Graph Neural Networks for Track Finding in ALERT

Mathieu Ouillon (Mississippi State University)

MISSISSIPPI STATE UNIVERSITY

Jefferson Lab

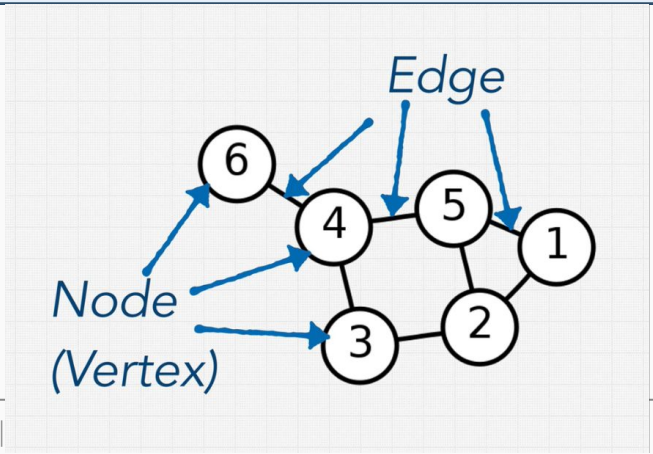
Introduction to Graph and Graph Neural Networks

Graph

A graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ is a data structure consisting of:

- Nodes $\mathbf{V} = \{\mathbf{v_1}, \cdots, \mathbf{v_N}\}$: the objects (e.g., detector hits)
- Edges $\mathbf{E} \subset \mathbf{V} \times \mathbf{V}$: the relationships between them

Each node carries a feature vector $\mathbf{x_i} \in \mathfrak{R}^{\mathbf{d}}$ and each edge can carry its own features $\mathbf{e_{ij}} \in \mathfrak{R}^{\mathbf{d_e}}$.



ALERT is designed to detect recoil nuclei

- Accurate track finding is crucial for reconstructing particle trajectories and physics kinematics.
- Traditional methods (like Multi-Layer Perceptrons, or CNN) struggle in high-multiplicity or noisy environments.

Goal: Implement and validate a Graph Neural Network (GNN) approach for robust track finding in ALERT.

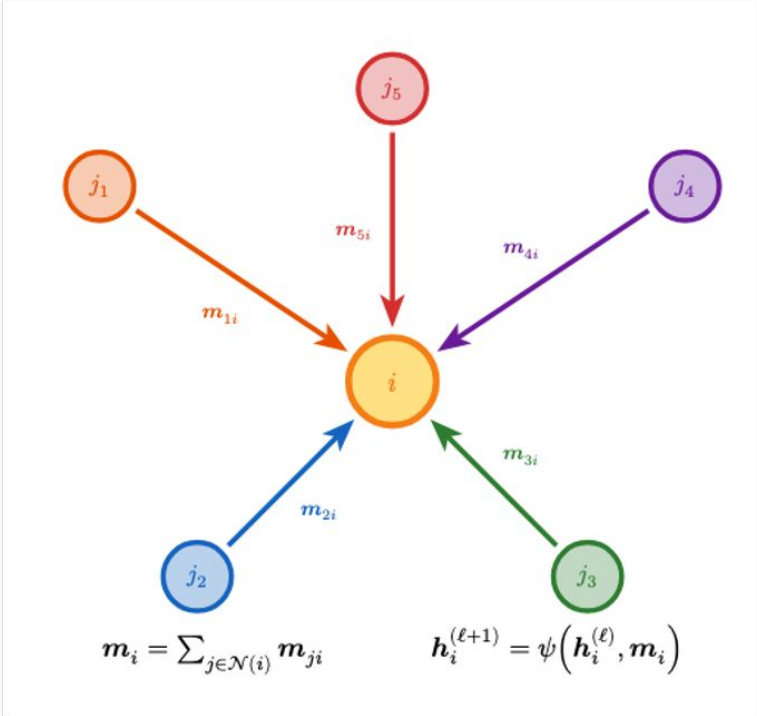
Graphs naturally encode arbitrary pairwise relationships between hits.

