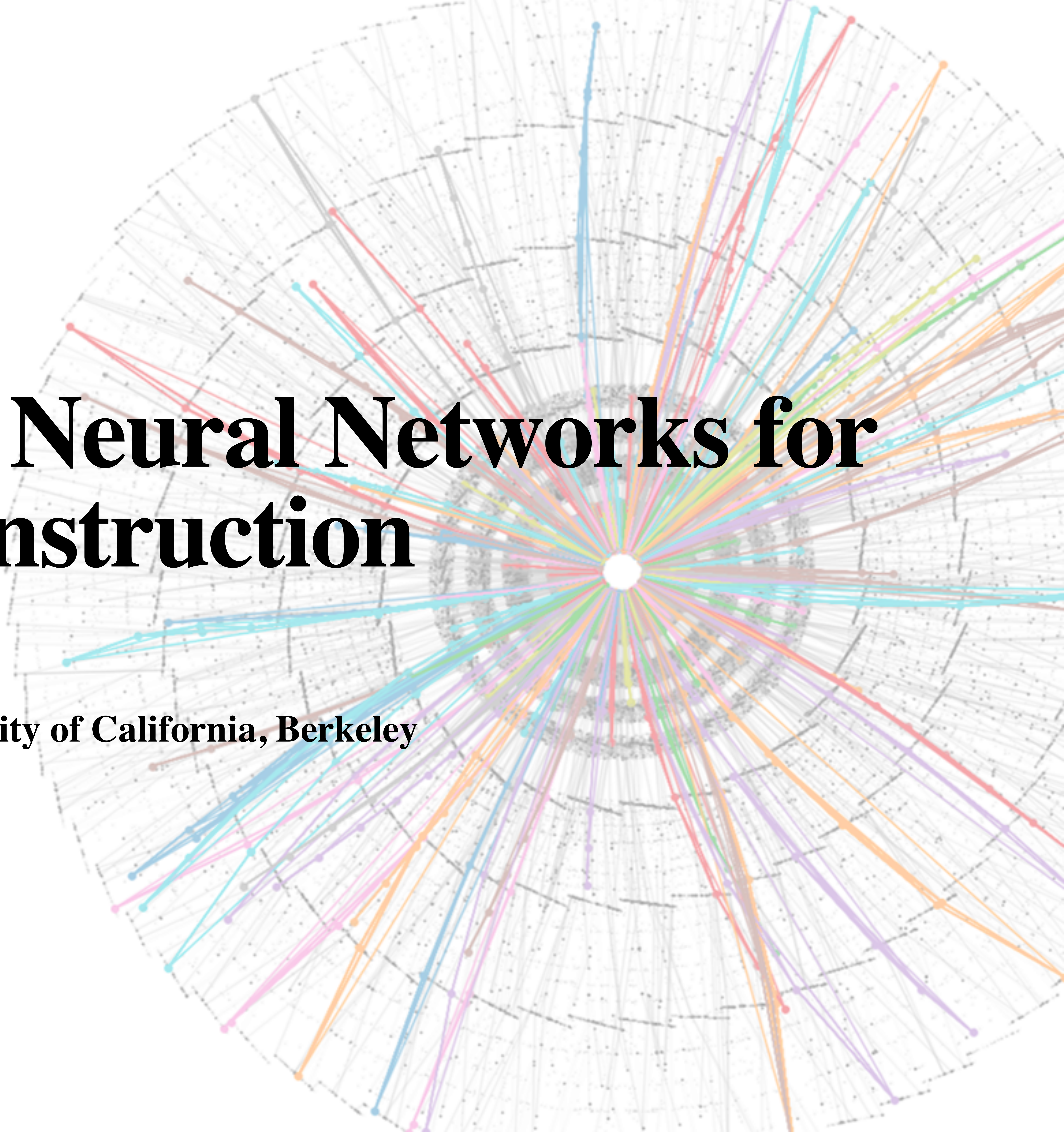
Hierarchical Graph Neural Networks for Particle Track Reconstruction



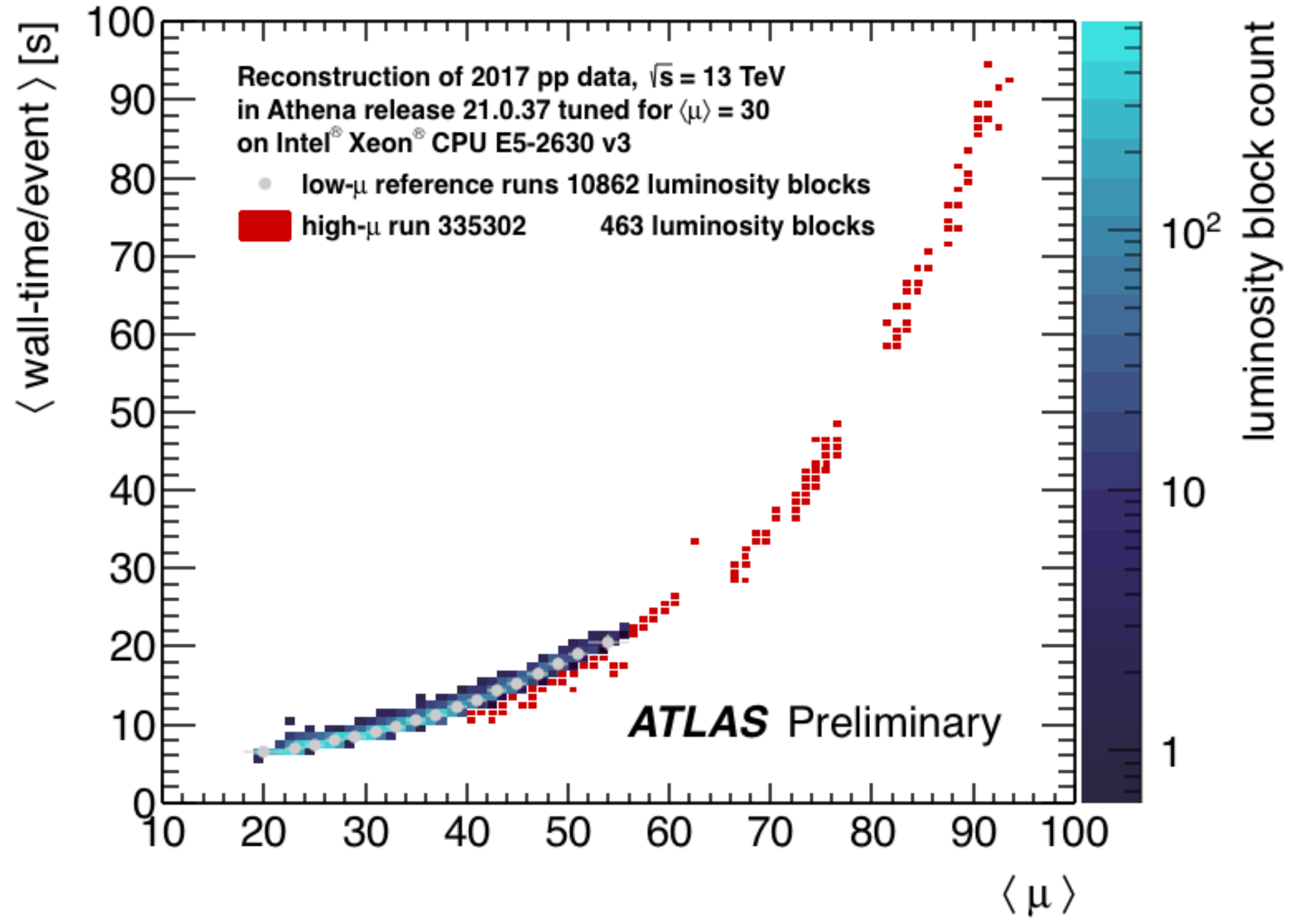
Ryan Liu
Lawrence Berkeley National Laboratory, University of California, Berkeley
Contact: liuryan30@berkeley.edu

