

Graph Neural Network-Based Track Finding in the LHCb Vertex Detector

ANTHONY CORREIA¹, FOTIS I. GIASEMIS^{2,1*}, NABIL GARROUM¹,
VLADIMIR VAVA GLIGOROV^{1,3}, BERTRAND GRANADO²

¹ *LPNHE, Sorbonne Université, CNRS/IN2P3, Paris, France*

² *LIP6, Sorbonne Université, CNRS, Paris, France*

³ *Organisation Européenne pour la Recherche Nucléaire (CERN)*

* Corresponding author: Fotis.Giasemis@cern.ch

ABSTRACT

The next decade will see an order of magnitude increase in data collected by high-energy physics experiments, driven by the High-Luminosity LHC (HL-LHC). The reconstruction of charged particle trajectories (tracks) has always been a critical part of offline data processing pipelines. The complexity of HL-LHC data will however increasingly mandate track finding in all stages of an experiment's real-time processing. This paper presents a GNN-based track-finding pipeline tailored for the Run 3 LHCb experiment's vertex detector and benchmarks its physics performance and computational cost against existing classical algorithms on GPU architectures. A novelty of our work compared to existing GNN tracking pipelines is batched execution, in which the GPU evaluates the pipeline on hundreds of events in parallel. We evaluate the impact of neural-network quantisation on physics and computational performance, and comment on the outlook for GNN tracking algorithms for other parts of the LHCb track-finding pipeline.

1 Introduction

Modern high-energy physics (HEP) collider experiments rely on a precise reconstruction of charged particle trajectories (tracks) to identify physics processes of interest and to separate them from backgrounds. As the instantaneous luminosity at which experiments operate increases so does the number and density of hits in each individual “event” and therefore the track-finding complexity. At the same time this increasing complexity makes it more essential to perform tracking at the earliest stages of the processing pipeline, including in real-time. The LHCb [1, 2] and ALICE [3–5] detectors at the Large Hadron Collider (LHC) already perform tracking¹ in software at the full LHC collision rate. The ATLAS [6] and CMS [7] collaborations are currently building upgraded detectors which will be able to operate at the High-Luminosity LHC (HL-LHC), at instantaneous luminosities up to four times higher than the current ATLAS and CMS detectors and almost forty times higher than the current LHCb detector. At the HL-LHC both ATLAS and CMS will increase the rate at which track finding is performed in software by roughly an order of magnitude compared to the present, while CMS will additionally perform [8] a partial track finding using FPGA cards at the full LHC collision rate.

While tracking is a major part of the overall computational budget for all these four experiments, the computational cost of classical tracking algorithms scales with the number of hits to the power 2-3 [9, 10] for the same physics performance. At the same time, the development of computing architectures is increasingly driven by machine-learning and artificial intelligence applications, such as the deployment of specialized hardware like Google’s tensor processing units and Nvidia’s tensor cores inside GPUs for efficient deep learning computations. While all major high-energy physics experiments have successfully reoptimized their classical tracking algorithms to efficiently exploit a range of current parallel computer architectures over the past decade, it is also worth asking whether tracking algorithms based on neural networks might be a better long-term fit to the hardware which our reconstruction has to run on.

There has been a significant amount of R&D [11–16] performed in the community on this topic over the last years. In particular, the Exa.TrkX collaboration developed a highly parallelizable Graph Neural Network (GNN)-based pipeline for track finding [17]. This pipeline was initially designed for 4π tracking detectors in a magnetic field, similar to those used in ATLAS and CMS, specifically for the High-Luminosity upgrade of the LHC (HL-LHC). Using this pipeline as a starting point, we developed [18] our own pipeline for track finding in LHCb’s vertex detector, called ETX4VELO. In this paper, we extend our earlier results [18] by partially implementing ETX4VELO in C++ and CUDA, excluding the triplet steps described in Section 5.3. Additionally, we present several optimizations, including minimizing the sizes of the ML models, improving the models’ architectures, and restricting k-NN computations to successive planes. We report physics and computational performance with and without neural network quantisation and discuss the outlook for further speedups of the ETX4VELO pipeline and applications to track finding in the rest of the LHCb detector.

2 The LHCb detector and computing framework

The LHCb Run 3 detector [19] is a general-purpose detector covering a pseudorapidity (η) region between 2 and 5 and optimised for making precision measurements of heavy flavour hadrons. The LHC proton bunches cross every 25 ns within its Vertex Locator (VELO), with a beam-beam collision rate of around 24 MHz in 2024 datataking and a nominal pileup of around five proton-proton collisions per LHC bunch crossing. The particles produced in these collisions are detected by the tracking system, illustrated in Figure 1. This system consists of:

- the **Vertex Locator (VELO)** [20], surrounding the proton-proton interaction region, encompassing $n_{\text{planes}} = 26$ planes of $55 \times 55 \mu\text{m}^2$ silicon pixels,
- the **Upstream Tracker (UT)** [21], placed before the magnet station, including 4 planes of silicon strips, and
- the **Scintillating Fibre Tracker (SciFi)** [21], located after the magnet station, composed of 12 planes of scintillating fibres.

¹This is a partial track reconstruction in LHCb’s case, and a full one in the case of ALICE.

