Proceedings of the CTD 2022
PROC-CTD2022-21
October 13, 2022

ATLAS ITk Track Reconstruction with a GNN-based pipeline

Sylvain Caillou, Paolo Calafiura, Steven Farrell, Xiangyang Ju, Daniel Murnane, Charline Rougier, Jan Stark, Alexis Vallier

On behalf of the ATLAS collaboration

ABSTRACT

In preparation for the upcoming HL-LHC era, ATLAS is pursuing several methods to reduce the resources consumption needed to reconstruct the trajectory of charged particles (tracks) in the new all-silicon Inner Tracker (ITk). This includes the development of new algorithms suitable for massively parallel computing architecture like GPUs. Algorithms for track pattern recognition based on graph neural networks (GNNs) have emerged as a particularly promising approach. Previous work using simulated data from the TrackML challenge show high track reconstruction efficiency. In the present document we describe a first functional implementation of a GNN-based track pattern reconstruction for ITk, achieving a high GNN track reconstruction efficiency and promising fake track rate.

PRESENTED AT

Connecting the Dots Workshop (CTD 2022)
May 31 - June 2, 2022

1 Introduction and strategy

During the upcoming High Luminosity phase [1] of the Large Hadron Collider [2] (HL-LHC) the peak instantaneous luminosity of the accelerator will be increased to an unprecedented 7.5 times the nominal luminosity. The bunch crossing time (25 ns) will remain unchanged: higher instantaneous luminosity will be achieved by increasing the number of protons per bunch. This leads to an increase of the average number of inelastic proton-proton collisions per bunch crossing $\langle\mu\rangle$ (pile-up) expected to reach 200 during the Run 5. As a result, the ATLAS experiment [3] will be implementing significant upgrades to the detector and to the data acquisition system in order to cope with the increased data rates and pile-up, including construction of a new silicon-only Inner Tracker with improved granularity, ITk [4].

Even with the upgraded detector, the HL-LHC conditions will lead to a steep increase in computing resources needed to process and analyse the data. Assuming a flat budget, the expected improvements in computing hardware may not be able to provide this increased capacity, as shown in Figure 1.

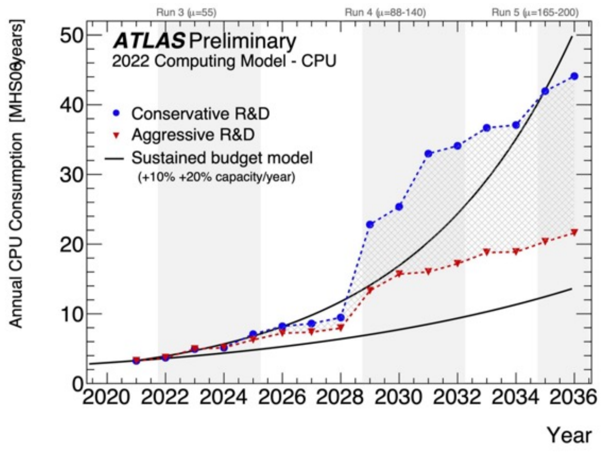


Figure 1: Projected evolution of compute usage from 2020 until 2036, under the conservative (blue) and aggressive (red) R&D scenarios. Figure taken from Ref. [5]

The offline reconstruction of ITk data represents a non-trivial fraction of computing resource needs, about 20% [5]. The CPU resources needed for event reconstruction tend to be dominated by charged particle reconstruction (”tracking”). Significant effort is therefore being invested into the reduction of computing resources needed for tracking. This includes improvements of existing algorithms and the development of new ones, based on machine learning (ML), suitable for massively parallel computing architectures like GPUs.

The authors of Ref. [6] identified the use of Graph Neural Networks (GNNs) as a promising ML solution for charged particle tracking at the HL-LHC. Since this publication, significant effort has been invested in the development of algorithms based on GNNs [7–12]. Excellent performance of GNN-based algorithms on the TrackML dataset [13] has recently been demonstrated in Refs. [8, 9].

We present a first functional implementation of a pattern recognition algorithm based on GNNs for ATLAS ITk. The method is illustrated in Figure 2.

It proceeds in three steps:

1. Graph construction: Collision events recorded by the ITk can be represented as graphs. Graph nodes represent space points created from energy deposits in the ITk (clusters). Two nodes can be connected by an edge. The existence of an edge means that the two nodes could potentially represent two successive space points on a track. Features are attached to each node (the space point coordinates) and each edge (geometric quantities calculated from the two nodes).