Message Passing: The Core of GNNs

Message Passing

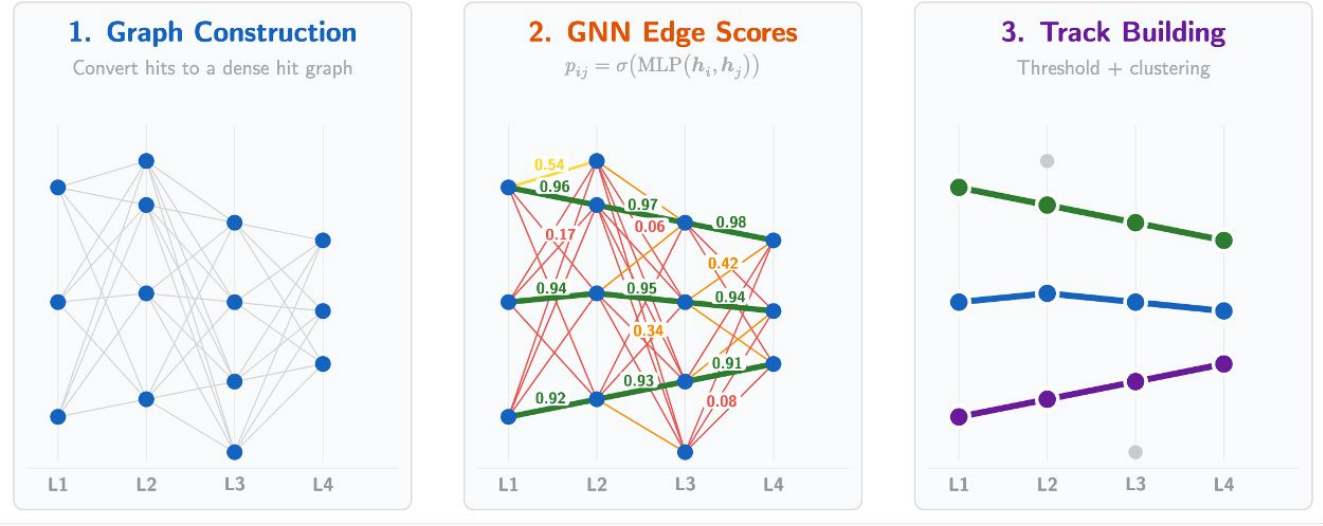
At each GNN layer ℓ, every node collects information from its neighbors, aggregates it, and updates its own representation. After L layers, each node encodes information from its L-hop neighborhood.

This allows the network to learn global track structures rather than just local pairwise features, making it highly robust against noise and overlapping tracks.



Edge Classification for Track Finding

The GNN assigns a score $e_{ij} \in [0, 1]$ to every edge, indicating the probability that hits i and j belong to the same particle track. High-scoring edges are kept; connected components in the surviving graph become track candidates.



The GNN transforms a dense candidate graph (left) into clean track candidates (right) by scoring each edge and keeping only the survivors.

Graph Construction and Training

Node Features (10D)

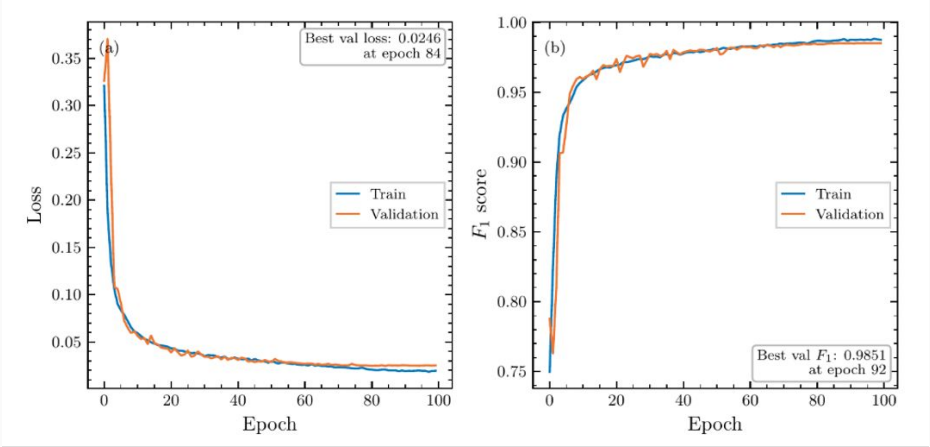
Each detector hit becomes a node with:
layer index, φ, r (normalized), stereo angle, wire midpoint (x, y, z), wire direction (u_x, u_y, u_z)

