MACHINE LEARNING
Science and Technology

PURPOSE-LED PUBLISHING™ Part of the MACHINE LEARNING Series

PAPER • OPEN ACCESS

Geometric GNNs for charged particle tracking at GlueX

To cite this article: Ahmed Hossam Mohammed *et al* 2025 *Mach. Learn.: Sci. Technol.* **6** 035049

IOP Publishing Mach. Learn.: Sci. Technol. 6 (2025) 035049 https://doi.org/10.1088/2632-2153/ae02df

MACHINE LEARNING
Science and Technology

OPEN ACCESS

RECEIVED
13 May 2025

REVISED
3 August 2025

ACCEPTED FOR PUBLICATION
3 September 2025

PUBLISHED
12 September 2025

PAPER

Geometric GNNs for charged particle tracking at GlueX

Ahmed Hossam Mohammed[1,*], **Kishansingh Rajput**[1,2], **Simon Taylor**[1], **Denis Furletov**[3], **Sergey Furletov**[1] and **Malachi Schram**[1,4]

[1] Thomas Jefferson National Accelerator Facility, Newport News, VA 23606, United States of America
[2] Department of Computer Science, University of Houston, Houston, TX 77204, United States of America
[3] Department of Physics, William & Mary, Williamsburg, VA 23185, United States of America
[4] Computer Science Department, Old Dominion University, Norfolk, VA 23529, United States of America
[*] Author to whom any correspondence should be addressed.

E-mail: ahmedm@jlab.org, kishan@jlab.org, staylor@jlab.org, dfurletov@wm.edu and furletov@jlab.org

Keywords: particle tracking, track finding, graph building, edge classification, graph neural networks, batched GNN pipeline

Abstract

Nuclear physics experiments are aimed at uncovering the fundamental building blocks of matter. The experiments involve high-energy collisions that produce complex events with many particle trajectories. Tracking charged particles resulting from collisions in the presence of a strong magnetic field is critical to enable the reconstruction of particle trajectories and precise determination of interactions. It is traditionally achieved through combinatorial approaches that scale worse than linearly as the number of hits grows. Since particle hit data naturally form a point cloud and can be structured as graphs, graph neural networks (GNNs) emerge as an intuitive and effective choice for this task. In this study, we evaluate the GNN model for track finding on the data from the GlueX experiment at Jefferson Lab. We use simulation data to train the model and test on both simulation and real GlueX measurements. We demonstrate that GNN-based track finding outperforms the currently used traditional method at GlueX in terms of segment-based efficiency at a fixed purity while providing faster inferences. We show that the GNN model can achieve significant speedup by processing multiple events in batches, which exploits the parallel computation capability of graphical processing units (GPUs). Finally, we compare the GNN implementation on GPU and field-programmable gate array and describe the trade-off.

1. Introduction

Nuclear Physics (NP) experiments aim to uncover the fundamental building blocks of matter and improve our understanding of the Universe. Many of these experiments involve high-energy collisions that can produce multiple charged particles in the presence of a magnetic field inside a charged particle detector. These charged particle detectors are composed of layers that record what we refer to as hits that register ionization energy deposits within the layers. These hits represent the coordinates at which the particles pass through the detector.

Reconstructing the charged particle tracks constitutes an important part in the analysis of high-energy physics experiments. On a high level, track reconstruction is generally divided into two subtasks, namely track finding and track fitting. In the track finding stage, hits are grouped into subsets, each deemed to belong to one of the particles forming a track. In simple terms, this subtask can be viewed as connecting the right dots to each other. For each of the subsets provided by the track finder, the track fitting algorithm estimates a set of parameters that uniquely describe the state of the particle [1].

Track finding helps in understanding the propagation patterns of charged particles subjected to a strong magnetic field. The charged particle curves in the presence of a magnetic field inversely proportional to their momentum (which is unknown at this stage); this makes it challenging, as different particles curve by different proportions making helix movement in the three-dimensional space [2]. In addition, the tracking data has noise due to background and secondary particles that may cross the particle trajectories confusing the algorithm.

The existing algorithms used for track finding are combinatorial in nature, which do not scale with the increase in multiplicity, slowing down the entire reconstruction pipeline [3]. The presence of particles with similar trajectories in addition to noise in the measurements motivates the exploration of new methods to improve performance.

In this study, we compare the performance of a graph neural network (GNN) model on track finding with a traditional method used at GlueX on forward drift chamber (FDC) data. The GlueX spectrometer [4] is composed of a large solenoid magnet that houses the FDC detectors. We demonstrate that the GNN-based track finding approach provides a 7.5% improvement in segment efficiency at a fixed purity value compared to the traditional method while maintaining a significantly lower inference time (81.6% reduction) by processing multiple events in batches, thereby leveraging the parallel computational capability of modern graphical processing units (GPUs). In addition, we present the deployment of the models on a field-programmable gate array (FPGA) and demonstrate the additional speedups that can be achieved, albeit with a slight reduction in performance.

The rest of the paper is structured as follows: In section 2, we describe some of the recent relevant studies found in the literature. Section 3 presents the detector geometry and describes the working mechanism of the traditional track finding method. In section 4, we describe our data pre-processing approach that produces graphs for GNN training and evaluation. Section 5 introduces our modeling approach and section 6 presents the performance and timing results of the proposed approach on both GPU and FPGA devices and compares them to the traditional method. Finally, we conclude with future outlook in section 7.

2. Previous work

Due to the helical propagation pattern of particles in a solenoidal magnetic field, the hit projections of a given particle track on the xy-plane would roughly form a circle passing through the origin of the form $(x-a)^2+(y-b)^2=R^2=a^2+b^2$. Conformal transformation was early adopted [5] to transform circular xy-plane projections into linear uv-plane projections of the form $2au+2bv=1$ where $u=x/(x^2+y^2)$ and $v=y/(x^2+y^2)$. The straight lines are then used for pattern recognition to group points belonging to a given line. The strong assumption that the circle passes through the origin–imposed by equating R^2 to a^2+b^2–is remedied by allowing a small difference between the two terms, which corresponds to a parabola fit with a small curvature in the uv-plane. Multiple particle scattering can be another source of deviation from the circular path in the xy-plane. The cellular automaton (CA) algorithm [6] used for track finding in conformal tracking consists of building and extending cells (defined as segments connecting two hits). In [7], the two stages (building and extension) run recursively as the final track finding strategy. Another used approach is the Hough transform [8], where each hit corresponds to a plane in the parameter space (Hough space). Hits are grouped on the basis of their intersection in the Hough space. In practice, planes in the Hough space do not intersect at a single point. Therefore, the space is divided into bins and points are grouped together if their planes cross the same bin [9]. An overview of other traditional methods such as artificial retina and Legendre transform can be found in [10].

Recently, machine learning (ML) has been introduced to address the track finding task. recurrent neural networks (RNNs), a class of ML architectures designed for time series prediction, was adopted in [11] for track finding where each track is modeled as a sequence of hits. The task is thus formulated as a regression problem in which the RNN iteratively attempts to estimate the coordinates of the next hit given the current and previous hit coordinates. This method yields poor first guesses due to its inability to estimate the track trajectories. The detector data naturally imposes itself as a graph structure with the event hits represented as graph nodes. The coordinates of each hit are represented as node features in the graph. To this end, the same study introduced GNNs to tackle the problem. On a high level, GNNs work by iteratively passing messages across neighboring nodes in the graph. This allows each node to update its representation based on its neighbors and itself. GNNs were first introduced in [12, 13]. Later works demonstrated the power of several variants of GNNs with different underlying message passing mechanisms such as graph convolutional network (GCN) [14], GraphSAGE [15], and graph attention network (GAT) [16]. Two GNN flavors were introduced to address the track finding problem [11]. The *first* flavor addresses the track finding task as a node classification problem. For a given training target track, four surrounding hits on the adjacent layer are connected. Seven graph iterations were executed in the GNN before a sigmoid activation was applied for each node to detect whether it belongs to the target track. Despite its novelty, this method requires a partially labeled graph (i.e. seeds) which limits its applicability. A more natural approach was adopted in the *second* GNN flavor that attempts to classify edges built with a predefined set of geometric constraints between adjacent detector layers. Hence, only the edges connecting hits from the same particle would be predicted to be true by the GNN.