2. Edge labeling: A GNN assigns an edge classification score to each edge in the graph. The classification score is a value between 0 and 1. The GNN is trained to assign a high score to edges that connect two successive space points from a target particle and a low score to other edges.

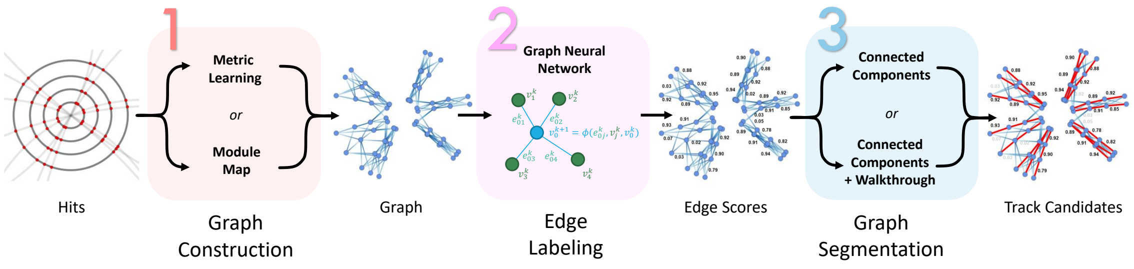


Figure 2: Schematic overview of the GNN-based track finding pipeline [14].

3. Graph segmentation: Track candidates (lists of nodes) are built from the graph based on the edge classification scores.

It is assumed that the track candidates found by this pattern reconstruction could then be run through a track fitting algorithm for track parameter estimation, and that further selections on the track candidates would be applied after fitting.

2 Simulated sample

The results presented in these proceedings are based on simulated $pp \to t\bar{t}$ events in which at least one of the top quarks decays leptonically, produced at $\sqrt{s} = 14$ TeV with a pile-up of $\langle\mu\rangle = 200$. The ITk, displayed in Figure 3, is divided into two sub-detectors:

- The pixel sub-detector: closest to the luminous region, pictured in red in Figure 3. It comprises 9416 silicon modules.
- The strip sub-detector: surrounds the pixel sub-detector, pictured in blue in Figure 3. It comprises 8944 silicon modules which are instrumented on both sides.

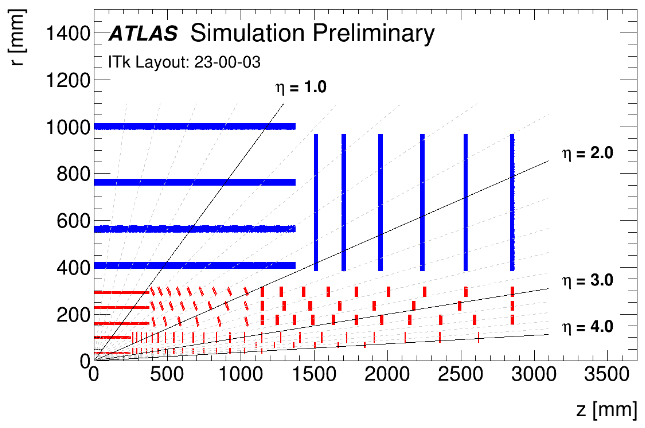


Figure 3: Schematic overview of ITk [15].

Space points are three-dimensional objects reconstructed from clusters. Pixel clusters directly translate into pixel space points. The combination of information of two strips clusters are needed to create strip space points. Figure 4 shows the creation of a particular kind of strip space point called *ghost* space point, which is caused by an overlap of strips fired by a given particle and strips on the other side of the module fired by another particle.

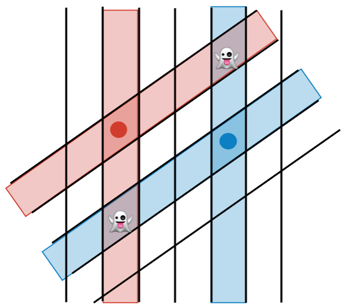


Figure 4: Schematic view of the creation of *ghost* space points created by a combinatorial ambiguity leading to two misconstructed space points.

In this study we use space points rather than clusters to reduce the numbers of nodes - and therefore edges - created within a graph.

