnxRuntime
- Works basically out-of-the-box!
- Can also do this with LibTorch

```
// ************
// GNN
// ************
std::vector<const char*> gInputNames{"g_nodes", "g_edges"};
std::vector<Ort::Value> gInputTensor;
gInputTensor.push_back(
    std::move(fInputTensor[0])
);
std::vector<int64_t> gEdgeShape{2, numEdgesAfterF};
gInputTensor.push_back(
    Ort::Value::CreateTensor<int64_t>(
        memoryInfo, edgesAfterFiltering.data(), edgesAfterFiltering.size(),
        gEdgeShape.data(), gEdgeShape.size()
);
// gnn outputs
std::vector<const char*> gOutputNames{"gnn_edge_score"};
std::vector<float> gOutputData(numEdgesAfterF);
std::vector<int64_t> gOutputShape{numEdgesAfterF};
std::vector<Ort::Value> gOutputTensor;
gOutputTensor.push_back(
    Ort::Value::CreateTensor<float>(
        memoryInfo, gOutputData.data(), gOutputData.size(),
        gOutputShape.data(), gOutputShape.size()
);
runSessionWithIoBinding(*g_sess, gInputNames, gInputTensor, gOutputNames, gOutputTensor);

torch::Tensor gOutputCTen = torch::tensor(gOutputData, {torch::kFloat32});
gOutputCTen = gOutputCTen.sigmoid();
// std::cout << gOutputCTen.slice(0, 0, 3) << std::endl;
```

OPEN PROBLEMS

OPEN PROBLEMS

- Extending TrackML inference timing and scaling studies to ATLAS ITk
- Investigating training and inference performance on lower p_T tracks (i.e. < 1 GeV) and high p_T tracks (i.e. > 10 GeV)
- Investigating performance on large radius tracks and dense track environments
- Direct comparison with combinatorial Kalman filter (current algorithm) efficiency and track parameter resolution

OPEN PROBLEMS

- We have typically been afraid of "dense" representations, hence the building of more and more sparse graphs. But sparse representations may miss interesting relationships, and models like GOAT try to apply dense models to graphs
- We cannot yet get GNNs onto FPGAs easily – indexing and scattering is non-trivial
- Training GNNs across GPUs is non-trivial:
  - Event-level parallelism gives no benefit (typical LHC hit-graph above OpenAI model of noise scale)
  - Node-level parallelism is difficult to implement (but we are working with Georgia Tech group to do this, library called Lasagne has been successfully tested with parallelised graph attention network)
- Currently use spacepoints as the lowest level object, but ATLAS tracking natively uses clusters – should enable this in GNN pipeline