Other studies have also demonstrated the suitability of edge-classifiers for particle tracking applications [17, 18]. In [19], the same approach was followed and the constraints with which graphs were built were extensively studied. Previous studies implemented similar GNN architectures on FPGAs, which require special management–such as subdividing the event graph into multiple sectors–due to memory limitations [20, 21].

3. Background

In this study, we train an edge-classifying GNN on simulated data for the GlueX detector. The simulation of the detector response is based on the GEANT4 software package [22]. Figure 1 illustrates the geometrical structure of the GlueX detector. A beam of high-energy photons impinges on a liquid hydrogen target in the bore of a solenoid magnet enclosing detectors for charged and neutral particle detection that form the core of the GlueX detector. After interactions of the beam with the target, many charged particles moving in the forward direction are reconstructed using hits in the FDC detectors. Hits in the FDC are separated into four packages, each containing six wire planes.

3.1. Traditional method

The traditional method used in the GlueX data analysis for track finding starts by looking for track segments in each of the four FDC packages. Starting with the most upstream plane in a given package containing hits, the algorithm proceeds from plane to plane looking for nearest neighbors subject to a 2 cm radial proximity threshold. If three or more hits are associated together, a preliminary helical fit is performed with the assumption that the particles emerged from the center of the target. These associated hits and the fitted helical parameters form the track segments. The results of the fits are used to project from one package to the next starting from the most upstream package containing track segments. The track segments that belong to a common track lie close to a circle in a plane perpendicular to the beam line. If the projected position is within a certain distance from the position of the segment in the projected package, the segments are linked together. The match threshold depends on the radius r_c of each circle and the separation distance d according to $d^2 < 1000/r_c$ subject to a minimum threshold value of 5 cm^2 and a maximum threshold value of 25 cm^2. If this match criterion is not met, the algorithm attempts to match the centers of the circles (x_{c1}, y_{c1}) and (x_{c2}, y_{c2}) of the two segments. The requirement is:

$$(x_{c1} - x_{c2})^2 + (y_{c1} - y_{c2})^2 < 25\,\text{cm}^2. \tag{1}$$

Sometimes, this simple approach does not successfully link all the segments on a given track together. If segments in two adjacent packages are linked together and there are unmatched isolated segments or unmatched linked pairs of segments remaining, the combined set of hits is used to redo the helical fit and the new fitted results are used to project to segments downstream of the second package in the pair and the match criteria are applied again. Each set of linked segments forms a track candidate.

3.2. ML pipeline overview

The simple approach described in the previous subsection provides a reasonable level of efficiency and purity in track finding but there is still scope to improve the performance with ML methods such as GNN. Figure 2 presents an overview of the pipeline workflow presented in this paper. The details of the two main components of the pipeline, namely the graph-builder and the GNN model, are discussed in the following two sections.

4. Data pre-processing (Graph Building)

The data acquired from GlueX FDC (depicted in figure 1) for each event is presented as a collection of hits. Each hit has three-dimensional spatial coordinates (x, y, z) at which the hit is recorded. In the simulation data, the identity of the particle associated with each hit is known, which enables assigning ground-truth label for each edge. An edge is given the label 'true' if both associated hits it connects belong to the same particle (i.e. constituting a particle track segment), and 'false' otherwise. As such, a graph can be constructed by creating edges between hits on adjacent layers.

Creating edges between all possible adjacent hits would cause a high imbalance between true and false edges in addition to demanding higher computational resources to process these dense graphs. The solution is to create an edge connecting two hits, a and b, only if it meets constraints based on the detector geometry. For our case, the constraints are as follows:

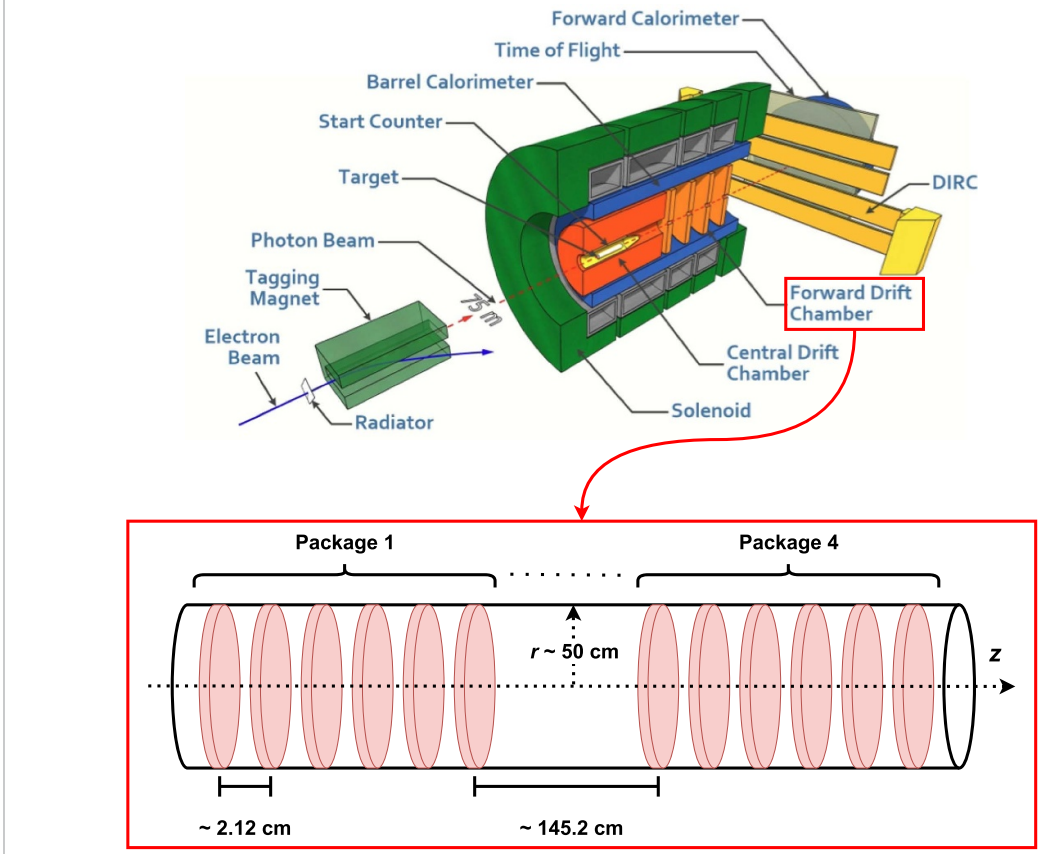


Figure 1. A 3D figure illustrating the geometry of the GlueX detector. It shows forward drift chamber (FDC) that is composed of four packages, each including six disc-shaped layers. Each of the layers has detector wires on which the (x, y, z) coordinates of the hits are recorded through ionization energy deposits. Particles propagate along the positive z-axis that is perpendicular to the layers.