3 Graph construction

A GNN classifier acts on data represented by a graph. In a $t\bar{t}$ event with a pile-up of $\langle\mu\rangle = 200$, $O\left(10^5\right)$ space points per event are expected. A fully connected graph of such an event would have $O\left(10^{10}\right)$ edges, most of them representing nonphysical connections. A key feature of graph construction is therefore the choice of the edge connections.

We define four categories of particles:

- Target particles: All primary particles (from the $t\bar{t}$ pair and the soft interaction) with $p_T > 1$ GeV and leaving at least 3 space points.
- Electrons: Which are excluded as a simplification.
- Non-target particles: All secondary particles, and primary particles (from the $t\bar{t}$ pair and the soft interaction) with $p_T < 1$ GeV that leave at least 3 space points.
- Non fiducial particles: are particles with a radius $r > 26$ cm or a pseudo-rapidity $|\eta| > 4$.

As we aim to reconstruct target particles, the choice of edges must guarantee the existence of edges connecting successive space points of target particles, while keeping the total number of edges low. Two kind of graph construction algorithms have been developed - the *Module Map*, a data-driven approach, and the *Metric Learning*, a machine learning approach.

3.1 The Module Map Approach

The *Module Map* is a list of connections between triplet of silicon modules used to create all possible edges in a graph. It is created by following the trajectory of target particles inside the ITk. As all space points left by a target particle are sorted by their distance from the particle vertex before being followed, a direction *inside-out* is given to each module triplets and therefore to each created edges. This direction is then used during the final step of the reconstruction 5.

To learn all possible module connections, 90000 simulated $t\bar{t}$ events are used. Connections induced by the inefficiencies of the silicon modules are also learned. Therefore, edges linking two space points of non-successive layers can also be mapped and constructed. The resulting module map comprises 1242265 triplet connections. To further reduce the number of edges, cuts are applied on geometric parameters. These cuts are automatically adjusted, separately for *each* module connection, such that no true edge coming from target particles is rejected in the training sample of 90000 $t\bar{t}$ events. The graphs created using this method have on average 2.7×10^5 nodes and 1.3×10^6 edges.

3.2 The Metric Learning Approach

In the *Metric Learning* approach, a multi-layer perceptron (MLP) is trained to embed each space point into an N-dimensional latent space, based on the space point positions and the direction and dimensions of the module containing the cluster. The MLP is trained using a hinge loss L defined as:

$$L = \begin{cases} x & \text{if true pair,} \\ \max(0, r_{emb} - x) & \text{otherwise.} \end{cases} \tag{1}$$

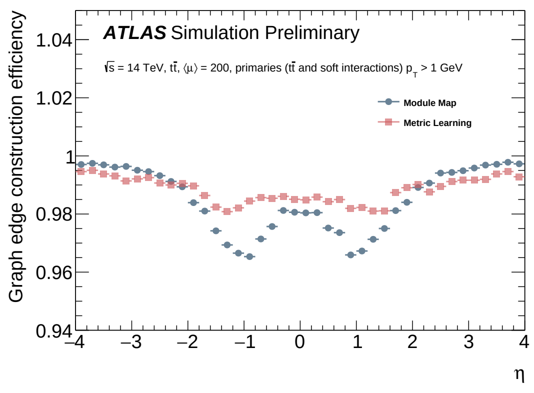
This teaches the MLP to embeds space points belonging to a given target particle to be close to each other, as measured by euclidean distance x. The embedding radius r_{emb} in the latent space is a hyperparameter of the model that can be adjusted to create graphs of various size and efficiency.

The graph is then constructed by connecting space point within the radius r_{emb} in the latent space, using a fast Fixed Radius Nearest Neighbor (FRNN) algorithm [16]. Edges are learned to be created between successive space points of the same target particle. Although FRNN creates two edges for each node pair (one in each direction), duplicate edges are chosen at random and removed for memory efficiency. Thus, while directionality is not retained by the metric learning construction, it can be recovered by following the increasing radius of the space points, and can thus be used during the final step of the reconstruction 5.

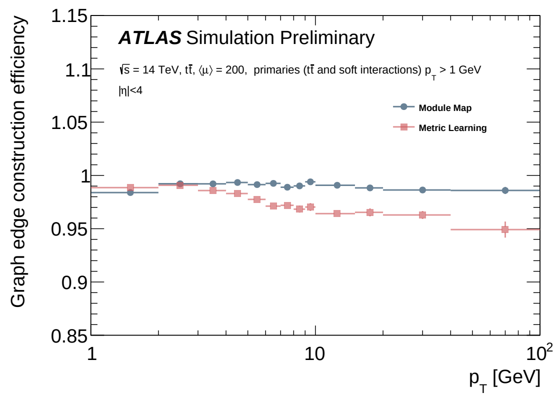
Additional filtering is done using another MLP dedicated to prune the graph. The selection requirements are adjusted to obtain graphs with the same size as the ones created using the Module Map method.