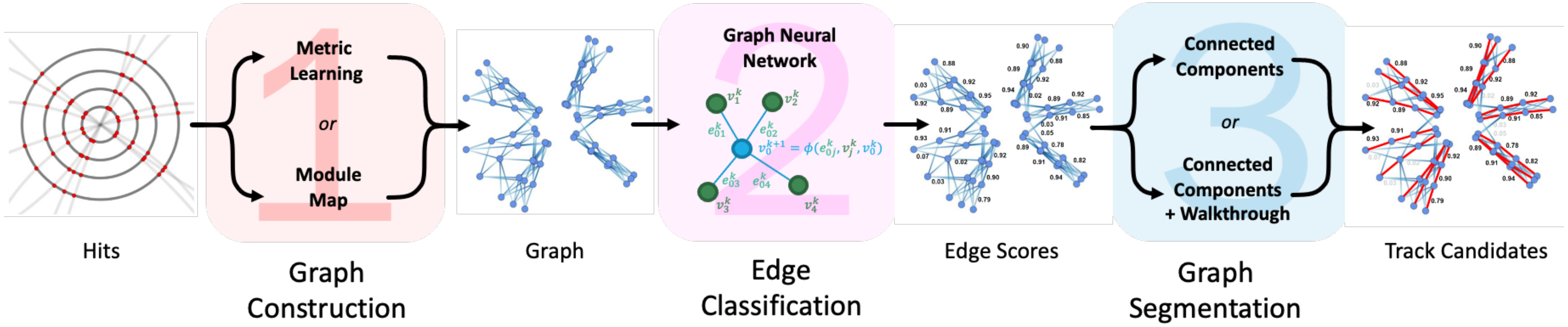
BERKELEY LAB

TrkX

The Exa.TrkX Track Reconstruction Pipeline

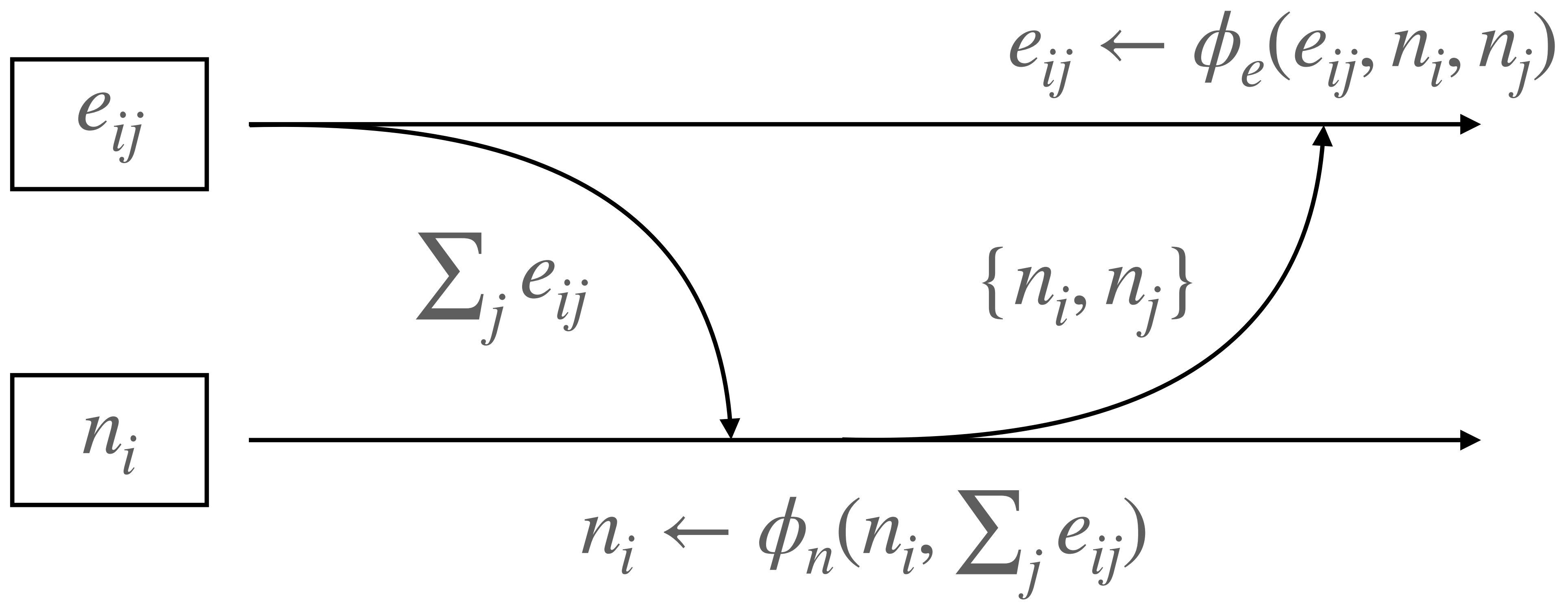
- The upcoming HL-LHC upgrade severely challenges computational resources and algorithm efficiency.
- Current tracking algorithms approximately scale **quadratically** with respect to the size of point cloud.
- The Exa.TrkX pipeline utilizes GNN methods to "connect the dots" in **less than quadratic** time.

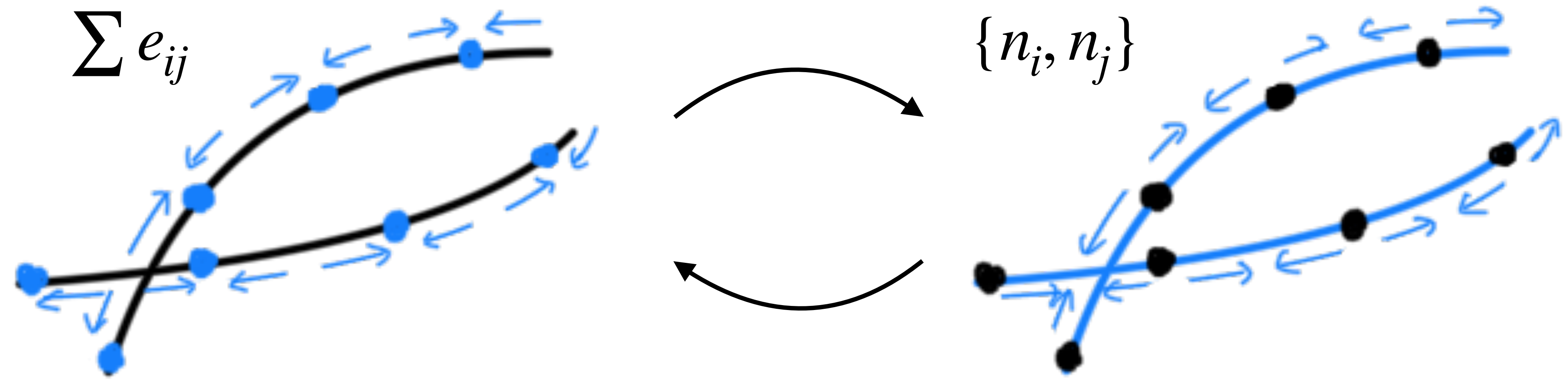




Graph Neural Networks

- **Graph Neural Network** (GNN) is a special type of neural network which takes a set of **nodes** ($X \in \mathbb{R}^{n \times d}$) and a **graph** (adjacency matrix $A \in \mathbb{R}^{n \times n}$) as input.
- **Permutation invariance**: for any permutation $P \in \mathbb{R}^{n \times n}$, inputting PX and PAP^{-1} yields the same output as not permuted inputs.
- There are many ways of realizing a GNN, such as graph convolutional network (GCN), graph attention network (GAT), and interaction network (IN).

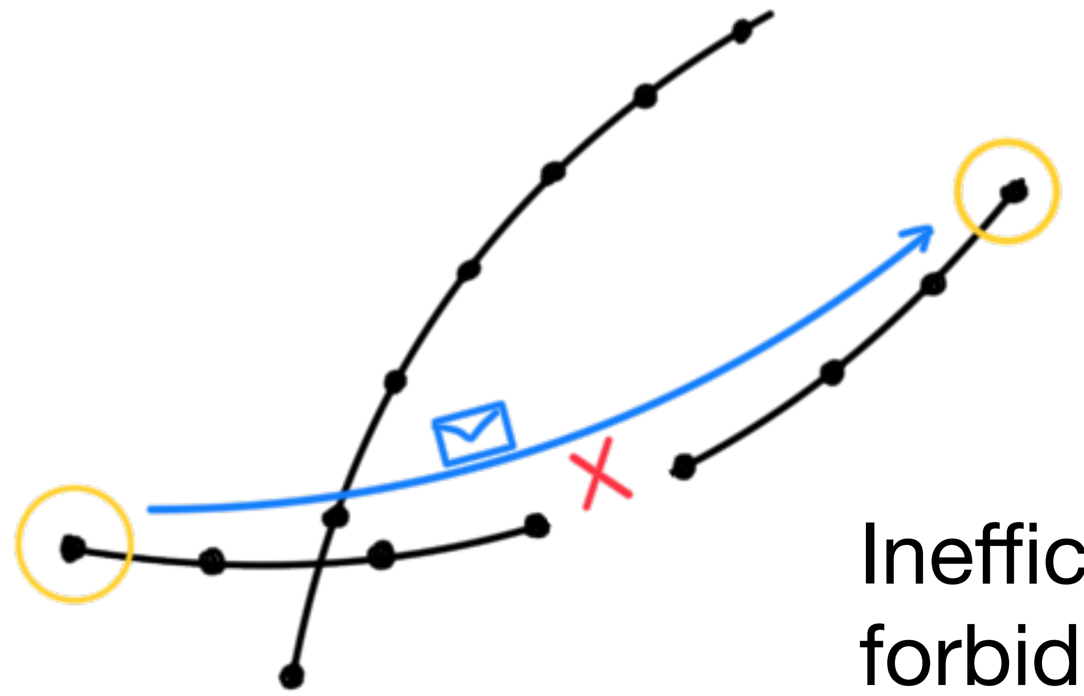
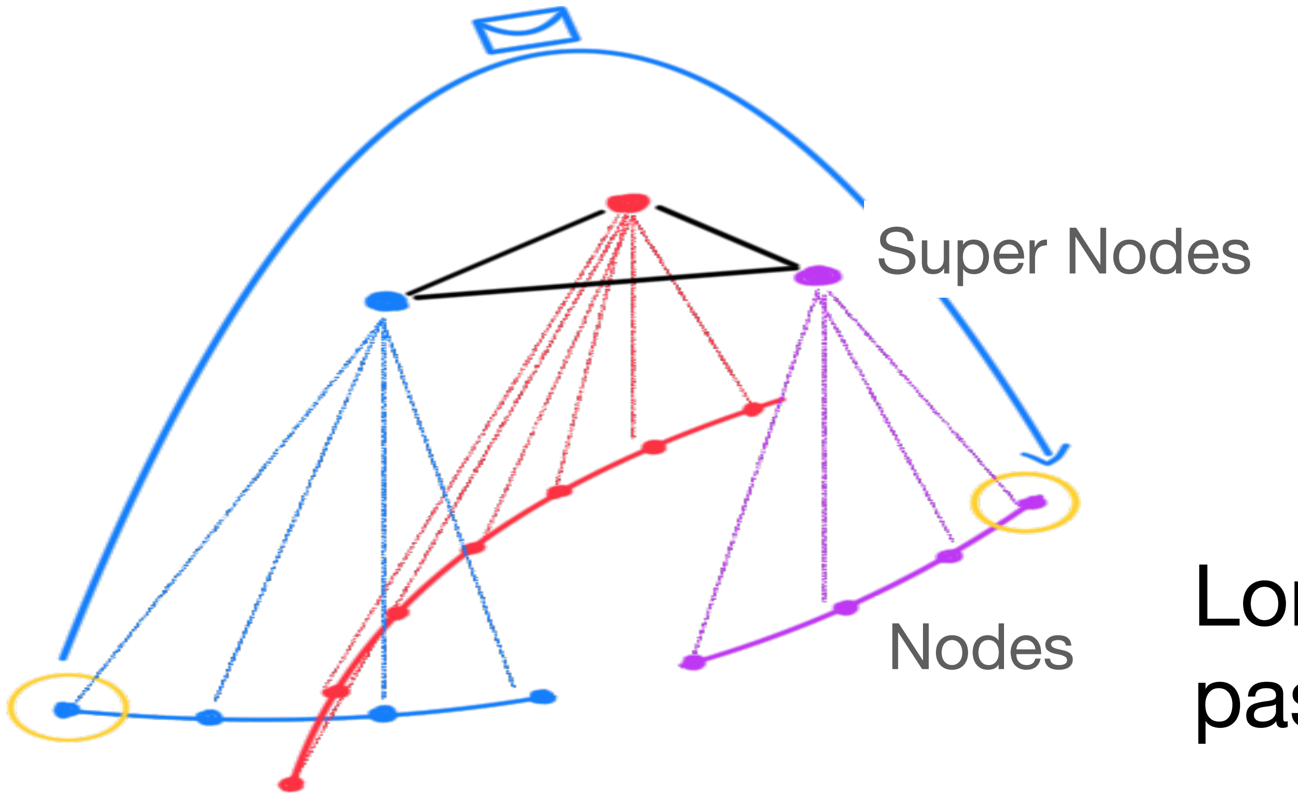