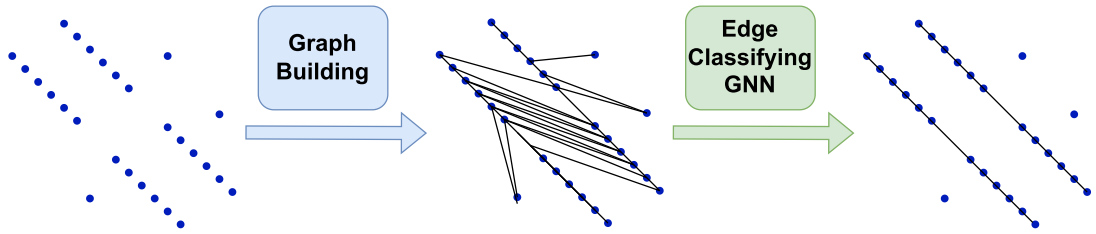
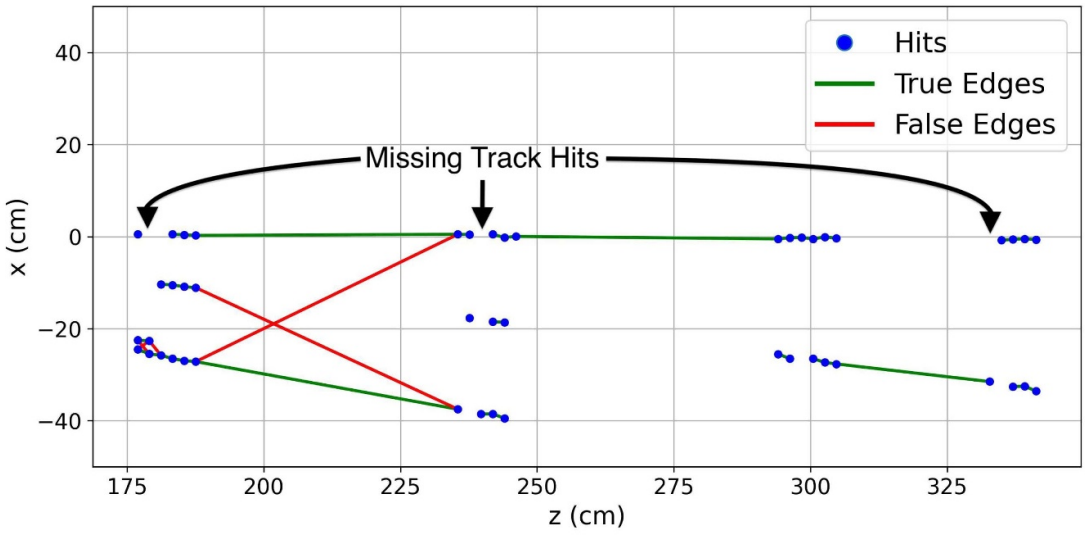


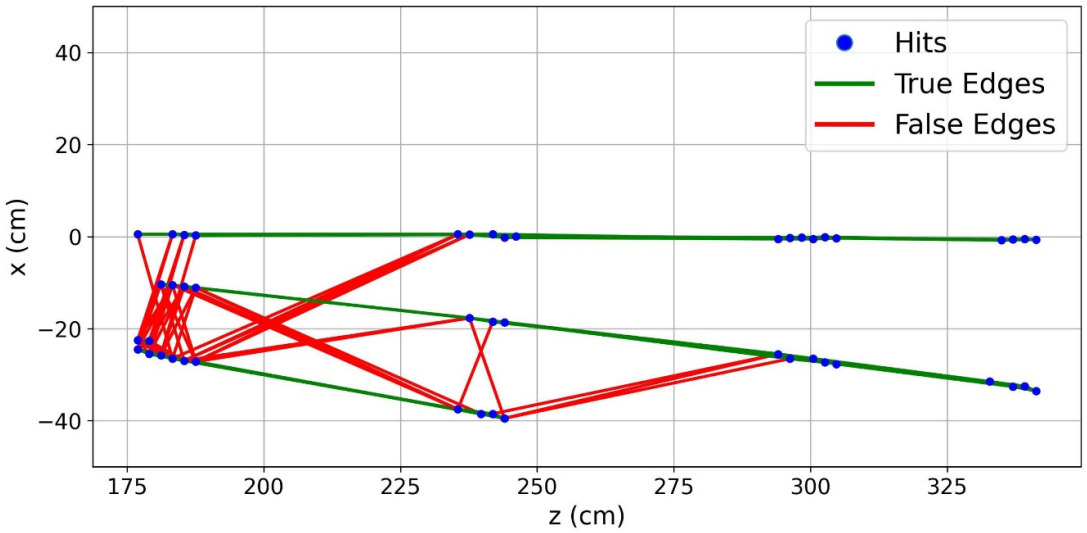
Figure 2. ML pipeline summary: the detector yields hit coordinates information. The graph-builder constructs graph from the raw hits by introducing edges between some of the hits based on graph building constraints. Finally, the edge-classifier GNN should discard false edges only (i.e. edges not corresponding to track segments).

- $d_{xy}^{(a,b)} < 34.4$ cm, where $d_{xy}^{(a,b)}$ is the Euclidean distance between the two hits in the xy-plane. The z dimension is not included due to the non-uniformity of the distance between the detector layers as shown in figure 1. Layers belonging to different groups have much larger separation compared to layers within the same group. Additionally, the inclusion of skip edges that will be introduced later in this section introduces non-uniformity in the z-distance even among layers within the same group.
- $d_{xy}^{(a,b)}/dz^{(a,b)} < 5.4$ where $dz^{(a,b)}$ refers to the physical distance between the layers on which the two hits are recorded. While the previous constraint limits the absolute Euclidean distance in the xy-plane, this constraint further limits the xy-plane distance for the hits that are close in z dimension.
- $|d\phi^{(a,b)}| < 2.3$ rad where $d\phi^{(a,b)}$ is the difference in the azimuth angle of the two hits (i.e. $\phi_b - \phi_a$).

The selection of the cut-off points for the various constraints, as listed above, is based on an empirical analysis with a multi-objective genetic algorithm [23] to optimize both efficiency and purity on realistic simulation data. The details of this study are described in appendix A. Each built event graph, g, has node features $g.X \in \mathbb{R}^{|\text{hits}| \times 3}$ where 3 refers to the used cylindrical coordinates (r, ϕ, z), and adjacency list of edges



(a) Event graph built without skip edges (i.e., $skip_{max} = 0$)



(b) Same event graph built with $skip_{max} = 2$

Figure 3. Graphs built with no skip edges (i.e. $skip_{max} = 0$) are more susceptible to the problem of missing hits as shown in part (a). This problem is resolved by introducing edges between hits on non-consecutive layers with a maximum separation of $1 + skip_{max}$. Part (b) shows the same event built with a $skip_{max}$ of 2 that does not suffer from the gap patterns which makes detecting the full track possible (i.e. higher efficiency).

$g.E \in \mathbb{R}^{|\text{edges}| \times 2}$. |hits| and |edges| refer to the cardinality of the sets of event hits and edges, respectively. If g is constructed from a simulated event, it would additionally have $g.label \in \mathbb{R}^{|\text{edges}|}$. The cylindrical coordinates of each hit are normalized to the [0-1] range before getting processed by the GNN.

In GlueX, limited detector efficiency causes some tracks to have missing hits. This causes gap patterns along those tracks as demonstrated in figure 3(a) where the simulated event graph is built by considering only edges connecting hits on consecutive layers. Note that the role of the GNN is limited to classifying existing edges in the input graph (i.e. no additional edges are introduced). To resolve this problem, we introduce the notion of skip edges (or residual edges) that allows connecting hits on non-consecutive detector layers. $skip_{max}$ is a parameter that defines the maximum number of intermediate layers an edge can cross (i.e. $0 < layer_b - layer_a \leqslant 1 + skip_{max}$ where $layer_i$ refers to the order of the detector layer on which hit i is located). Figure 3(b) shows the same event as figure 3(a) but built with $skip_{max}$ value of 2. This comes with the cost of building more dense graphs in addition to introducing redundant long edges by connecting two hits directly with an edge that are already connected with multiple shorter edges. This can be addressed in a post-processing step after the ML algorithm rejects the false edges.