3.3 Graph edge construction efficiencies

The graph edge construction efficiency of both graph construction methods is defined as the ratio of the number of edges coming from target particles found in a graph over the total number of expected edges coming from target particles. Only the edges linking successive space points of target particles are considered. Figure 5 shows the graph edge construction efficiency as a function of η (Figure 5a) and p_T (Figure 5b).



(a) Graph edge construction efficiency vs. η



(b) Graph edge construction efficiency vs. p_T

Figure 5: Graph edge construction efficiency as a function of η and p_T using the Module Map and Metric Learning methods [14].

The drops at $|\eta| = 1$ found in the Module Map efficiency have for origin the lower efficiency found in the strip barrel, where the space point spatial longitudinal resolution is lower than in the pixel sub-detector. Building a Module Map using more events should mitigate this inefficiency. High-p_T tracks represent a small fraction of the defined target particles training set, which explains the efficiency degradation at high-p_T of the Metric Learning method.

In the following sections of this note, the Module Map method is used to build the graphs. Space point coordinates in (r, ϕ, z)-space are attached to each node as *features*. For each edge, the observables $(\Delta\eta, \Delta\phi, \Delta r, \Delta z)$ are computed from the space point coordinates of origin and destination node, and they are attached to the edge as features.

4 Deep geometric learning of track patterns

4.1 Graph Neural Network model

The GNN model used for this study has an encode-process-decode architecture. At the encode stage, an encoder transforms the features attached to each node and each edge into a $\mathcal{D}$-dimensional space latent representation. The encoder is implemented using two MLPs: one for the node features and one for the edge features. At the process stage, an interaction network [17] is applied iteratively (called a *message-passing* mechanism) $\mathcal{L}$ times,to update the latent features of the graph. This mechanism propagates information through the graph so that the GNN classify edges based on track patterns. The interaction network is divided into two steps. An edge network updates the latent features of each edge, using an MLP applied to the latent features of each adjoining node and the previous iteration's edge latent features. A node network then updates the latent features of each node using as input a concatenation of the latent features of the node with the sum of the latent feature of the incoming and outgoing edges.

At the decode stage, a decoder implemented using an MLP transforms the latent features of each edge into an edge classification score, capturing the pseudo-probability that the edge represents two successive space points of a track.

4.2 Training the Graph Neural Network model

The GNN model described in Section 4.1 ($\mathcal{L} = 8$, $\mathcal{D} = 128$, two layers in each of the MLPs) is trained using 400 graphs produced with the Module Map method. The events used to produce the graphs are also used during the Module Map creation, so that their efficiency is guaranteed to be 100%. The amsgrad [18] optimizer is used with a fixed learning rate of 5×10^{-4}. Given the size of the graphs described in 3.1, the model was trained on an Nvidia A100 with 80 GB of memory using *large model support* [19]. The full memory capacity of the GPU is used as well as 97 GB of memory on the host. A binary cross-entropy loss function is used with weight w_{edge} given to each class:

$$w_{edge} = \begin{cases} 1.0 & \text{if true edge,} \\ 0.1 & \text{if fake edge,} \\ 0.0 & \text{if masked edge.} \end{cases} \tag{2}$$

A *true edge* is defined as an edge linking two successive space points of a target particle. A *fake edge* is defined as: a) An edge linking space points of different particles; b) An edge involving an electron(s) or non fiducial particle; or c) An edge linking nonsuccessive space points of the same particle. A *masked edge* is defined as an edge linking two successive space points of non-target particle. A weight 0.0 is given to masked edges in order to not confuse the GNN by setting to false edges from non-target particles that can have similar topology to the one of the target particles.

Figure 6 shows the GNN edge classification performance after training. Background rejection rate is given as $1/\epsilon_{bkg}$ where ϵ_{bkg} is the fraction of fake edges that pass the classification requirement. Signal efficiency is defined as the number of true edges above a given classification score cut over the total number of true edges.