A sketch of Interaction Network

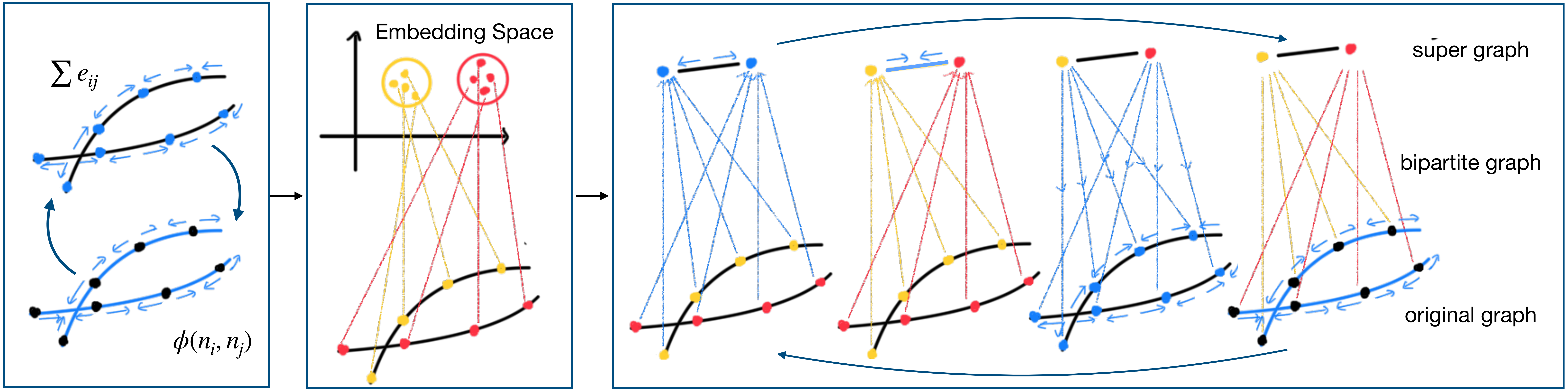
Short Comings of “Flat” GNNs

- The current Exa.TrkX pipeline is “flat”: it can only pass messages between directly connected nodes. This make the pipeline very sensitive to the quality of the graph

Current Problem	Proposed Solutions
Performance limited by input graph	Make predictions less graph-dependent
Message passing obstructed by inefficiencies	Construct hierarchical structure
Flat GNN  Inefficient graph construction forbids message passing	Hierarchical GNN  Long distance message passing is possible

Hierarchical GNN for Robust Track Reconstruction

- A Hierarchical GNN can:
  1. Pass long-distance messages across missing edges (Rampášek et al., 2021)
  2. Capture higher level structure such as particles rather than just space points. (Xing et al., 2021)

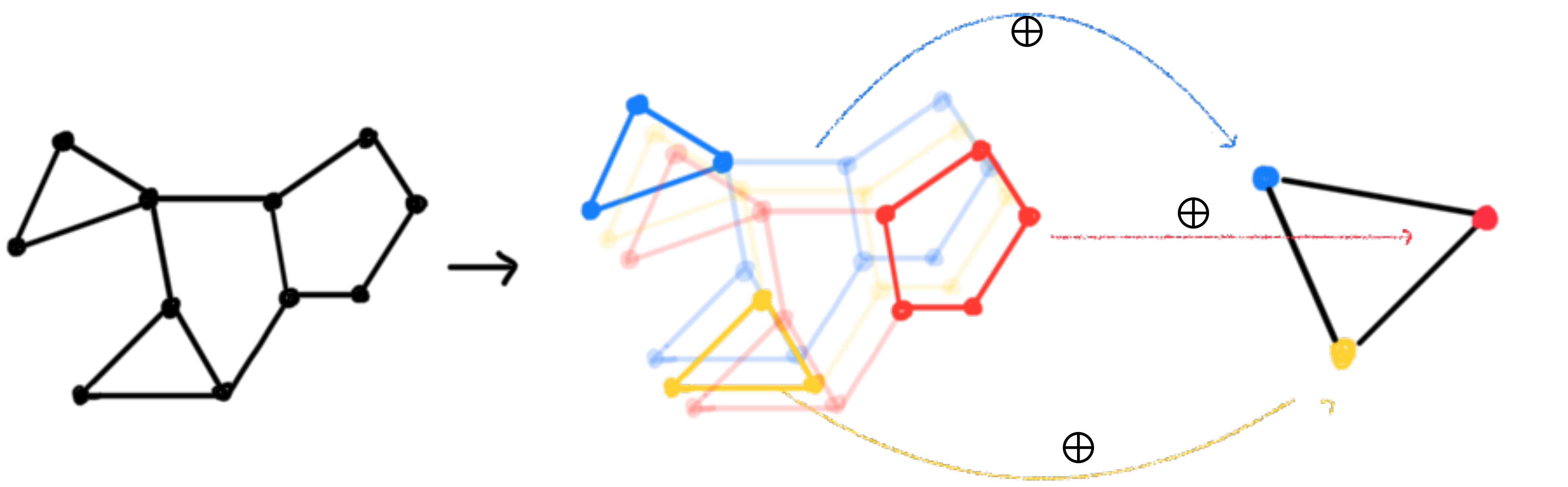


Flat GNN | Construct the Hierarchy | Hierarchical GNN

Previous Work: Pooling and Graph Construction

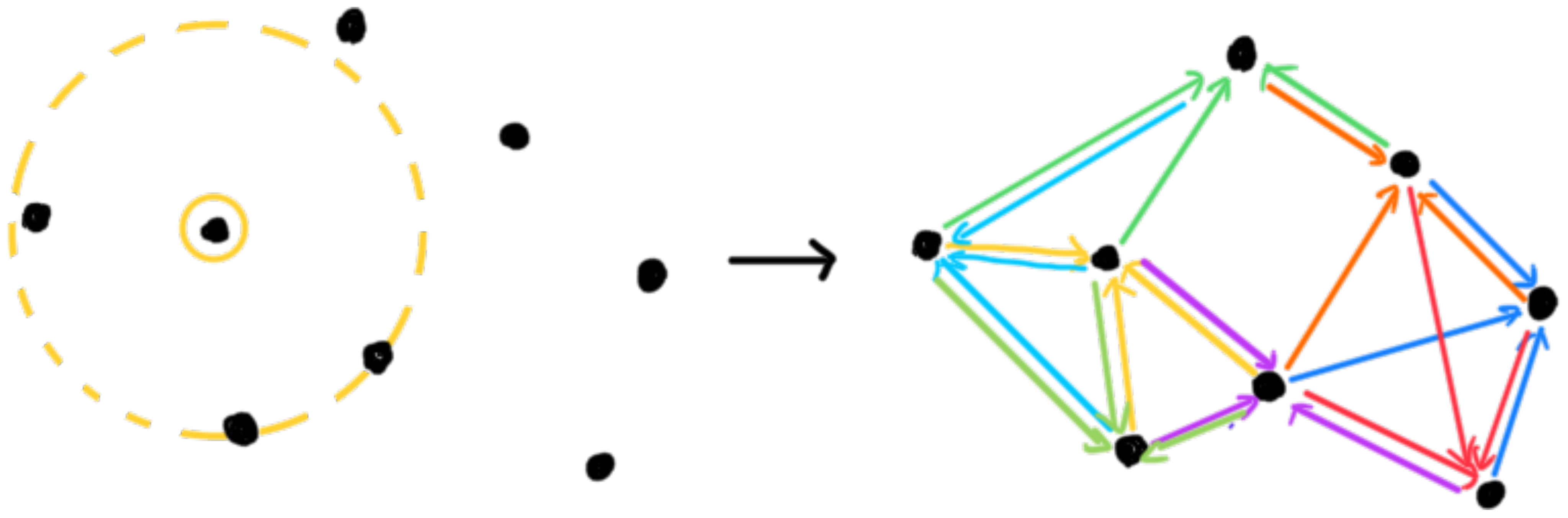
- DiffPool proposes to use GNN to generate an **assignment matrix** $S \in \mathbb{R}^{N \times K}$, then aggregate node features $N \in \mathbb{R}^{N \times D}$ to form supernode features $X = S^T N \in \mathbb{R}^{K \times D}$, and finally create super graph by $A' = S^T A S$ where A is the adjacency matrix.
- GravNet builds a dynamic graph by using **potential weighted edge** and **k-nearest neighbor graph** to guarantee its sparsity. This graph is updated for each iteration.