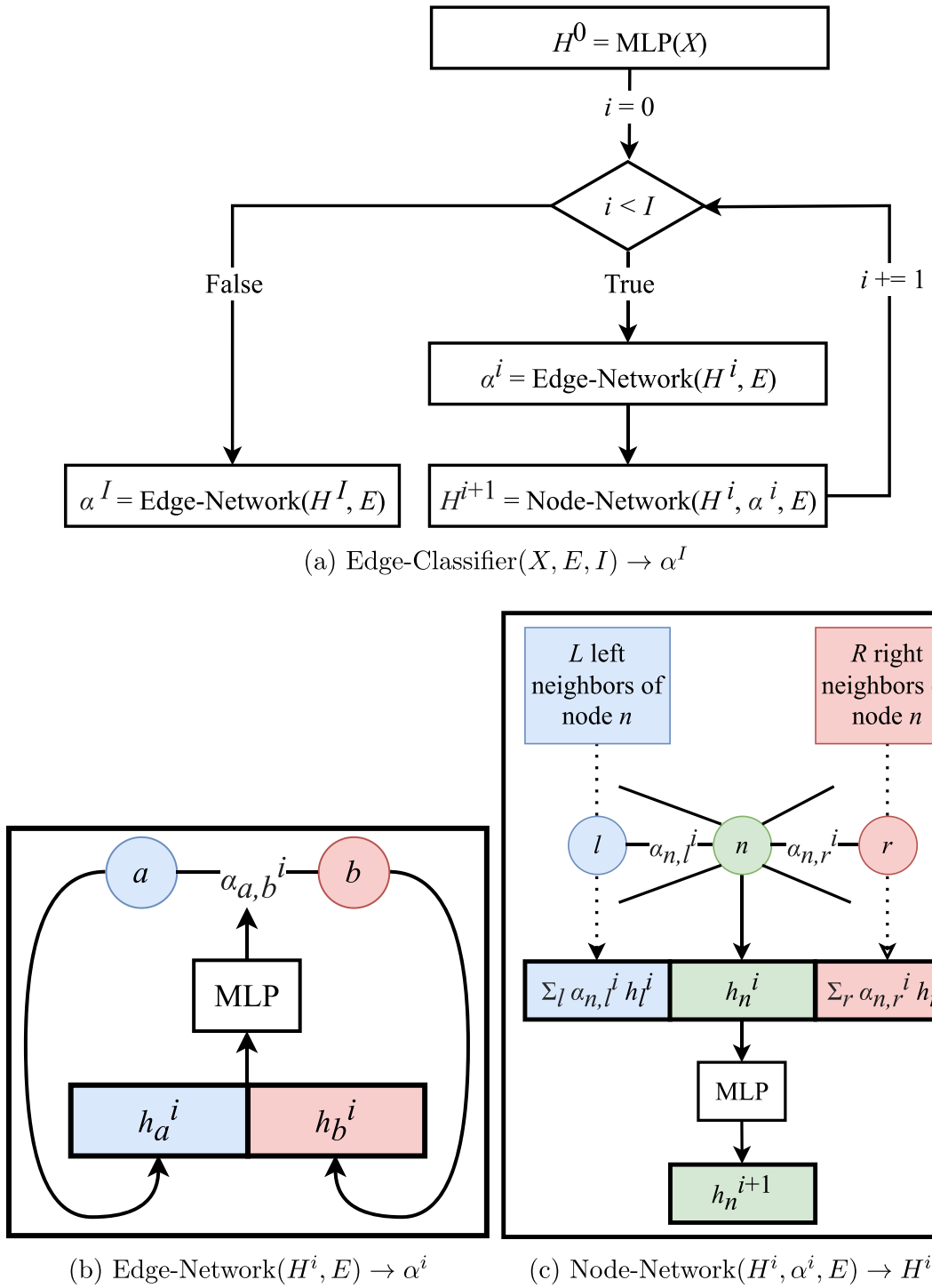


Figure 4. Edge-classifier GNN takes a graph (with X and E representing normalized coordinates of graph hits and adjacency list, respectively) and returns the probability of each edge being true after performing I message passing iterations.

5. Edge-classifier GNN

The edge-classifier GNN architecture depicted in figure 4 takes the built event graphs as input and is trained in a supervised fashion to output the correct label for each edge. The classifier starts by expanding the dimensionality of the normalized features using a multilayer perceptron (MLP) and then alternates between edge-network and node-network (described in the following subsections) I times, where I represents the number of message passing iterations carried out by the GNN. Table 1 presents the parameters used for training the model and Algorithm 1 outlines the detailed implementation of the GNN edge-classifier. Note that before the edge-network network is applied, the current node representation, H, is concatenated with the input node features (i.e. $H \leftarrow [H; X]$) to preserve the initial feature information.

Table 1. Model parameters.

Model Parameter	Value
Number of message passing iterations (I)	1
MLP hidden layers width (W)	128
MLP number of hidden layers (D)	1
Default activation	Rectified linear unit (ReLU) [24]
Edge-network output activation	Sigmoid
Dropout probability	0.05
Loss	Binary cross-entropy
Optimizer	Adam [25]
Learning rate	0.001

Algorithm 1. Edge-classifier forward method.

Require: Graph g with attributes: $g.X \in \mathbb{R}^{|\text{hits}|\times 3}$ & $g.E \in \mathbb{R}^{|\text{edges}|\times 2}$

$H^0 \leftarrow$ Edge-Classifier.MLP($g.X$) $\triangleright H^0 \in \mathbb{R}^{|\text{hits}|\times W}$

$H^0 \leftarrow$ Concatenate$[H^0; g.X]$ $\triangleright H^0 \in \mathbb{R}^{|\text{hits}|\times (W+3)}$

for $i = 0;\ i < I;\ i++$; **do**

  $\alpha^i \leftarrow$ Edge-Network($H^i, g.E$) $\triangleright \alpha^i \in \mathbb{R}^{|\text{edges}|}$

  $H^{i+1} \leftarrow$ Node-Network($H^i, \alpha^i, g.E$) $\triangleright H^{i+1} \in \mathbb{R}^{|\text{hits}|\times W}$

  $H^{i+1} \leftarrow$ Concatenate$[H^{i+1}; g.X]$ $\triangleright H^{i+1} \in \mathbb{R}^{|\text{hits}|\times (W+3)}$

end for

$\alpha^I \leftarrow$ Edge-Network($H^I, g.E$) $\triangleright \alpha^I \in \mathbb{R}^{|\text{edges}|}$

return α^I

Algorithm 2. Edge-network forward method.

Require: $H \in \mathbb{R}^{|\text{hits}|\times (W+3)}$, $E \in \mathbb{R}^{|\text{edges}|\times 2}$

$l \leftarrow E[:, 0]$ $\triangleright l \in \mathbb{R}^{|\text{edges}|}$

$r \leftarrow E[:, 1]$ $\triangleright r \in \mathbb{R}^{|\text{edges}|}$

$H_l \leftarrow H[l, :]$ $\triangleright H_l \in \mathbb{R}^{|\text{edges}|\times (W+3)}$

$H_r \leftarrow H[r, :]$ $\triangleright H_r \in \mathbb{R}^{|\text{edges}|\times (W+3)}$

$\alpha \leftarrow$ Concatenate$[H_l; H_r]$ $\triangleright \alpha \in \mathbb{R}^{|\text{edges}|\times [2\times (W+3)]}$

$\alpha \leftarrow$ Edge-Network.MLP(α) $\triangleright \alpha \in \mathbb{R}^{|\text{edges}|}$

return α

5.1. Edge-network

For each edge in the adjacency list, E, the edge-network concatenates the embeddings of the two associated hits, a and b, and assigns it a scalar weight in the [0-1] range, denoted by $\alpha^i_{a,b}$ where i is the index of the message passing iteration. This weight is later used to scale the messages between the two nodes in the node-network. In addition, it represents the probability of the edge being true at the end of the classifier. Hence, a sigmoid activation function is used at the end of the MLP in the edge-network. Intuitively, the goal is to scale down messages passed between hit nodes that belong to different particles. Algorithm 2 shows the detailed implementation of the edge-network presented graphically in figure 4(b).