Edge Features (8D)

$\Delta\varphi$, layer gap, z_1, z_2, r_1, r_2, Δstereo, same-detector flag

Training Data

To simulate multi-track events:
– Merge 1–5 single-track events into one event
– Inject 5–30 noise hits
– 1M events: 75% train / 12.5% val / 12.5% test



Best validation F1 = 0.985 at epoch 92.
Train and validation curves track closely: no overfitting.

Track-Level Metric Definitions

Let $\mathcal{T} = \{T_1, \ldots, T_{ntrue}\}$ be true tracks (MC truth) and $\mathcal{R} = \{R_1, \ldots, R_{nreco}\}$ be reconstructed tracks (connected components from thresholded edge scores).

All metrics are defined in terms of hit-set overlaps $|\mathcal{R} \cap \mathcal{T}|$.

Purity (per reco track)

For each reconstructed track R_ℓ, purity is the fraction of its hits that belong to the true track with the largest intersection:

$$\text{purity}(R) = \max_{T \in \mathcal{T}} \frac{|R \cap T|}{|R|}$$

Efficiency

A true track Tk is declared found if its best-matching reconstructed track covers at least 80% of its hits.

$$\text{eff} = \frac{n_{found}}{n_{true}}$$

Completeness (per true track)

For each true track T_k, completeness is the fraction of its hits captured by the best-matching reconstructed track:

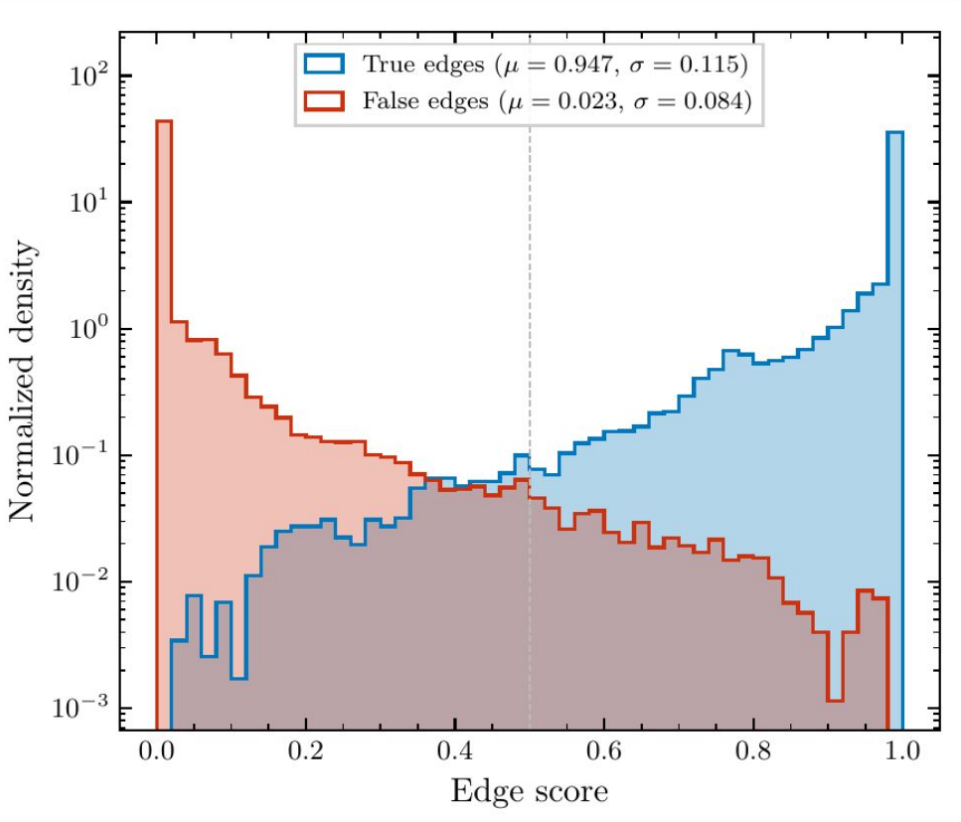
$$\text{compl.}\,(T) = \max_{R \in \mathcal{R}} \frac{|T \cap R|}{|T|}$$