DiffPool



Original Graph — Assignment Matrix — Pooling Result

GravNet



KNN search — Potential weighted message passing

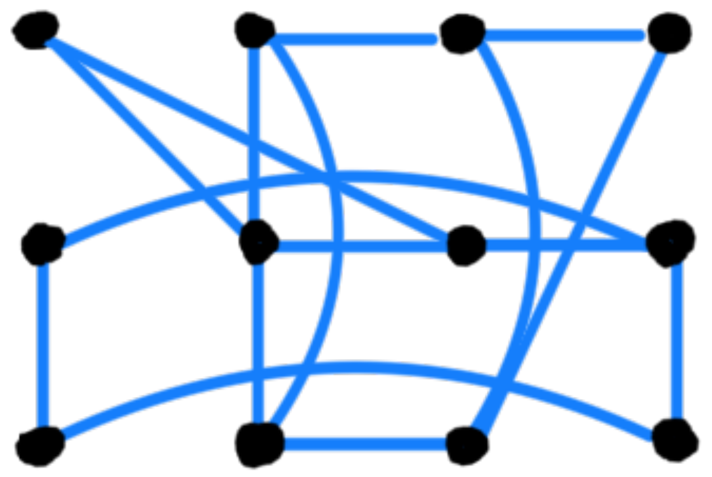
Criteria for Pooling Algorithm

- **Sparseness** guarantees that the time complexity remains linear instead of quadratic.
- **Differentiability** is essential for gradient-based learning algorithms.
- **Variable number of clusters** is important in the context of HEP since number of collisions obey poisson distribution instead of being a constant.
- **Soft assignments** allow each node to have connections with multiple clusters, which is crucial for message passing between the super graph and original graph.

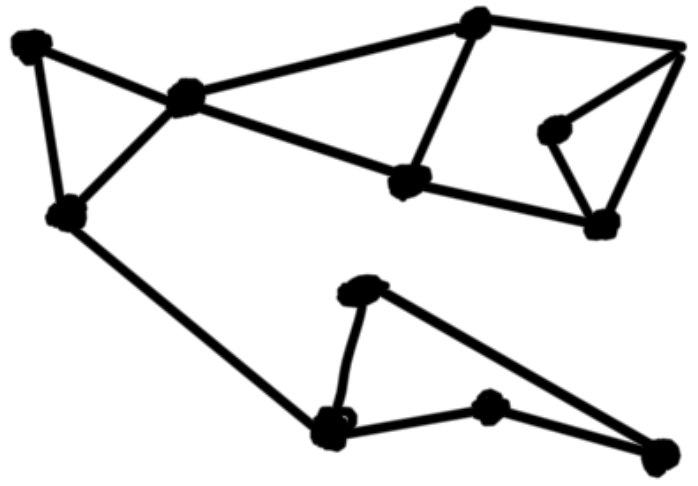
Model	Sparse	Differentiable	Variable #Clusters	Soft Assignment
DiffPool	✘	✓	✘	✓
GarNet	✘	✓	✘	✓
SAGPool	✓	✓	✘	✘
EdgePool	✓	✓	✓	✘
Our HGNN	✓	✓	✓	✓

Overview of the Algorithm

A Graph

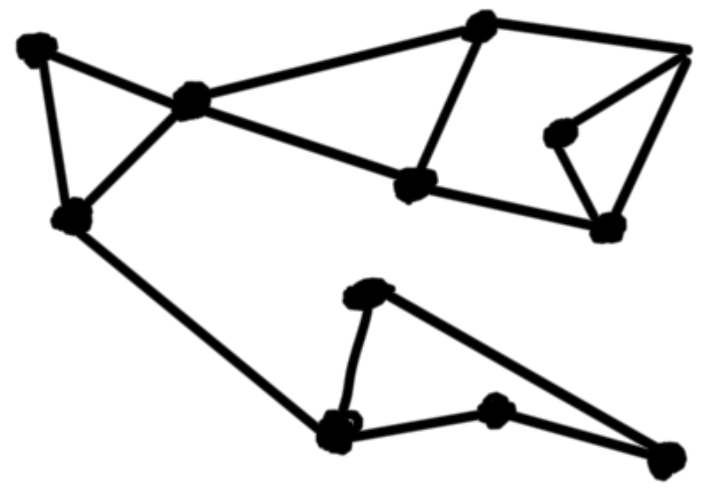


2D Embeddings

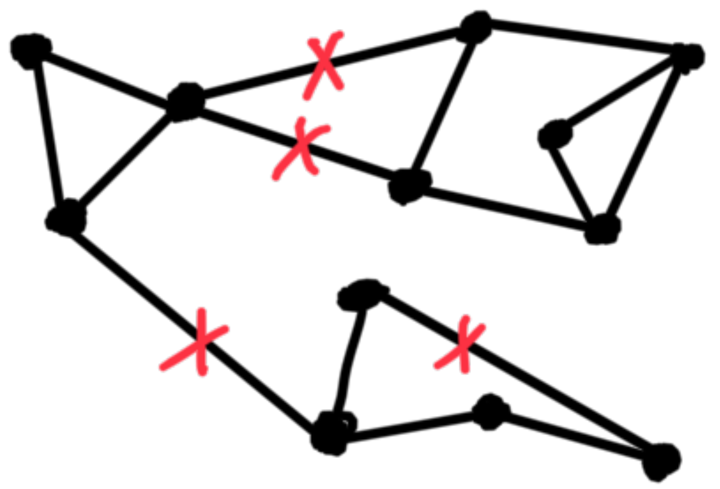


Generate Node Embeddings

2D Embeddings

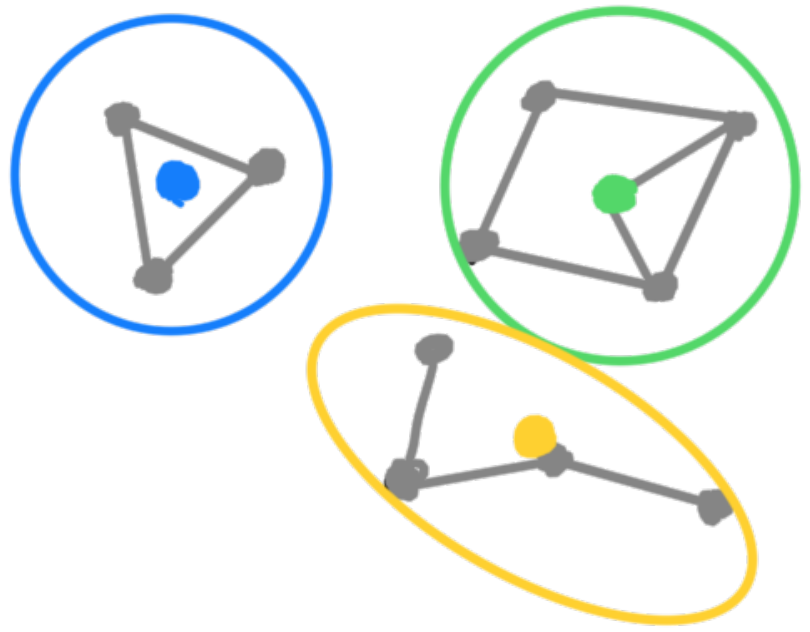


Graph Segmentation



Identify Clusters by Cutting Distant Connections

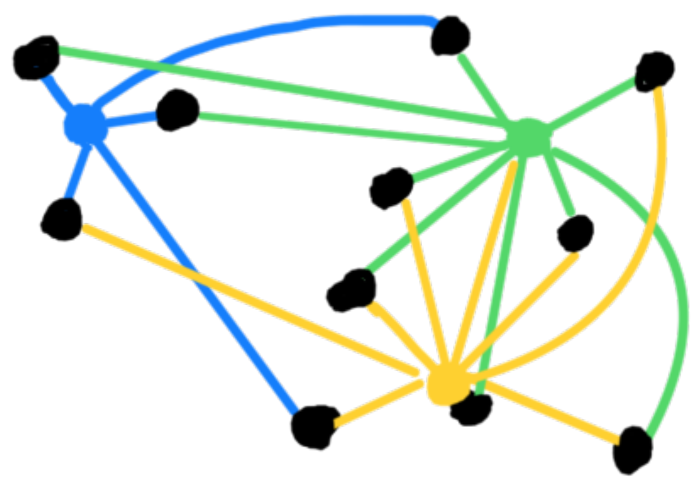
Cluster Embeddings



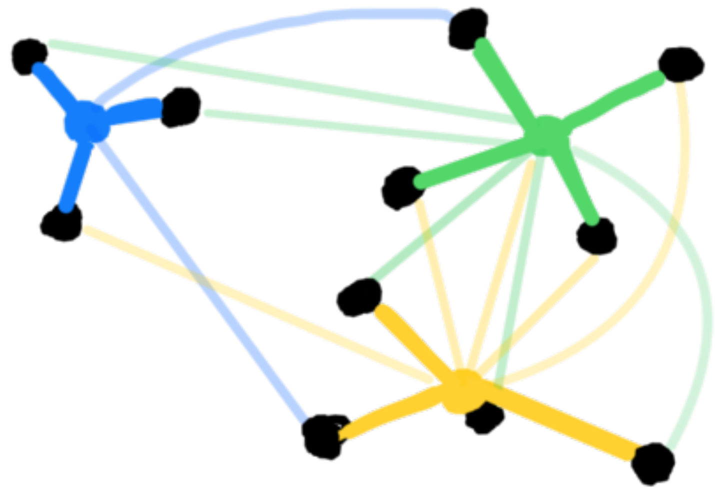
Given the node embeddings, we are able to define the centroid of each cluster, which will play a crucial role when we try to soften the assignments.