4.3 Edge classification performance

In this section, the performances at edge-level of the GNN edge classifier described above are obtained considering a cut on the edge classification score $s = 0.5$. All performances shown in the following are computed using 100 events withheld from those used during the GNN training. Those 100 events have been used during the module map creation so that their graphs construction efficiency is 100%.

Per-edge efficiency is defined as the number of true edges passing a given cut on the classification score over the number of total true edges. A per-edge purity is defined as the number of true edges passing the cut over the total number of edges passing the cut. Masked edges do not appear in these performance calculations.

Figure 7 shows the performance of the GNN edge classifier as a function of the pseudo-rapidity η. A drop in the central region is found in the efficiency (Figure 7a) and in the purity (Figure 7b).

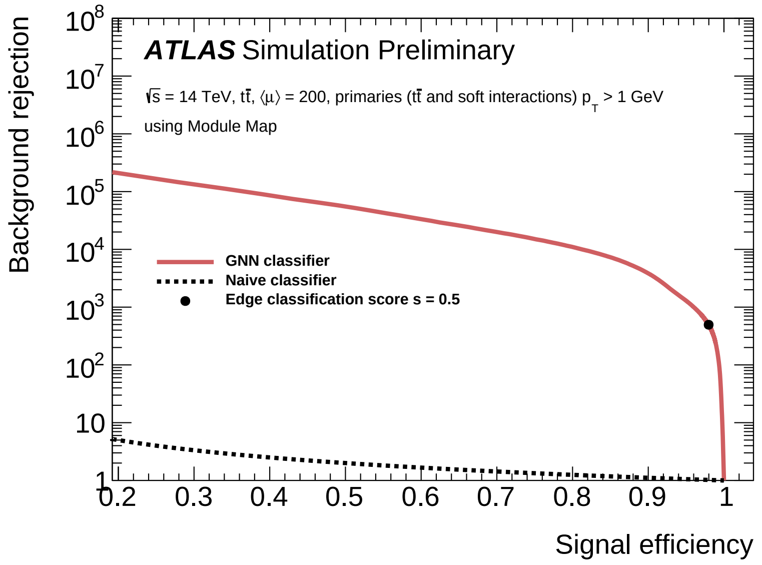
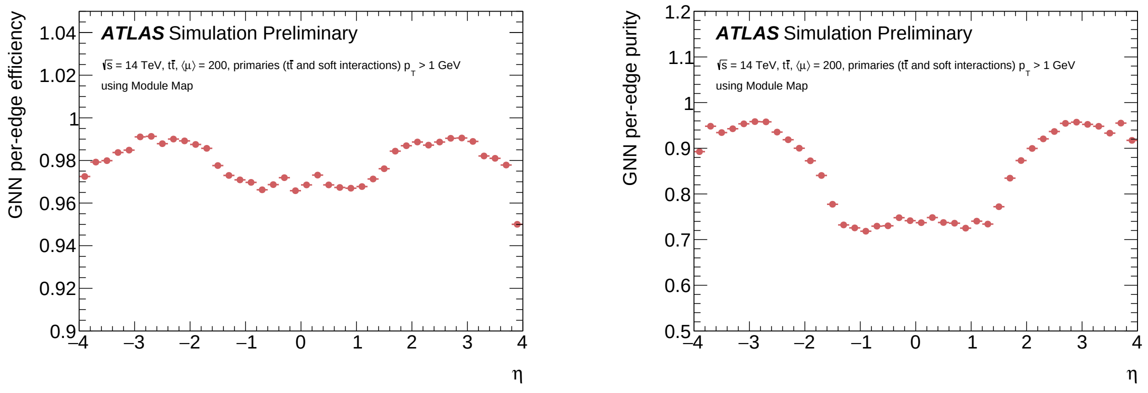


Figure 6: Graph Neural Network classification performance, based on GNN per-edge efficiency and GNN fake rate (red). The black dashed curve shows the performance of randomly classifying edges as true or false [14].



(a) GNN per-edge efficiency vs. η

(b) GNN per-edge purity vs. η

Figure 7: Performance of the GNN edge classifier as a function of the pseudo-rapidity η [14]. The asymmetric drop at η=4 could be explained by a lack of training statistics in that particular region, future trainings with larger samples will permit to conclude.

Figure 8 shows the GNN edge classifier performance as a function of the transverse momentum p_T for the full detector (red curve), for $|\eta| > 2$ (blue curve) and for $|\eta| < 2$ (green curve) . High-p_T particles tend to be produced in the central region, where the efficiency is found to be the lower, see Figure 7a.