5.2. Node-network

This block is responsible for updating the node embeddings after performing message passing across neighboring nodes. As shown in figure 4(c), for a given hit n, three embeddings are concatenated: ***(i)*** Aggregated scaled embeddings from left neighbor hits located on detector layers $< layer_n$, ***(ii)*** Embedding of hit n itself, and ***(iii)*** Aggregated scaled embeddings from right neighbor hits on detector layers $> layer_n$. This is then forwarded to the MLP of the node-network. Algorithm 3 shows the detailed implementation of the node-network. Note that the aggregation of the embeddings is performed via a gather reduction operation using PyTorch's index_add function.

Algorithm 3. Node-network forward method.

Require: $H \in \mathbb{R}^{|\text{hits}| \times (W+3)}$, $\alpha \in \mathbb{R}^{|\text{edges}|}$, $E \in \mathbb{R}^{|\text{edges}| \times 2}$

$l \leftarrow E[:, 0]$ ▷$l \in \mathbb{R}^{|\text{edges}|}$

$r \leftarrow E[:, 1]$ ▷$r \in \mathbb{R}^{|\text{edges}|}$

$H_l \leftarrow \alpha \times H[l, :]$ ▷element-wise multiplication: $H_l \in \mathbb{R}^{|\text{edges}| \times (W+3)}$

$H_r \leftarrow \alpha \times H[r, :]$ ▷element-wise multiplication: $H_r \in \mathbb{R}^{|\text{edges}| \times (W+3)}$

$L \leftarrow$ index_add(destination = zeros_like(H), index = r, source = H_l)

$R \leftarrow$ index_add(destination = zeros_like(H), index = l, source = H_r)

$H \leftarrow$ Concatenate[L; H; R] ▷$H \in \mathbb{R}^{|\text{hits}| \times [3 \times (W+3)]}$

$H \leftarrow$ Node-Network.MLP(H) ▷$H \in \mathbb{R}^{|\text{hits}| \times W}$

return H

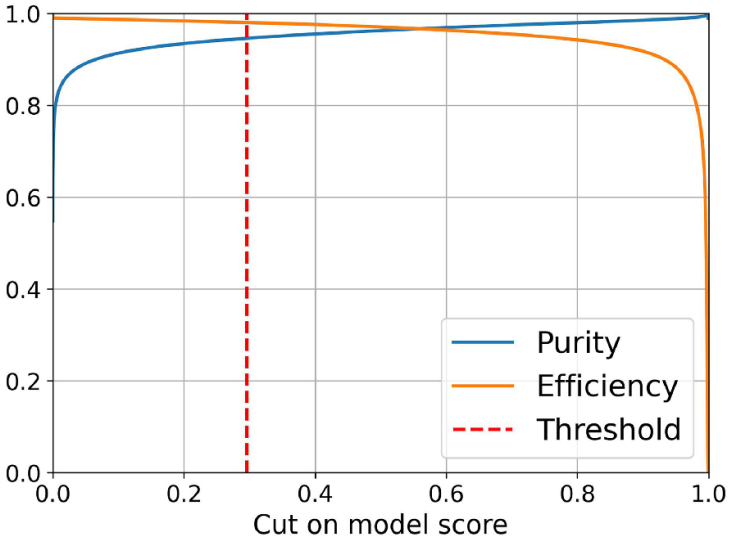


Figure 5. Overall efficiency & purity of the ML pipeline evaluated on simulated test events. At a fixed purity level of 0.9462, the pipeline achieves efficiency of 0.9806 compared to 0.9119 scored by the traditional method (7.5% increase) at a threshold ≈ 0.3.

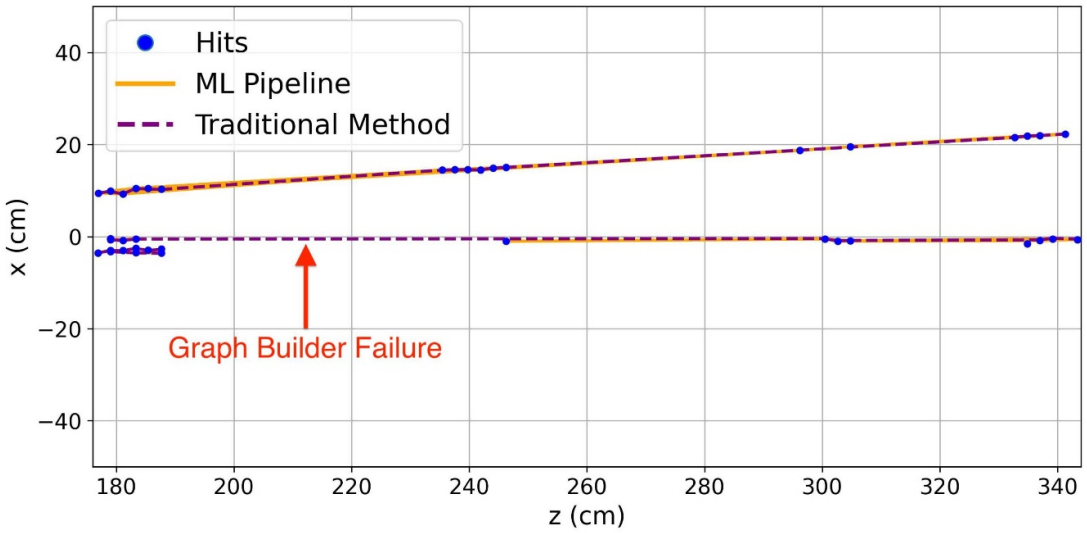
6. Results & discussion

6.1. Performance analysis

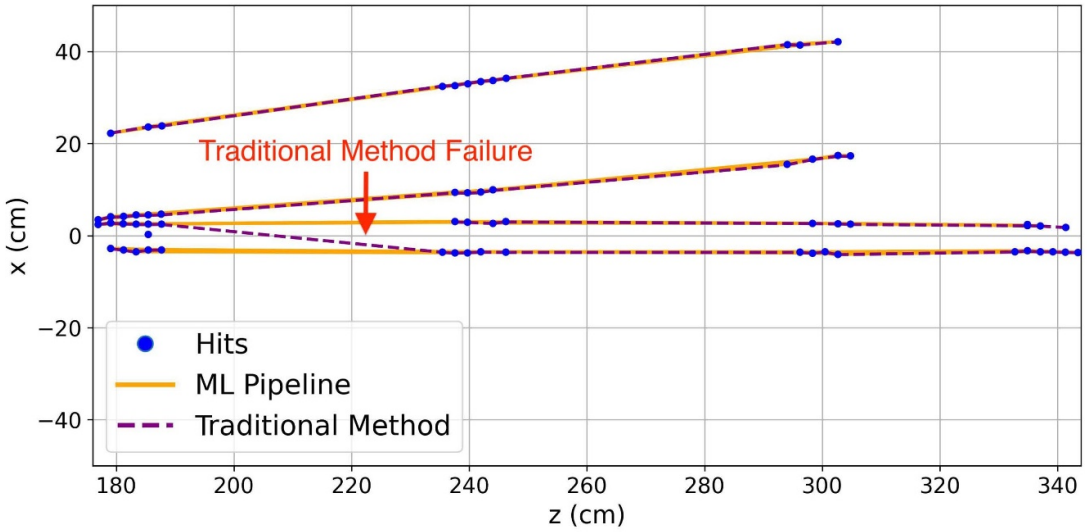
The simulated dataset comprises approximately 91 K events, that are randomly partitioned into training, validation, and testing sets in approximate proportions of 70%, 15%, and 15%, respectively. We begin by evaluating the traditional method on the testing simulated events in terms of efficiency and purity. Efficiency is defined as the fraction of true edges retained by the system (e.g. traditional method, graph-builder, or ML pipeline), whereas purity refers to the proportion of correct predictions among all detected edges. The efficiency and purity presented in this study are calculated on the segment (edge) level. The reported efficiency and purity of the traditional method on the test dataset are 0.9119 and 0.9462, respectively. Unlike the ML pipeline that outputs the probability score for each edge, this method produces edges as part of predicted tracks that can be treated as binary labels.