Generate Cluster Representation

Build KNN Graph



Weight by Distance



Softly Assign Nodes to Supernodes (Clusters)

Possible Loss Functions for HGNN

- Metric learning: embed nodes into an embedding space and impose **hinge embedding loss**. Use **spatial clustering algorithms** to select track candidates.

  Pros: applicable to both HGNN and vanilla GNN.

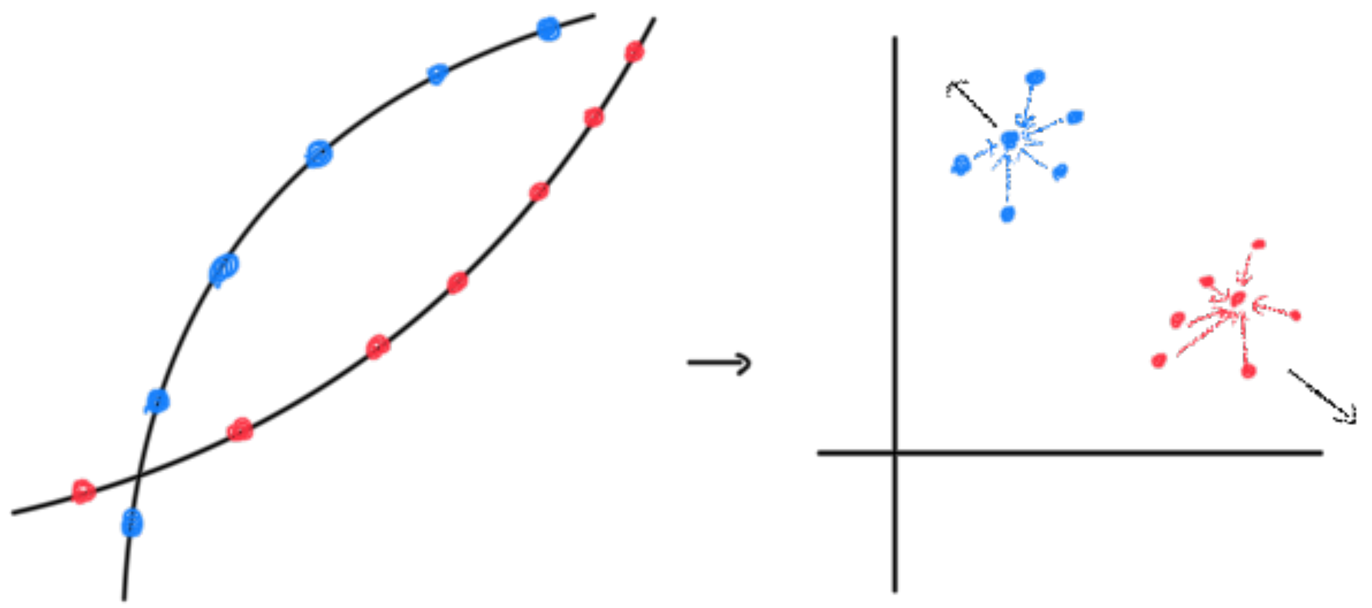
  Cons: spatial clustering algorithms are typically quadratic in size of point cloud.

- Bipartite Classifier: at the end of HGNN, generate **edge scores** for assignments (bipartite edges). Select track candidates by applying a score cut.

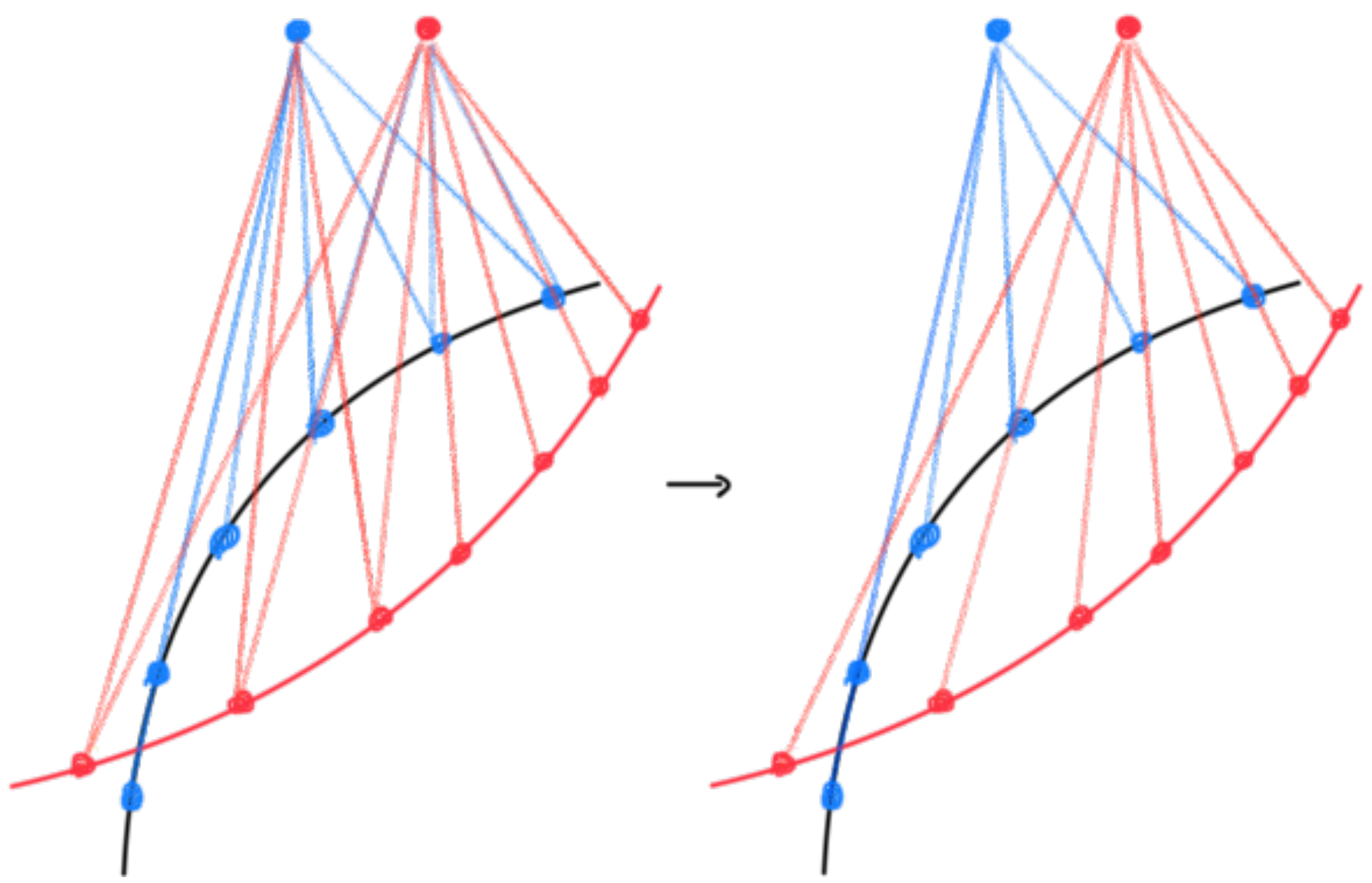
  Pros: capable of matching one hit to multiple tracks (shared space points)

  Cons: ground truth of assignments is undefined, needs a matching process.

Metric Learning



Bipartite Classifier



Dataset and Evaluation

- There are two **filter-processed** (see Exa.TrkX pipeline for reference) datasets:
  1. Full TrackML dataset: 2200 events of $O(7k)$ particles and $O(120k)$ spacepoints per event.
  2. TrackML 1GeV background cut (i.e. remove any particle that has $p_T < 1GeV$): 320 events of $O(1k)$ particles and $O(10k)$ spacepoints per event.
- Evaluation metrics:

Tracking efficiency: $\dfrac{\text{\# matched reconstructable particles}}{\text{\# reconstructable particles}}$

Tracking purity: $\dfrac{\text{\# matched reconstructable particles}}{\text{\# track candidates - \# matched non-reconstructable particles}}$

1. Matched means that the candidate finds more than 50% of the particle and the particle occupies more than 50% of the candidate
2. Reconstructable particles are those which left more than 5 hits in the detector and has a $p_T > 1GeV$

Experiment Results—TrackML1GeV

- Embedding models provide an apple-to-apples comparison between HGNN and flat GNN.
- Bipartite classifier is HGNN with all its power unleashed with a loss function designed for it.
- Edge classifier is a baseline model, which is also part of the standard Exa.TrkX pipeline.
- Truth CC (truth connected components) is a measure of graph quality, which takes the input graph and prune it down with ground truth. This is the upper bound of edge classifier model performance.
- Timing performance of embedding models are dominated by spatial clustering algorithm (HDBSCAN).