Fake Rate

A reconstructed track R_ℓ is declared a fake if its purity falls below 50%.

$$\text{fake rate} = \frac{n_{fake}}{n_{reco}}$$

Edge Score Separation



Clean Separation

– True edges: μ = 0.947, σ = 0.115

– False edges: μ = 0.023, σ = 0.084

– 4 orders of magnitude density contrast in the tails

The small central overlap at scores 0.3–0.5 is where the classifier is genuinely uncertain: this is what drives the residual fake rate/contamination.

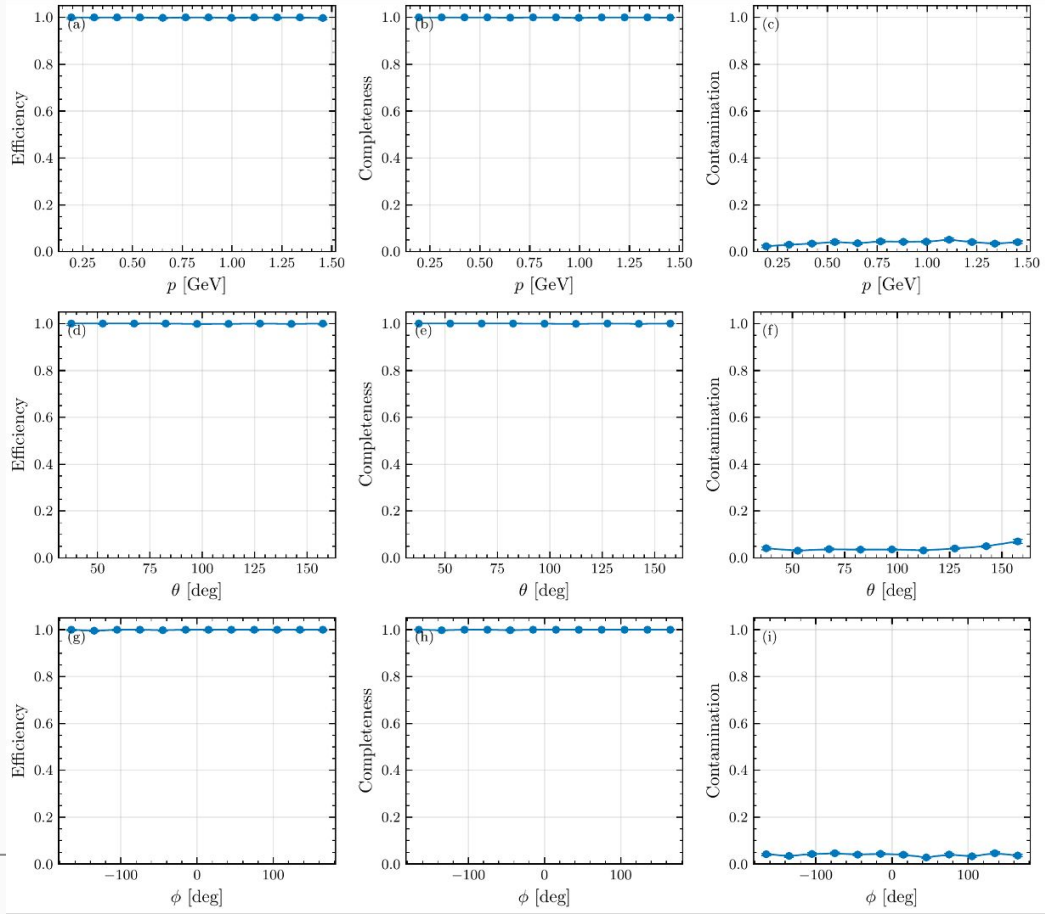
GNN vs MLP: Track-Level Performance

Metric	GNN	MLP	Improvement
Efficiency	99.95%	60.50%	+39.4 pp
Avg. completeness	99.96%	73.39%	+26.6 pp
Avg. contamination	8.56%	21.36%	−12.8 pp
Fake rate	5.90%	9.85%	−3.9 pp