Figure 9 shows the performances of the GNN edge classifier in the (z, r) plan. Figure 9a shows the efficiency, which is found to be high across the full detector. Figure 9b shows the purity. The drops already seen in Figure 7 are again seen, and found to be in the layers of the strip sub-detector. The lower purity and efficiency in these layers can be explained by the lower spatial space point resolution in the z direction.

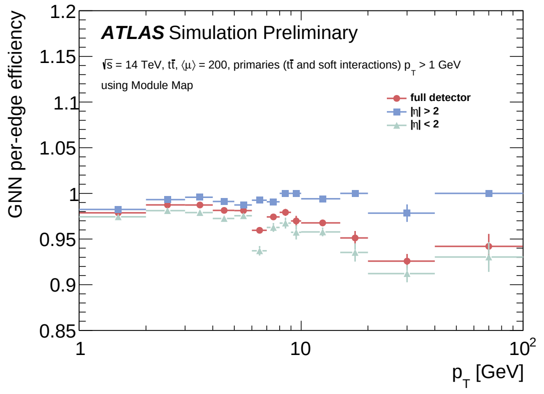
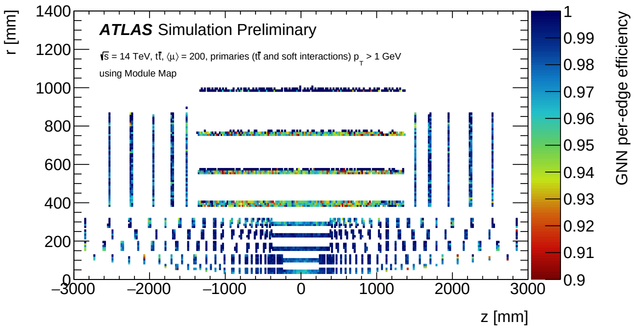
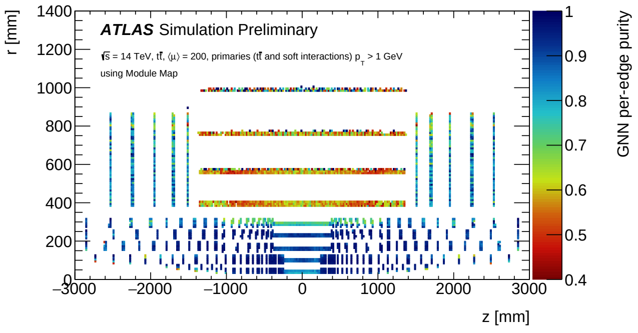


Figure 8: GNN per-edge efficiency as a function of the transverse momentum p_T [14].



(a) GNN per-edge efficiency in (z, r) plane



(b) GNN per-edge purity in (z, r) plane

Figure 9: Performance of the GNN edge classifier in the (z, r) plane. The position in (z, r) of an edge is defined by the position of its source node [14].

Figure 10 summarizes the origin of all fake edges. An edge composed of two non-successive space points from the same particle is considered fake. This means that this edge *jumps* one (or more) space point in the track of a particle. This could imply that space points will be missing in the reconstructed track if edges linking successive space points are not well-classified. In this paper, electrons and non-fiducial particles are considered as fake. They represent around 10% of the total fake edges.

A ghost is a strip space point being an accidental combination of two clusters which do not belong to the same particle. In Figure 10, fake edges linking a well-constructed node (representing a space point with both clusters coming from the same particle) with a ghost node are studied. Only ghosts having one of their two clusters belonging to the same particle as the well-constructed node are represented in this bin. If such a fake edge is included during the track building stage, this would lead to a cluster on the created track being wrongly associated.