	Embedding HGNN	**Embedding IN**	**Bipartite HGNN**	**Edge Classifier IN**	**Truth CC**
Tracking efficiency	97.32%	98.16%	98.86%	98.54%	99.91%
Tracking purity	95.78%	90.15%	98.76%	93.79%	95.28%
Time	**0.5280**	**0.3514**	**0.2625**	**0.2108**	**N/A**

Blue labels represent the best models; Green labels are the second best models.

Experiment Results—Inefficient Graph

- Randomly remove 20% of edges from the input graph to test models' robustness against inefficient input graphs.
- Edge classifier models are significantly impacted by inefficiencies of input graph.
- When the graph is inefficient, HGNN outperforms flat GNN on node embedding task.
- Bipartite classifiers has a tracking efficiency higher than Truth CC, which means that it has successfully reconstructed some of the broken tracks.

	Embedding HGNN	Embedding IN	Bipartite HGNN	Edge Classifier IN	Truth CC
Tracking efficiency	97.33%	92.78%	98.83%	91.93%	97.19%
Tracking purity	94.08%	92.19%	98.53%	74.52%	78.66%

Blue labels represent the best models; Green labels are the second best models.

Experiment Results—Full TrackML

- The full TrackML contains more spacepoints, which makes embedding a harder task. Edge classifier thus becomes more competitive since its inference is localized on graph.
- Bipartite Classifier was able to provide better tracking efficiency compared with Truth CC.
- In terms of embedding performance, Hierarchical GNN is always better than Interaction Network.
- Timing performance of embedding models become worse as its time complexity is quadratic.

	Embedding HGNN	Embedding IN	Bipartite HGNN	Edge Classifier IN	Truth CC
Tracking efficiency	94.70%	93.80%	97.80%	96.36%	97.75%
Tracking purity	32.74%	32.92%	35.31%	31.57%	28.27%
Time	**8.1718**	**7.9430**	**1.0262**	**0.4188**	**N/A**

Blue labels represent the best models; Green labels are the second best models.

Conclusion

- Hierarchical Graph Neural Network:
    1. HGNN is a variant of GNNs where a set of supernodes are created as coarsened representation of the original graph
    2. No additional supervision needed for the hierarchical structure construction.
    3. HGNN can recover broken tracks and is more robust against inefficiencies.
    4. HGNN is capable of performing message passing process across long distances.
- Ongoing process to train better Bipartite Classifier!
- All codes are available on GitHub and paper is coming soon!

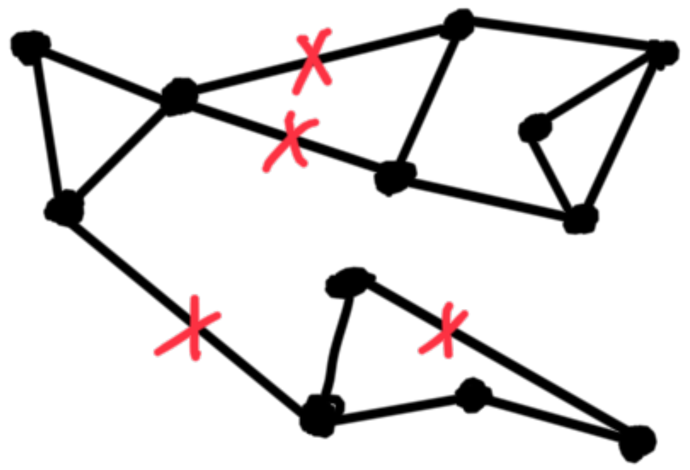
References

1. Battaglia, P., Pascanu, R., Lai, M., & Jimenez Rezende, D. (2016). Interaction networks for learning about objects, relations and physics. Advances in neural information processing systems, 29.
2. Diehl, F. (2019). Edge contraction pooling for graph neural networks. arXiv preprint arXiv:1905.10990.
3. Ju, X., Murnane, D., Calafiura, P., Choma, N., Conlon, S., Farrell, S., ... & Lazar, A. (2021). Performance of a geometric deep learning pipeline for HL-LHC particle tracking. The European Physical Journal C, 81(10), 1-14.
4. Kipf, T. N., & Welling, M. (2016). Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907.
5. Lee, J., Lee, I., & Kang, J. (2019, May). Self-attention graph pooling. In International conference on machine learning (pp. 3734-3743). PMLR.
6. Qasim, S. R., Kieseler, J., Iiyama, Y., & Pierini, M. (2019). Learning representations of irregular particle-detector geometry with distance-weighted graph networks. The European Physical Journal C, 79(7), 1-11.
7. Rampášek, L., & Wolf, G. (2021, October). Hierarchical graph neural nets can capture long-range interactions. In 2021 IEEE 31st International Workshop on Machine Learning for Signal Processing (MLSP) (pp. 1-6). IEEE.
8. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., & Bengio, Y. (2017). Graph attention networks. arXiv preprint arXiv:1710.10903.
9. Xing, Y., He, T., Xiao, T., Wang, Y., Xiong, Y., Xia, W., ... & Soatto, S. (2021). Learning hierarchical graph neural networks for image clustering. In Proceedings of the IEEE/CVF International Conference on Computer Vision (pp. 3467-3477).
10. Ying, Z., You, J., Morris, C., Ren, X., Hamilton, W., & Leskovec, J. (2018). Hierarchical graph representation learning with differentiable pooling. Advances in neural information processing systems, 31.