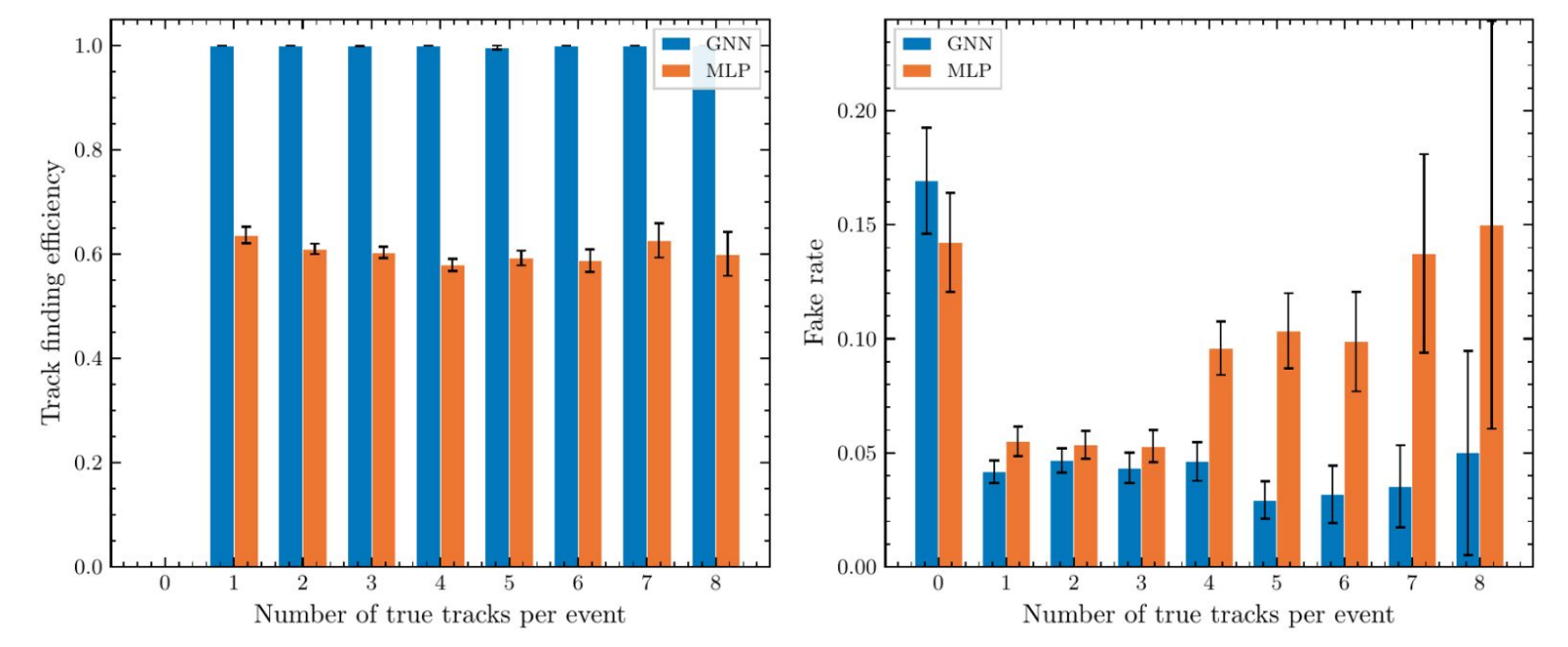
N_{events} = 2823, N_{true} = 5704 true tracks.
GNN threshold = 0.1, MLP threshold = 0.2.
The GNN finds nearly every true track while producing fewer fake candidates than the MLP.