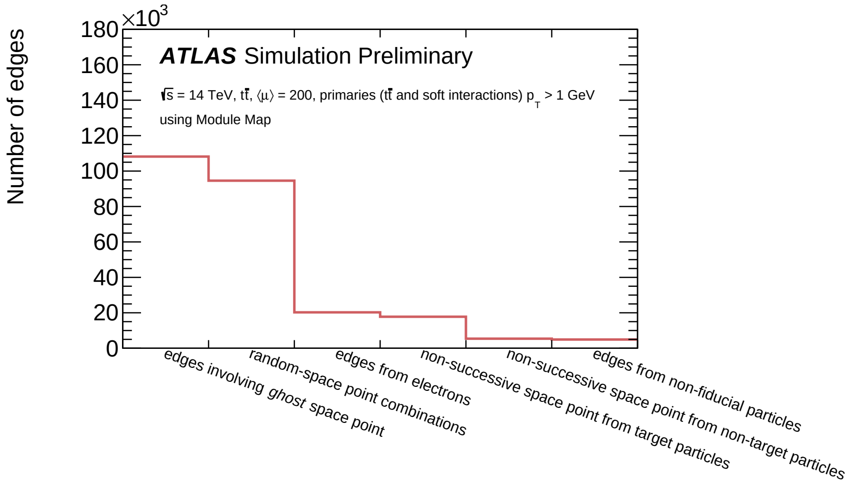


Figure 10: Origin of the fake edges [14].

5 Tracks reconstruction from Graph Neural Network output

The last stage of the pipeline presented in these proceedings is the tracks reconstruction stage based on the edge classification score and the graph topology. This stage is divided into two steps.

In the first step, a low cut of $s = 0.1$ on the edge classification score is applied to the graphs. This reduces the number of edges from $O(10^6)$ to $O(10^4)$. A set of *connected components*, are so created in the graph, where each node is linked to others via one or more paths. If a single path exist - i.e. if each node in a connected component has at most one incoming and one outgoing edge - then that connected component is directly labeled as a track candidate.

If more than one path exists, connected components are disentangled using a walk-through algorithm, called *Wrangler*. A topological sort is first applied such that nodes with no incoming edges are considered as starting points for Wrangler. Given one of these starting nodes, we choose the outgoing edge with the highest classification score, given a minimum score of $s > 0.3$. The attached node is then added to a track candidate. If other outgoing edges have a classification score $s > 0.8$, attached nodes are also used to build new track candidates. Then the procedure is repeated until no more node can be add to the track candidates. Wrangler can therefore build several track candidates from a starting node. The ambiguity is solved by keeping only the longest track candidate created.

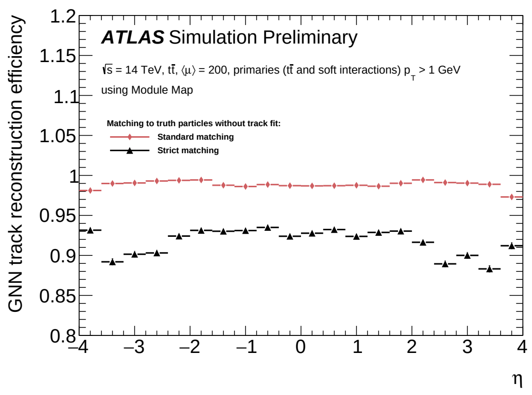
To present the GNN track reconstruction efficiency, two kinds of matching criteria are used. Note that matching is done without any track fitting. A *standard matching* is defined as a track candidate having more than 50% of its space points belonging to the same truth track. A *strict matching* is defined as a track candidate having all space points belonging to the same truth track and no space points missing from this truth track.

Figure 11 shows the performance of the GNN track reconstruction efficiency as a function of η (Figure 11a) and p_T (Figure 11b). The performances are given relative to the target particles.

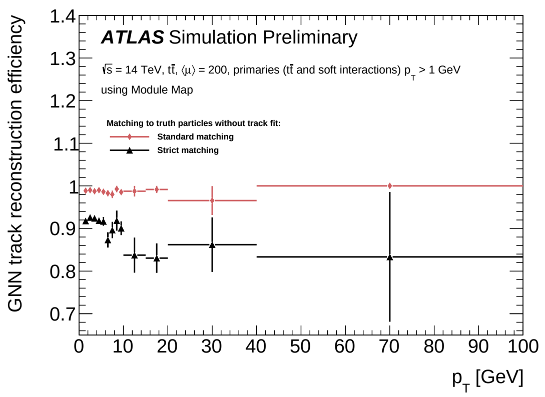
A fake track is defined as a track candidate not matched to any truth track. The ratio of fake tracks created to all target track candidates is found to be $O(10^{-3})$. Fitting the tracks is expected to reduce this amount and will be the object of a dedicated future study.