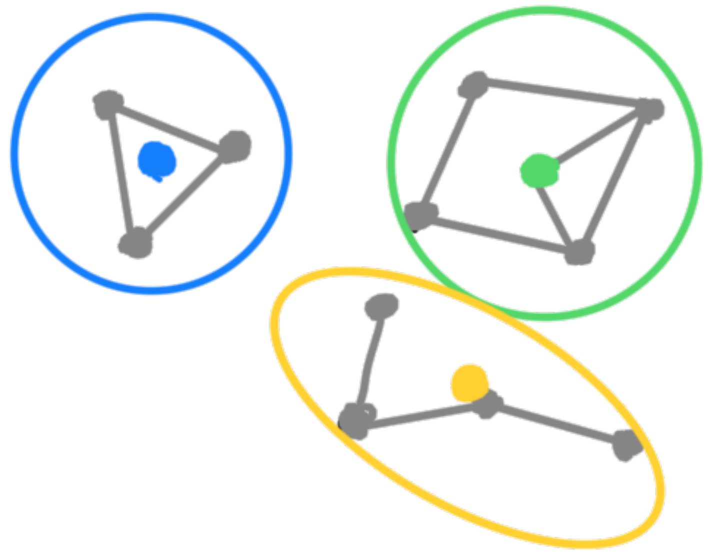
Backups

Backup: Clustering and Cluster Embeddings

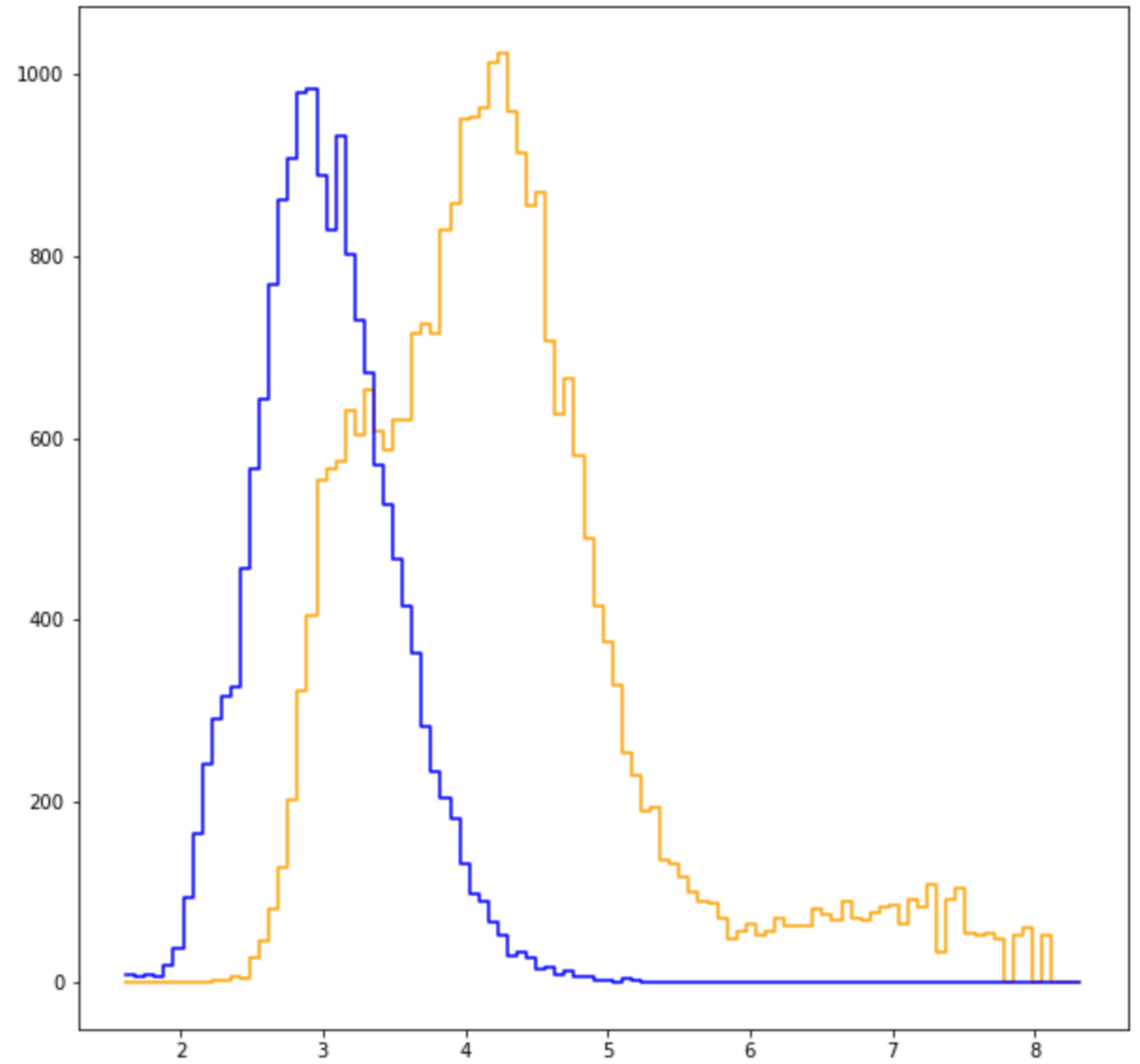
Graph Segmentation



Cluster Embeddings



Distribution of s_{ij}



Blue: fake edges
Yellow: true edges

Algorithm 1: Determine Score Cut

Input $\{n_i\}, G, r$

$s_{ij} \leftarrow \tanh^{-1}(n_i \cdot n_j) \quad \forall (i,j) \in G$

$p_{in}(s), p_{out}(s) \leftarrow \text{GaussianMixtureModel}[\{s_{ij}\}]$

$s_{cut} \leftarrow \text{Solve}[\ln(p_{in}(s)) - \ln(p_{out}(s)) = r]$

Return s_{cut}, s_{ij}

r: cluttering granularity; $\{n_i\}$: node embeddings

Algorithm 2: Compute Cluster Embeddings

Input $\{n_i\}, G, s_{cut}, N_{min}$

$\{C_\alpha\} \leftarrow \text{ConnectedComponents}[\{(i,j) \mid s_{ij} > s_{cut}\}]$

$\{C_\alpha\} \leftarrow \{C_\alpha \mid N(C_\alpha) > N_{min}\}$

$X_\alpha \leftarrow \text{normalize}\left[\frac{1}{N(C_\alpha)} \sum_{i \in C_\alpha} n_i\right]$

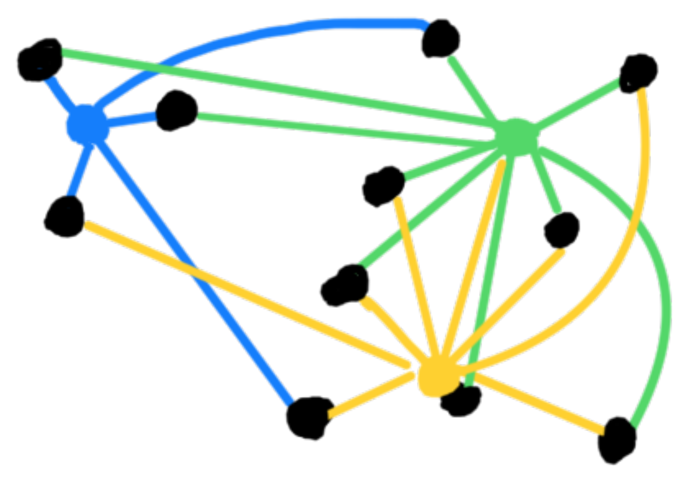
Return X_α

N_{min}: minimum size of a cluster

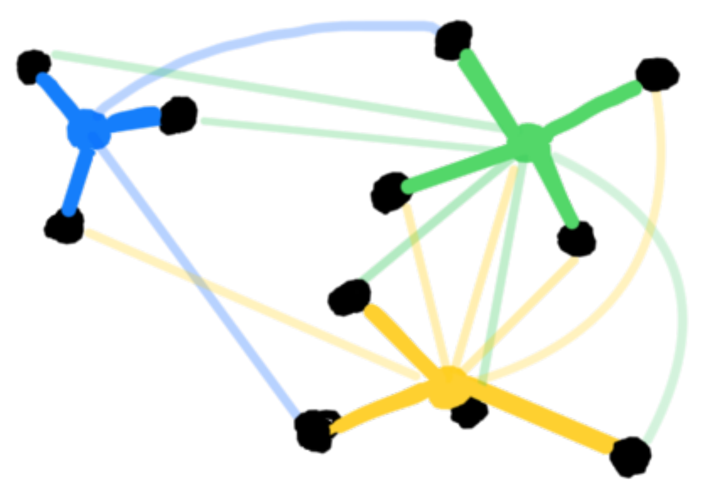
*the embedding space in this work is $\mathbb{S}^n$ instead of $\mathbb{R}^n$: we normalize all embeddings under L_2 norm

Backup: Assignment Graph Construction

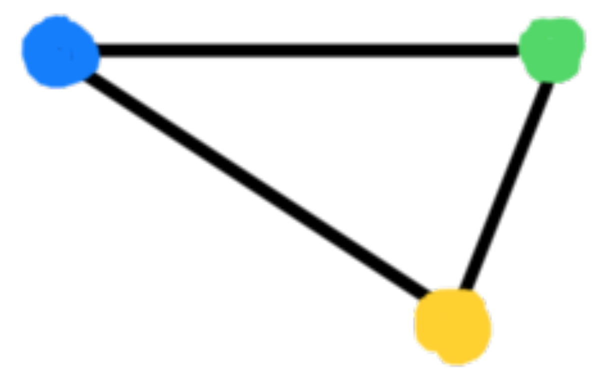
Build KNN Graph



Weight by Distance



The Super Graph



Algorithm 3: Build KNN graph

Input $\{n_i\}, \{X_\alpha\}, G, c$

For each $i \in G$**:**

Find $\mathrm{id}_i(k) :=$ n_i's k-th nearest X's index

$\mathcal{N}(i) \leftarrow \{\mathrm{id}_i(k) \,|\, k \leq c\}$

Return $\{\mathcal{N}(i) \,|\, i \in G\}$

c: desired connectivity

Algorithm 4: Weight Edges by Similarity

Input $\{n_i\}, \{X_\alpha\}, G, \{\mathcal{N}(i)\}, f(s)$

$G_{assignment} \leftarrow \{(i, \alpha) \,|\, \alpha \in \mathcal{N}(i),\, i \in G\}$

$s_{i\alpha} \leftarrow \mathrm{BatchNorm}[n_i \cdot X_\alpha] \quad \forall (i, \alpha) \in G_{assignment}$

$$w_{i\alpha} = \frac{f(s_{i\alpha})}{\sum_{\alpha \in \mathcal{N}(i)} f(s_{i\alpha})}$$

Return $G_{assignment}, w_{i\alpha}$

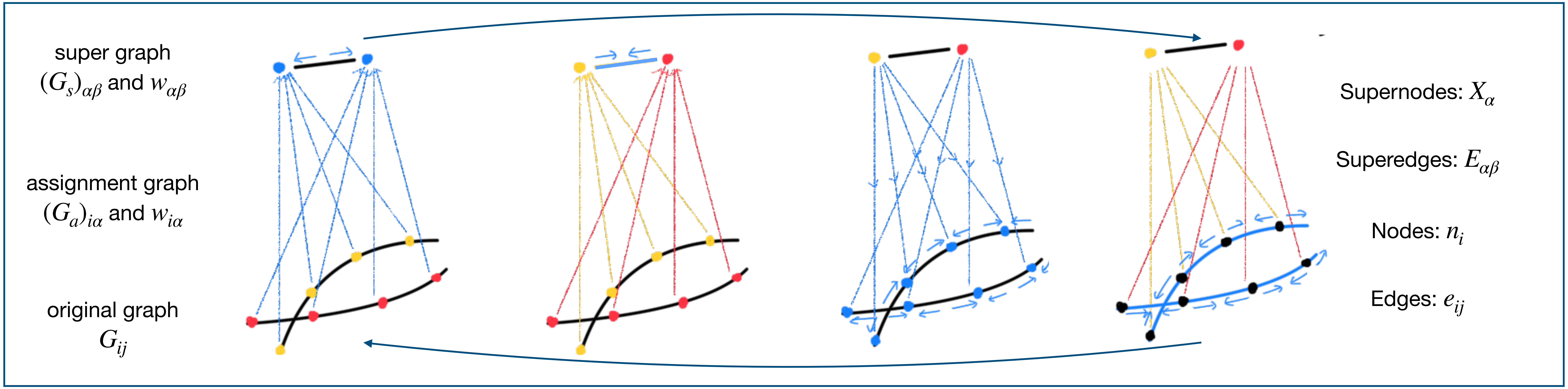
Build the Super Graph

To build the super graph, the procedure is completely identical, except that we must:

1. Use $\{X_\alpha\}$ as both source and destination.
2. Symmetrize the graph
3. Use a different weighting $f(s)$

f is chosen to be e^s for assignment graph, and sigmoid(s) for super graph.