Kinematic Uniformity

Efficiency, completeness, and contamination are flat across momentum p, polar angle θ, and azimuth φ



Robustness: Efficiency & Fake Rate vs Multiplicity



GNN: flat at 100% efficiency from 1 to 8 tracks/event. MLP: stuck at ~ 60% regardless.
GNN fake rate stays 3–5% while MLP climbs to 10–15% at high multiplicity.

GNN vs MLP: Inference speed

Compare the inference speed of GNN vs MLP in COATJAVA:

Metric	MLP	GNN
Tracks	**142**	**302**
Tracks/event	0.021	0.045
Time/event	0.025 ms	0.859 ms

Phase	Time / event	Share
Graph builder (Java, O(N²) edges)	0.008 ms	0.9%
Model inference (DJL forward)	**0.792 ms**	**98.1%**
CC track extractor (Java)	0.002 ms	0.3%
Cluster construction (Java + per-track PreClusterFinder)	0.004 ms	0.5%
Total "track finding"	**0.807 ms**	100%

MLP is faster on small event, but GNN is still fast.

A full event take around ~500ms so GNN is 0.16%

Most of the time for the GNN is the model inference. I try smaller models but the efficiency on simulation was lower.

GNN vs MLP: Performance on elastic data

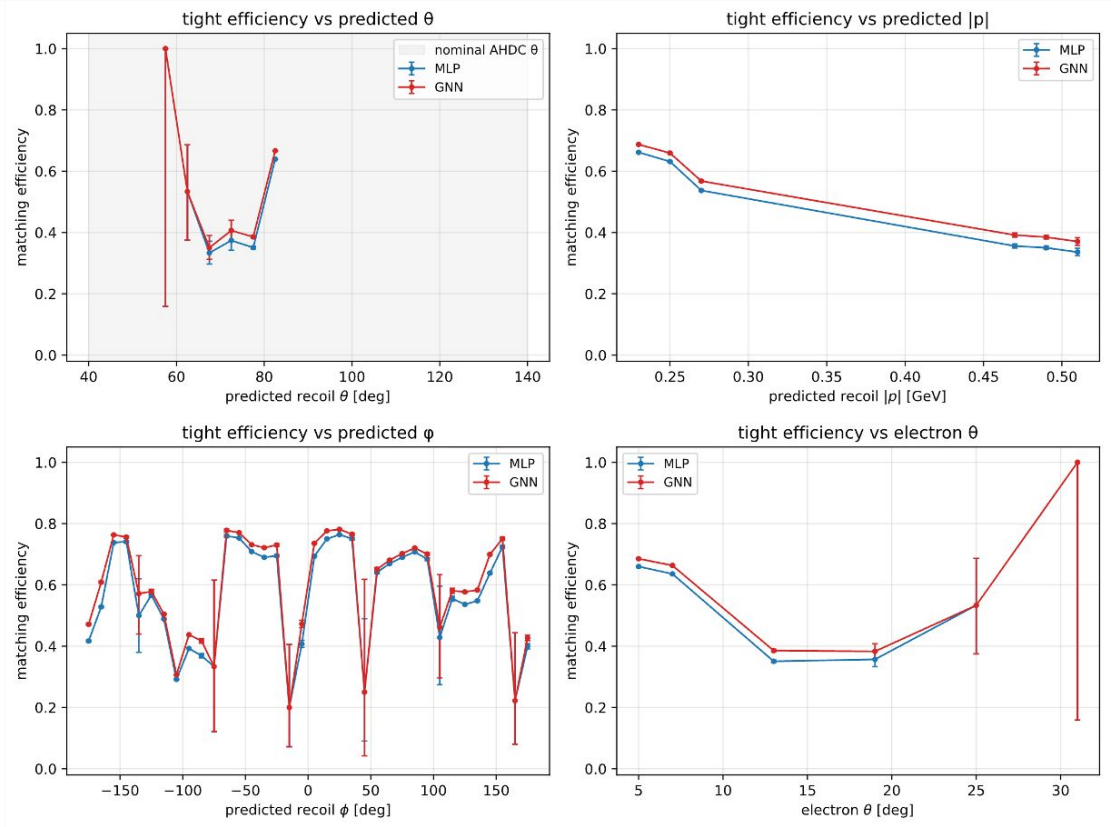
Elastic scattering on deuterium, use polar angle of the electron to compute it momentum and the expected kinematics of the recoil.