The efficiency and purity of the graph-builder with the constraints listed in section 4 are 0.9905 and 0.5473, respectively. Figure 5 shows the efficiency and purity of the pipeline (i.e. Graph-Builder + Edge-Classifier) as a function of the probability score produced by the model. To compare the pipeline with the traditional method, we applied a threshold on the output that yields the same purity value (0.9462). The corresponding efficiency is 0.9806 representing a 7.5% increase over the 0.9119 efficiency of the traditional method.

Designing a track finding system that generates minimal false positives is crucial to avoid error from propagating to the fitting stage. We therefore compare the traditional method and the pipeline in terms of



(a) Graph-Builder with $skip_{max}$ of 3 does not create edge with 7 consecutive missing hits. The pipeline may not connect distant hits that have number of consecutive missing hits more than $skip_{max}$ parameter.



(b) Traditional method mistakenly merging two different tracks

Figure 6. Real GlueX examples.

purity at a fixed efficiency. In this case, higher purity corresponds to a system that generates less false positives. We ignored edges involving background noise to focus the attention on edges that mistakenly connect distinct tracks. The purity value of the traditional method and the GNN pipeline are 0.9863 and 0.9933, respectively representing an increase of 0.7%. When no edges are ignored (i.e. background hits are considered), the purity improvement percentage became 4.3% (traditional method: 0.9462 vs GNN pipeline: 0.9869). These results indicate that the GNN pipeline is slightly more resilient to the problem of merging distinct particle tracks compared to the traditional method. However, its main strength lies in its ability to reject false edges that erroneously connect background noise to a given particle track.

Figure 6 shows two demonstrative examples of events from GlueX detector that compare the traditional method with the ML pipeline by highlighting their limitations. The event shown in figure 6(a) demonstrates an instance where the event graph was built with a $skip_{max}$ parameter of 3 whereas the track had 7 consecutive missing hits, as such graph-builder does not connect the distant hits. To fix this, it is generally desired to increase this parameter to at least the possible number of consecutive missing hits. This solution is

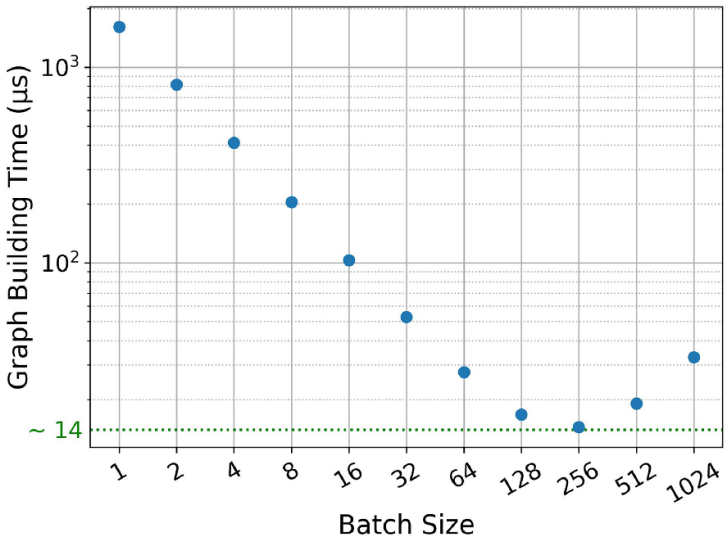


Figure 7. Average pre-processing time per GlueX event for different batch sizes on A100 GPU. The average is taken over ten trials where 16 384 event graphs are built in each trial. An average of approximately 14 μs per event is scored at a batch size of 256.

only valid on the simulated data with available ground-truth information. It is worth reiterating that setting the $skip_{max}$ parameter to an arbitrarily large number can significantly increase the number of redundant edges, resulting in extremely dense graphs that are computationally expensive to process. Figure 6(b) on the other hand shows an event in which the traditional method merges two distinct particle tracks. As a result, the two tracks were not properly captured.

6.2. Parameters selection

The performance results presented in the previous subsection were obtained using a GNN model configured with the parameters listed in table 1. In this subsection, we justify certain design choices based on both, pipeline performance and inference time. In designing a new track-finding system, it is essential to consider inference time alongside performance, particularly in comparison to the traditional method, which takes approximately 152 ± 1 μs per GlueX event.

Parallel computation was employed to reduce the pipeline's inference time. To avoid the complexities associated with multiprocessing, a vectorized PyTorch implementation of the graph-building stage was deployed on GPU to leverage its strong parallelism capabilities. Specifically, a batch of events are pre-processed simultaneously, with a constraint that prevents edge connection between hits belonging to different events. As a result, the output of this stage is a large graph composed of multiple disconnected event subgraphs. This structure does not affect the subsequent GNN model since no message passing occurs between the disconnected components.

Figure 7 illustrates a clear trend of decreasing pre-processing time per event as the batch size increases up to 256 beyond which the time begins to rise again–likely due to GPU compute or memory saturation.

To limit the model inference time, only a single message passing iteration was used (i.e. $I = 1$). In addition, shallow MLP modules with a depth, D, of 1 are used within the classifier GNN. To compensate for the reduced expressivity resulting from the shallowness of the network, we adopted a relatively large width (i.e. $W = 128$) compared to the input dimensionality corresponding to three positional coordinates. To expand further on this, we performed a light grid scan on the parameters W and I of the network. Table 2 shows both the pipeline efficiency and inference time. Based on the results, we adopt the configuration $\{I = 1, W = 128\}$, as further increases in these parameters result in no significant performance gains while substantially increasing inference time. Generally, the number of message passing iterations is an important hyperparameter in GNN models. Increasing this parameter allows each node to aggregate information from more distant nodes, thereby gaining broader awareness of its graph neighborhood. However, setting this number too high can lead to the common issue of over-smoothing, where node representations become indistinguishably similar across the graph [26, 27]. To mitigate this effect, we concatenate the original embedding, X, into the node embedding after every message passing iteration as described earlier in

Table 2. Pipeline efficiency; inference time (μs). Efficiency values are reported for the test set and measured at a fixed purity of 0.9462, matching the traditional method. The inference time values are calculated at a batch size of 256 (which minimizes graph construction time) over an ensemble of 10 trials, building 16 384 event graphs in each trial on A100 GPU.

	$I=1$	$I=2$	$I=3$
$W=64$	0.9771; 27 ± 1	0.9839; 37 ± 1	0.9848; 41 ± 5
$W=128$	0.9806; 28 ± 1	0.9848; 41 ± 2	0.9857; 87 ± 21
$W=256$	0.9808; 74 ± 26	0.9847; 112 ± 17	0.9852; 219 ± 23

Table 3. Efficiency values at different dropout rates for the $\{I=1, W=128\}$ setting.

Dropout Probability	0	0.05	0.1	0.15	0.2
Pipeline Efficiency	0.979	0.9806	0.9803	0.9783	0.9732

Table 4. Maximum resource values of the U200 part 'xcu200-fsgd2104-2-e' used for synthesis.

BRAM	DSP	FF	LUT	URAM
4320	6840	2364 480	1182 240	960

section 5. The limited performance improvement caused by increasing I could be attributed to the shortcut edges introduced by the $skip_{max}$ parameter, which allows a node to become aware of its distant neighbors after a single message passing iteration. With an average inference time of 28 μs, the pipeline achieves an 81.6% speedup compared to the traditional method (152 ± 1 μs), while also yielding better performance, as demonstrated in the previous subsection. We maintain a batch size of 256, as the pipeline inference times for batch sizes of 128 and 512 are higher - 44 ± 1 μs and 34 ± 2 μs, respectively.