Backup: Hierarchical Message Passing



Step 1: Update Supernode

$\text{input}_1(\alpha) \leftarrow \sum_{i:(i,\alpha)\in G_a} w_{i\alpha} n_i$

$\text{input}_2(\alpha) \leftarrow \sum_{\beta:(\alpha,\beta)\in G_s} w_{\alpha\beta} E_{\alpha\beta}$

$X_\alpha \leftarrow \phi_{sn}(X_\alpha, \text{input}_1(\alpha), \text{input}_2(\alpha))$

Step 2: Update Superedge

$E_{\alpha\beta} \leftarrow \phi_{se}(E_{\alpha\beta}, X_\alpha, X_\beta)$

Step 3: Update Node

$\text{input}_1(i) \leftarrow \sum_{\alpha:(i,\alpha)\in G_a} w_{i\alpha} X_\alpha$

$\text{input}_2(i) \leftarrow \sum_{j:(i,j)\in G} e_{ij}$

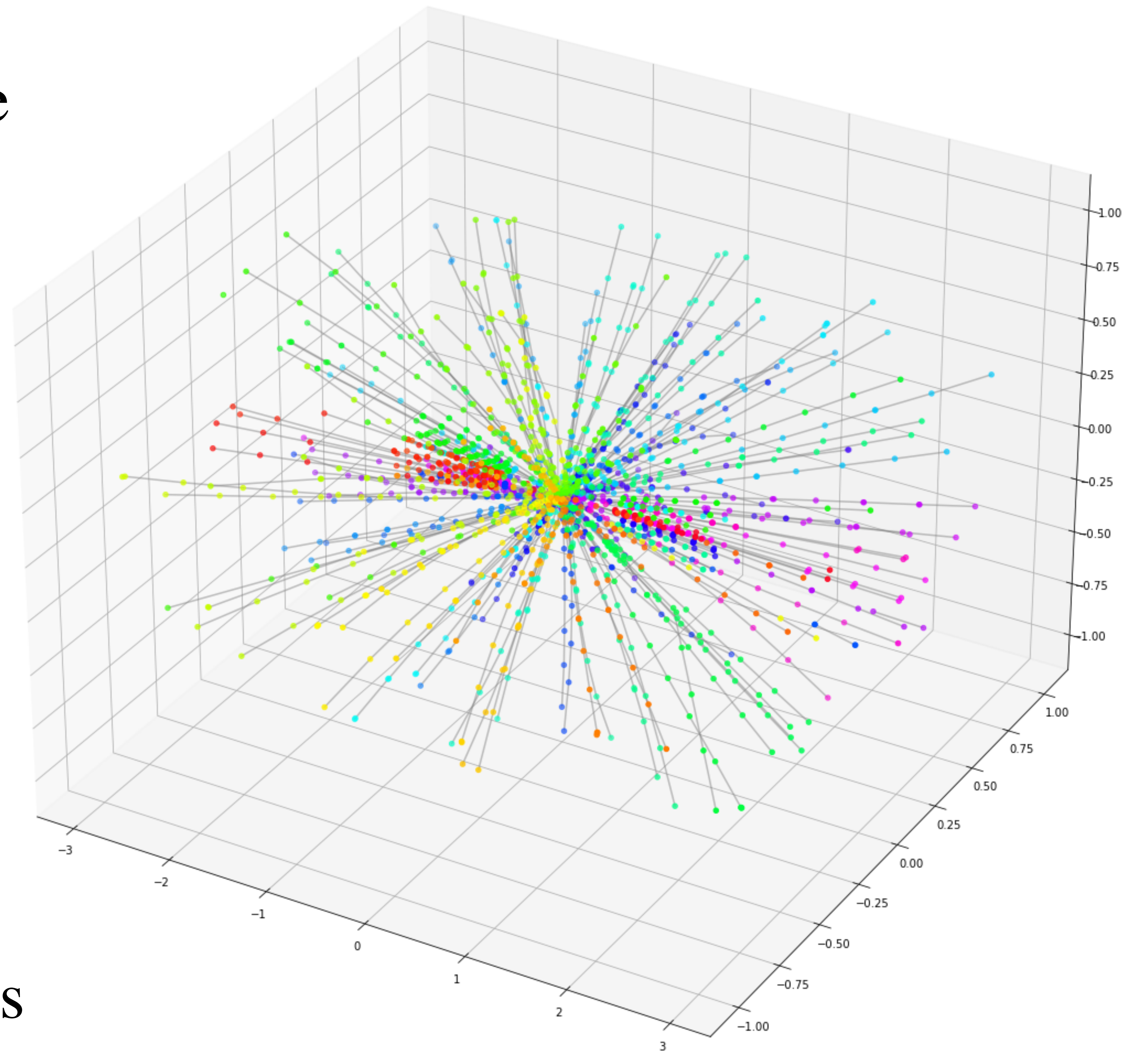
$n_i \leftarrow \phi_n(n_i, \text{input}_1(i), \text{input}_2(i))$

Step 4: Update Edge

$e_{ij} \leftarrow \phi_e(e_{ij}, n_i, n_j)$

Backup: What Does the Algorithm Learn?

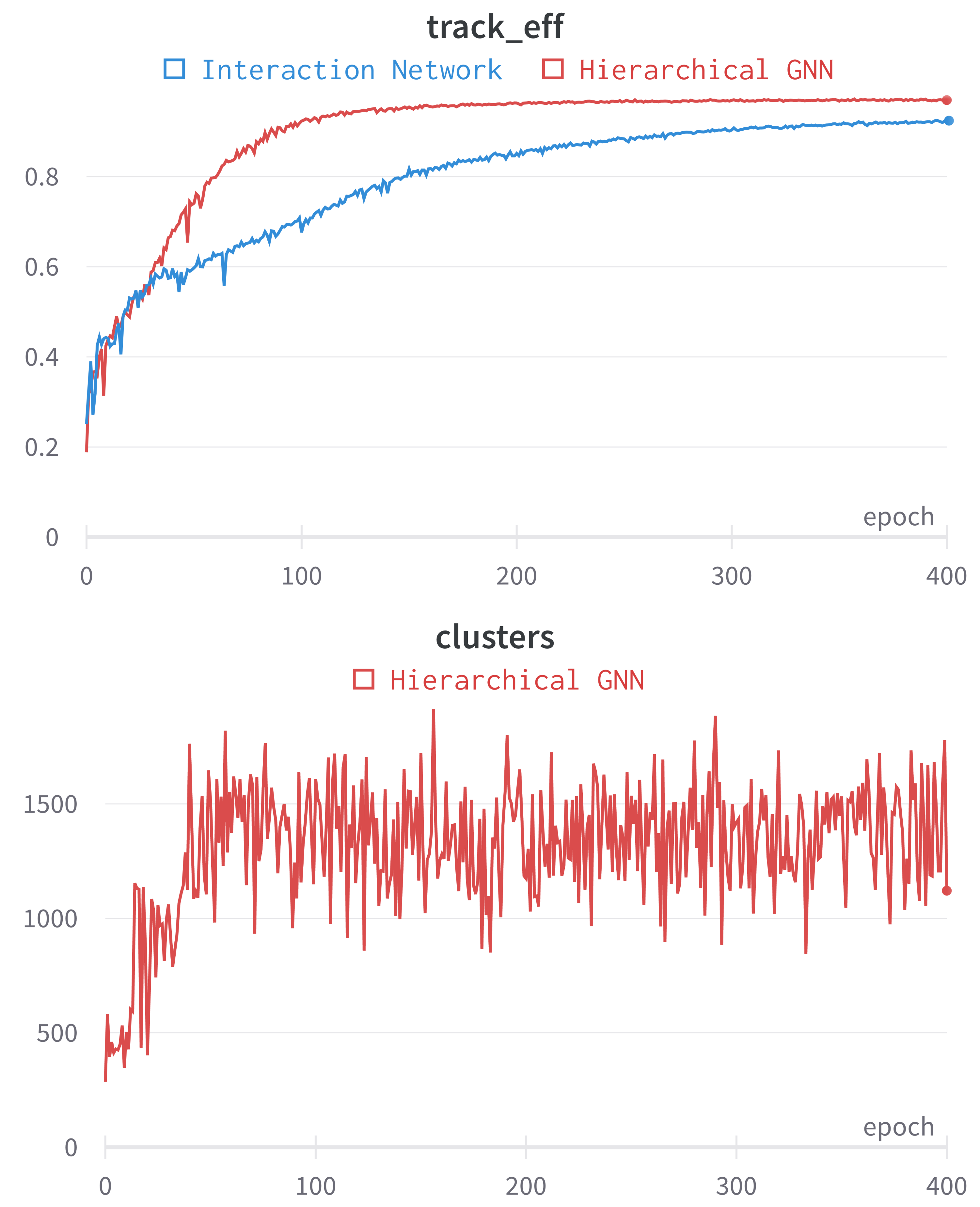
- For small sized event (e.g. 1GeV cut), it is possible to leave intermediate embedding space completely unsupervised.
- To demonstrate the properties of the intermediate embedding space, we do the following visualization:
    1. Use T-SNE transformation to reduce the dimensionality from $n \sim O(10)$ to 1
    2. Randomly select k particles and plot them in 3D space using spacepoint coordinates in the detector
    3. Use color maps to color each spacepoint by the reduced embeddings, which is now one dimensional
- Such visualization scheme can capture the distance relation in high-dimensional embedding space by coloring closer hits with similar colors.



TrackML1GeV Node Embedding
Intermediate Embedding Space

Backup: Training Behavior

- It's worth noting that even if HGNN has more parameter and more complicated architecture, **its convergence is no worse than** flat GNN.
- Surprisingly, on the TrackML1GeV dataset, the model learns about the **same number of clusters and particles** (untrue for Full TrackML where low-pT tracks dominate the event)
- As we move on to Full TrackML, a "training wheel loss" is needed for training. It is basically a hinge embedding loss which fades out after 50 epochs.



TrackML1GeV, showing first 400 epochs