Look at the kinematics of the track found by MLP and GNN:

- A track is match when both angular residuals are inside the matching windows

- GNN found more tracks that pass the Helix fitter than MLP

- With no specific phase space

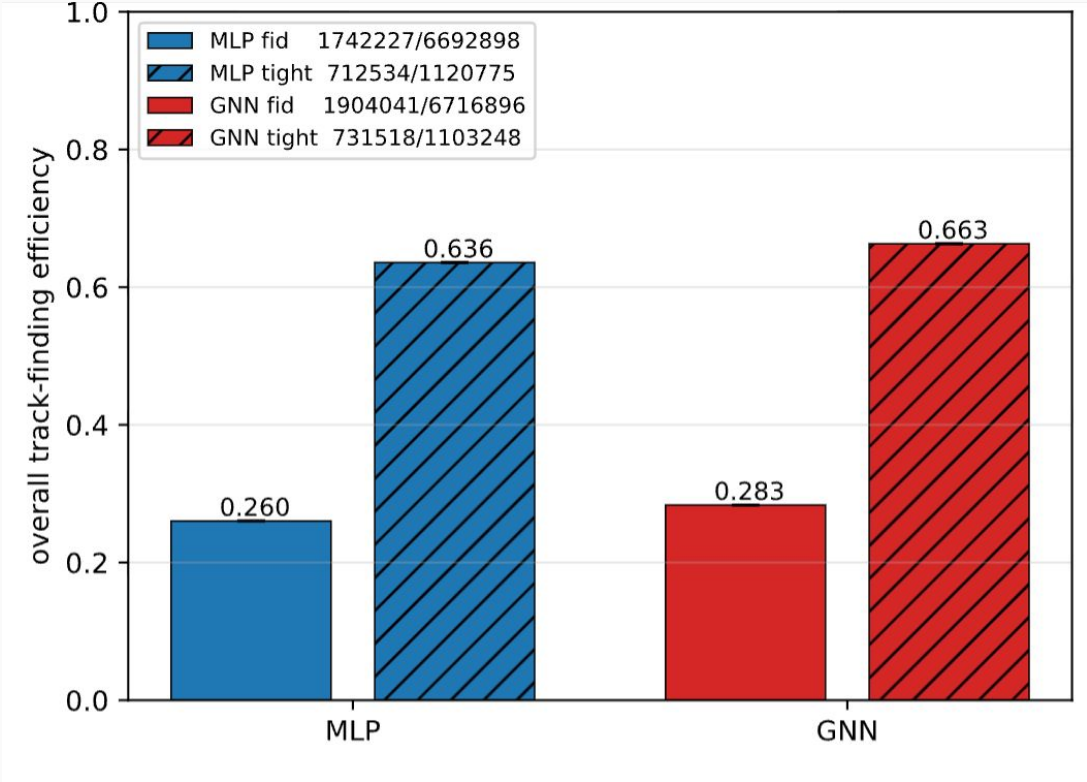


GNN vs MLP: Performance on elastic data

GNN found a bit more tracks than MLP:

Mostly because elastic tracks are easy to find and are in a environment with low background

GNN will give better result in more noisy environment like 10.6 GeV runs



Summary and Outlook

- Test and implement a GNN for track finding in the ALERT detector:
  - Train in PyTorch a GNN using simulation
  - Hyperparameter optimization (Optuna)
  - Test it on both data and simulation, and compare it to previous method (MLP)
  - GNN find 99.9% of the tracks in simulation
  - GNN find 66% of the elastic tracks in data
  - GNN is slower than MLP, but it is not significantly (0.16% of a standard reconstruction time)

- Have been implemented in COATJAVA with a yaml key to use it