6 Conclusions and future prospects

We present the first GNN-based track reconstruction algorithm based on ATLAS ITk simulated data for HL-LHC. The GNN track reconstruction efficiency shows promising results on a realistic scenario. In the future, a high priority will be given to completing a comparison with the Kalman Filter currently used



(a) GNN track reconstruction efficiency vs. η



(b) GNN track reconstruction efficiency vs. p_T

Figure 11: GNN track reconstruction efficiency vs p_T [14].

in the ATLAS reconstruction chain. This will require an implementation of the pipeline presented in this proceedings in a tracking framework, such as ACTS and Athena. A dedicated study and tuning of the GNN architecture will be made to improve the performance degradation in the strip barrel sub-detector.

References

[1] G. Apollinari et al., *High-Luminosity Large Hadron Collider (HL-LHC): Technical Design Report V. 0.1*. CERN Yellow Reports: Monographs. CERN, Geneva, 2017. `https://cds.cern.ch/record/2284929`.

[2] Lyndon Evans and Philip Bryant, *LHC Machine*, JINST **3** (2008) S08001–S08001, `https://doi.org/10.1088/1748-0221/3/08/s08001`.

[3] ATLAS Collaboration, *The ATLAS Experiment at the CERN Large Hadron Collider*, JINST **3** (2008) S08003–S08003, `https://doi.org/10.1088/1748-0221/3/08/s08003`.

[4] ATLAS Collaboration,, *Technical Design Report for the ATLAS Inner Tracker Pixel Detector*, tech. rep., CERN, Geneva, Sep, 2017. `https://cds.cern.ch/record/2285585`.

[5] ATLAS Collaboration, *ATLAS Software and Computing HL-LHC Roadmap*, tech. rep., CERN, Geneva, Mar, 2022. `https://cds.cern.ch/record/2802918`.

[6] S. Farrell et al., *Novel deep learning methods for track reconstruction*, in *4th International Workshop Connecting The Dots 2018*. 10, 2018. `arXiv:1810.06111 [hep-ex]`.

[7] X. Ju et al., *Graph Neural Networks for Particle Reconstruction in High Energy Physics detectors*, in *33rd Annual Conference on Neural Information Processing Systems*. 3, 2020. `arXiv:2003.11603 [physics.ins-det]`.

[8] C. Biscarat et al., *Towards a realistic track reconstruction algorithm based on graph neural networks for the HL-LHC*, EPJ Web Conf. **251** (2021) 03047, `arXiv:2103.00916 [physics.ins-det]`.

[9] X. Ju et al., *Performance of a geometric deep learning pipeline for HL-LHC particle tracking*, Eur. Phys. J. C **81** (2021) 876, `arXiv:2103.06995 [physics.data-an]`.

[10] G. DeZoort et al., *Charged Particle Tracking via Edge-Classifying Interaction Networks*, Comput. Softw. Big Sci. **5** (2021) 26, `arXiv:2103.16701 [hep-ex]`.

[11] S. Thais et al., *Graph Neural Networks in Particle Physics: Implementations, Innovations, and Challenges*, `arXiv:2203.12852 [hep-ex]`.

[12] S. Rukh Qasim et al., *End-to-end multi-particle reconstruction in high occupancy imaging calorimeters with graph neural networks*, `arXiv:2204.01681 [physics.ins-det]`.

[13] M. Kiehn et al., *The TrackML high-energy physics tracking challenge on Kaggle*, EPJ Web Conf. **214** (2019) 06037.

[14] ATLAS Collaboration, *Track finding performance plots for a Graph Neural Network pipeline on ATLAS ITk Simulated Data*, 2022. `https://atlas.web.cern.ch/Atlas/GROUPS/PHYSICS/PLOTS/IDTR-2022-01/`.

[15] ATLAS Collaboration,, *Expected tracking and related performance with the updated ATLAS Inner Tracker layout at the High-Luminosity LHC*, tech. rep., CERN, Geneva, Jul, 2021. `https://cds.cern.ch/record/2776651`.

[16] Lixin Xue, *Fixed Radius Nearest Neighbors Search*, `https://github.com/lxxue/FRNN`.

[17] P. W. Battaglia et al., *Interaction Networks for Learning about Objects, Relations and Physics*, `arXiv:1612.00222 [cs.AI]`.

[18] S. J. Reddi et al., *On the Convergence of Adam and Beyond*, `arXiv:1904.09237 [cs.LG]`.

[19] D. L. Tung et al., *TFLMS: Large Model Support in TensorFlow by Graph Rewriting*, CoRR (2018), `arXiv:1807.02037`.