Finally, table 3 presents the efficiency values obtained at different dropout rates, with a maximum efficiency observed at a dropout probability of 0.05.

6.3. FPGA

The motivation for using FPGA lies in achieving even greater speed improvements compared to the A100 GPU. While the models presented earlier demonstrate strong performance, their large size renders them infeasible for deployment on FPGAs, which are constrained by limited hardware resources. To determine the largest model that could fit on our FPGA, we conducted a simple study by varying the width, W, of the MLP modules in the edge-classifier GNN. Several randomly instantiated models, each with a different W, were converted into FPGA-compatible C++ code using an open-source Python package named high-level synthesis (HLS) for ML, widely known as `hls4ml` [28]. Each model was designed to process events with up to 150 nodes and 256 edges. The U200 FPGA board was used to obtain resource estimates during synthesis, whose maximum resources can be seen in table 4

Figure 8 shows the different resource usage estimates for different W values, namely, digital signal processing (DSP), flip-flop (FF), and look up table (LUT) when Dataflow is enabled, an optimization technique that allows concurrent execution of high-level functions, significantly improving throughput and reducing Initiation Interval (II). The estimates were obtained using Vitis HLS tool [29]. For FPGA deployment, we chose a pre-trained model with $W=16$ where the FF and LUT resources become nearly saturated. Table 5 shows that the model utilizes up to 77% of the memory resources (i.e. FF) while significantly reducing the time between successive outputs (i.e. II) to 2.5 μs. This represents a substantial speedup compared to the A100 GPU inference time. However, this speedup comes at the cost of a slight reduction in efficiency (0.9565 vs. 0.9806) at the same purity level of 0.9462, attributed to the reduced model capacity. Furthermore, we have not fully developed and optimized graph building for FPGA yet, as such these plots only show the model resource usage and timing performance.

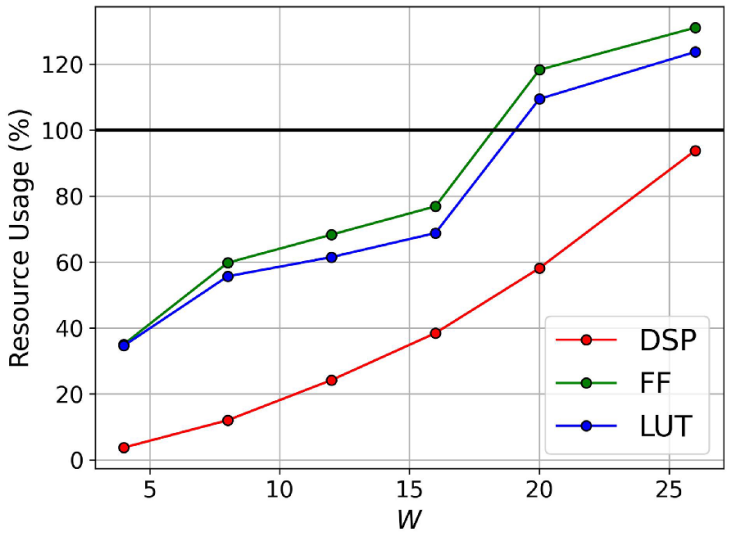


Figure 8. Digital signal processing (DSP), flip-flop (FF), and look up table (LUT) resources usage on FPGA. W represents the width of the hidden layers of MLP modules in the GNN model. At $W = 16$, FF and LUT resources become nearly saturated.

Table 5. Resources and timings of various functions on the FPGA with the same hyperparameters specified previously ($I = 1$, $W = 16$, $D = 1$). `READ_X` and `READ_E` refer to reading node features and adjacency list from the data stream, respectively. `WRITE_prediction` refers to writing last edge-network call (i.e. `edge_network_I`) predictions to the data stream.

	Latency cycles; μs	II cycles; μs	% DSP	% FF	% LUT
`edge_classifier`	1713; 8.57	503; 2.52	38.44	76.87	68.78
`READ_X`	452; 2.26	452; 2.26	0.00	0.80	0.35
`READ_E`	502; 2.51	502; 2.51	0.00	0.34	0.40
`edge_classifier_mlp`	160; 0.80	160; 0.80	6.39	4.73	1.99
`edge_network_i`	417; 2.09	417; 2.09	9.33	18.71	11.40
`node_network`	161; 0.81	161; 0.81	13.95	11.24	8.01
`edge_network_I`	265; 1.33	265; 1.33	8.77	5.92	7.03
`WRITE_prediction`	252; 1.26	252; 1.26	0.00	0.00	0.12

7. Conclusion & future work

In this study, we presented a ML pipeline to solve the challenging track finding problem. The pipeline is composed of: 1) graph-builder that addresses the missing track hits problem resulting from the limited efficiency of the detector by introducing skip edges, and 2) edge-classifier GNN that learns to filter out false edges. By batching multiple event graphs, the parallel computational resources of modern GPUs are leveraged, resulting in a significant reduction of the average ML pipeline inference time compared to the traditional method used for track finding in the GlueX experiment at Jefferson Lab (28 μs vs 152 μs). On simulated data, the pipeline outperforms the traditional method by yielding a 7.5% higher efficiency at a fixed purity level. The pipeline also shows modest improvement in preventing the merging of distinct particle tracks on simulated data. In addition, FPGA implementation was explored to reduce the inference time even further at a slight cost of reduced efficiency compared to the big model used on GPU.

In the future, we will explore deploying the pipeline in the experimental halls using both GPUs and FPGAs. GNN model will be updated to include uncertainty quantification with each prediction. We also aim to address the track fitting problem that accounts for the majority of reconstruction time via novel ML architectures. The results presented in this study are focused on the GlueX experiment at Jefferson Lab which is characterized by a relatively low track multiplicity. We intend to apply this method on experiments with significantly higher track multiplicity. It is expected that the presented approach would yield significant speed improvements over the traditional combinatorial method that scales worse than linearly as the multiplicity increases. The quantification of such potential improvements is yet to be explored.

Data availability statement

The data cannot be made publicly available upon publication due to legal restrictions preventing unrestricted public distribution. The data that support the findings of this study are available upon reasonable request from the authors.

Acknowledgments

This manuscript has been authored by Jefferson Science Associates (JSA) operating the Thomas Jefferson National Accelerator Facility for the U.S. Department of Energy under Contract No. DE-AC05-06OR23177. The authors acknowledge support by the U.S. Department of Energy, Office of Science. The US government retains and the publisher, by accepting the article for publication, acknowledges that the US government retains a nonexclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this manuscript, or allow others to do so, for US government purposes. DOE will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (http://energy.gov/downloads/doe-public-access-plan).

Appendix. Graph-builder constraint optimization using genetic algorithm

The determination of graph-builder constraint values is driven by the need to optimize efficiency and purity. The role of the edge-classifier GNN is limited to filtering out the false edges without introducing any new edges. As such, the input graphs should have maximum efficiency. Graph builder can reduce some of the unwanted edges by applying some constraints based on detector geometry. However, there is a trade-off between efficiency and purity. This competing relationship is depicted in figure A1, which presents the Pareto front generated using the NSGA-II [30] multi-objective genetic algorithm (MOGA) [23].

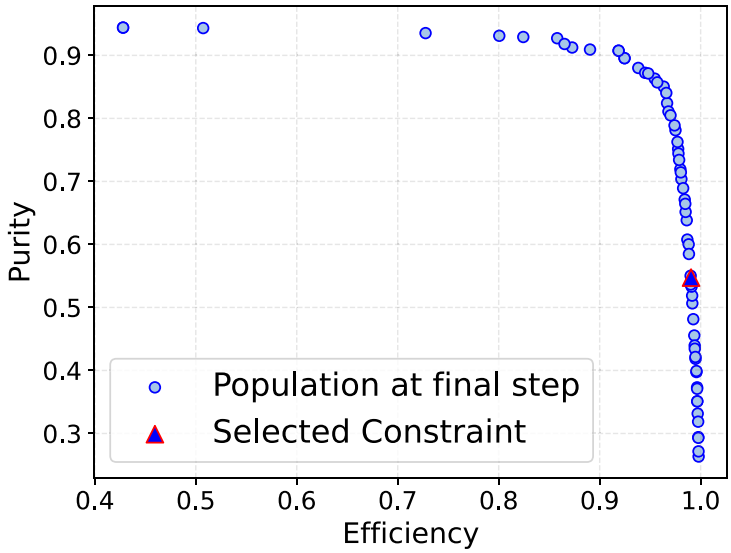


Figure A1. Pareto front of optimal efficiency and purity produced at the final step of MOGA optimization over the graph-builder constraints.

Multi-objective optimization (MOO) problems are characterized by solutions that form a Pareto front. This front consists of solutions where improving one objective necessarily compromises another. Essentially, it represents the set of optimal trade-offs among competing objectives. To evaluate the quality of a Pareto front, the hypervolume metric (also called the S-metric) measures the portion of the objective space that is covered by the Pareto front in relation to a predefined reference point. This reference point is typically chosen to be worse than the ideal values across all objectives. The hypervolume is computed by determining the size of the region in the objective space that the Pareto front dominates while being constrained by the reference point.

MOGA was run for 15 steps and stopped after Pareto-hypervolume started to plateau as shown in figure A2. We used the standard reference point of (0, 0) for the hypervolume calculation. The MOGA

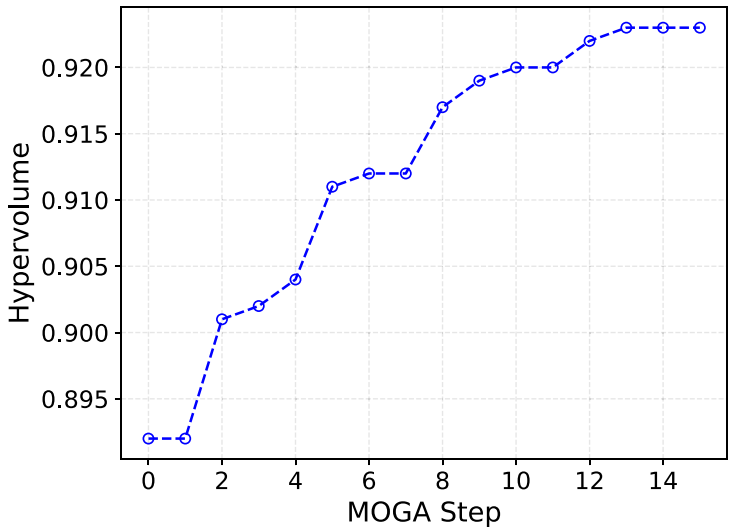


Figure A2. Hypervolume evolution during MOGA optimization.

parameters of population size, mutation score, and cross-over probability were set to their default values of 64, 0.01, and 0.95, respectively. From all the possible constraint values on the Pareto front, we chose the one that achieved a minimum efficiency of 0.99 for highest purity. The efficiency and purity of the graph-builder with these constraints as listed in section 4 are 0.9905 and 0.5473, respectively.

ORCID iDs

Ahmed Hossam Mohammed 0000-0002-6427-3003
Kishansingh Rajput 0000-0002-4430-9937
Simon Taylor 0009-0005-2542-9000
Denis Furletov 0000-0002-2238-8857
Sergey Furletov 0000-0002-7178-8929
Malachi Schram 0000-0002-3475-2871

References

[1] Frühwirth R and Strandlie A 2021 Event Reconstruction *Pattern Recognition, Tracking and Vertex Reconstruction in Particle Detectors* (Springer) pp 23–31
[2] Okawa H 2024 Charged particle reconstruction for future high energy colliders with quantum approximate optimization algorithm *Intelligent Computers, Algorithms and Applications* ed C Cruz, Y Zhang and W Gao (Springer) pp 272–83
[3] Heinrich L, Huth B, Salzburger A and Wettig T 2024 Combined track finding with GNN & CKF (arXiv:2401.16016)
[4] Adhikari S *et al* 2021 *Nucl. Instrum. Methods Phys. Res. A* **987** 164807
[5] Hansroul M, Jeremie H and Savard D 1988 *Nucl. Inst. Methods Phys. Res. A* **270** 498–501
[6] Kisel I 2006 *Nucl. Inst. Methods Phys. Res. A* **566** 85–88
[7] Brondolin E, Leogrande E, Hynds D, Gaede F, Petrič M, Sailer A and Simoniello R 2020 *Nucl. Instrum. Methods Phys. Res. A* **956** 163304
[8] Hough P V C 1959 *Conf. Proc.* C **590914** 554–8
[9] Zhou H, Sun K, Lu Z, Li H, Ai X, Zhang J, Huang X and Liu J 2025 *2025 Nucl. Instrum. Methods Phys. Res. A* **1075** 170357
[10] Frühwirth R and Strandlie A 2021 Track Finding *Pattern Recognition, Tracking and Vertex Reconstruction in Particle Detectors* (Springer) pp 81–102
[11] Farrell S *et al* 2018 Novel deep learning methods for track reconstruction (arXiv:1810.06111)
[12] Gori M, Monfardini G and Scarselli F 2005 A new model for learning in graph domains *IEEE Int. Joint Conf. on Neural Networks, (Proc.)* vol 2 pp 729–34
[13] Scarselli F, Gori M, Tsoi A, Hagenbuchner M and Monfardini G 2009 *IEEE Trans. Neural Netw.* **20** 61–80
[14] Kipf T N and Welling M 2017 Semi-supervised classification with graph convolutional networks (arXiv:1609.02907)
[15] Hamilton W L, Ying R and Leskovec J 2018 Inductive representation learning on large graphs (arXiv:1706.02216)
[16] Veličković P, Cucurull G, Casanova A, Romero A, Lió P and Bengio Y 2018 Graph attention networks (arXiv:1710.10903)
[17] Ju X *et al* 2020 Graph neural networks for particle reconstruction in high energy physics detectors (arXiv:2003.11603)
[18] Ju X *et al* 2021 *Eur. Phys. J. C* **81** 1–14
[19] DeZoort G, Thais S, Duarte J, Razavimaleki V, Atkinson M, Ojalvo I, Neubauer M and Elmer P 2021 *Comput. Softw. Big Sci.* **5** 1–13
[20] Heintz A *et al* 2020 Accelerated charged particle tracking with graph neural networks on FPGAs (arXiv:2012.01563)
[21] Elabd A *et al* 2022 *Front. Big Data* **5** 828666

[22] Allison J et al 2016 Nucl. Instrum. Methods Phys. Res. A 835 186–225
[23] Fonseca C and Fleming P 1993 Multiobjective genetic algorithms IEE Coll. on Genetic Algorithms for Control Systems Engineering p 6/1–6/5
[24] Nair V and Hinton G E 2010 Rectified linear units improve restricted boltzmann machines Proc. 27th Int. Conf. on Machine Learning (ICML-10) pp 807–14
[25] Kingma D P and Ba J 2014 arXiv:1412.6980
[26] Cai C and Wang Y 2020 (arXiv:2006.13318)
[27] Rusch T K, Bronstein M M and Mishra S 2023 A survey on oversmoothing in graph neural networks (arXiv:2303.10993)
[28] Team F 2024 fastmachinelearning/hls4ml (available at: https://github.com/fastmachinelearning/hls4ml)
[29] AMD 2021 Vitis High-level synthesis user guide: introduction (UG1399) AMD (available at: https://docs.amd.com/r/en-US/ug1399-vitis-hls/Introduction) (Accessed: 21 March 2025)
[30] Deb K, Pratap A, Agarwal S and Meyarivan T 2002 IEEE Trans. Evol. Comput. 6 182–97