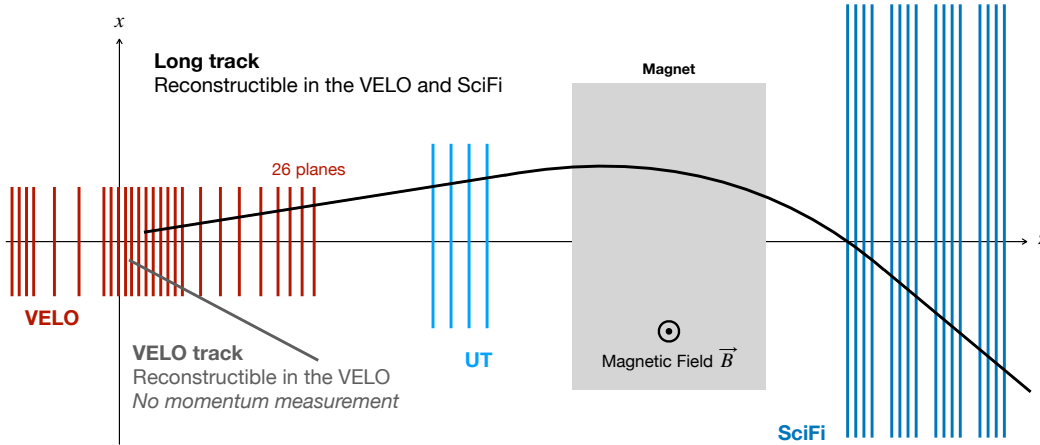


Figure 1: Sketch of the LHCb tracking system in Run 3.

The LHCb data acquisition can sustain a peak data rate of 5 TB/s which must be reduced to 10 GB/s before being written to long-term storage. This reduction is performed by a two-stage real-time processing system, the so-called trigger. The first stage is implemented on GPUs and is called Allen² [2]. Allen performs a partial event reconstruction, including tracking, and reduces the rate to between 70 and 200 GB/s using around 50 physics selections. The second stage is implemented on CPUs using the Moore³ application based on the Gaudi⁴ framework. Moore performs a full offline-quality reconstruction of the LHCb detector and reduces the rate to 10 GB/s using over two thousand physics selections.

Given this context, the Allen framework and LHCb offer an interesting use-case for developing Neural Network-based algorithms on GPUs, with a specific emphasis on high-throughput performance. In particular LHCb processes the full LHC collision rate using only around 320 GPU cards, leading to a minimum required throughput of around 80 kHz for the full processing pipeline on a single GPU. This in turn makes it essential to minimize overheads caused by data transfers between the host server and the GPU, which Allen achieves by processing batches of $O(1000)$ events at a time. To the best of our knowledge this is the first time that a GNN tracking pipeline has been implemented using batched processing of this kind, or targeting these kinds of throughputs per GPU.

3 Track Topologies in the VELO

As there is no magnetic field in the VELO, the tracks are straight lines. Since the LHCb detector is not symmetric around the interaction region, tracks are classified as “forward”, i.e. travelling towards the LHCb magnet and the rest of the LHCb detector, or “backward”, travelling away from the LHCb detector. When evaluating tracking efficiencies we only consider particles which leave at least three hits in the VELO as “reconstructible”. [22]

A VELO plane consists of four overlapping sensor layers, displaced along the z -axis, collaboratively covering the desired acceptance in the (x, y) -plane. The effective hit efficiency of the VELO is around 99% in simulation, so tracks must be allowed to skip a VELO plane during the reconstruction. A track can also

²<https://gitlab.cern.ch/lhcb/Allen>

³<https://gitlab.cern.ch/lhcb/Moore>

⁴<https://gitlab.cern.ch/gaudi>

leave more than one hit per plane, in most cases when it traverses overlapping sensor layers within a given plane. Material interactions frequently produce positron-electron pairs, resulting in two tracks that initially share hits before diverging. In fact 55% of electrons share hits with another particle, which motivates the need to take particular care when reconstructing them. Two tracks may also accidentally intersect, leading to a shared hit. The tracks that an effective VELO tracking algorithm should be capable of reconstructing are illustrated in Figure 2.

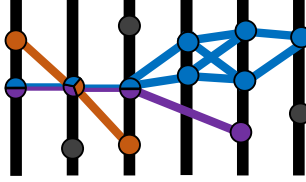


Figure 2: Simplified example of tracks to be reconstructed in the VELO. The blue and purple tracks share three hits prior to diverging. The purple track jumps from plane 3 to 5, missing 4. The orange track intersects the blue and purple tracks. Dark points represent hits unassociated with any particle. When considering the hits as graph nodes, lines between hit nodes represent the genuine edges, as defined in this work. Reproduced from [18].

4 Datasets

In order to have the information required for the training process of the machine learning models involved, such as the truth labels required for calculating the loss for a classification task, the pipeline is trained on simulated events. Similarly, for the evaluation of the accuracy of the algorithm, a simulation sample is necessary in order to have information about the particles produced in each event.

The results presented in this paper have been obtained using events produced with the full LHCb detector simulation in which p - p collisions are generated using PYTHIA [23] and decayed using EvtGen [24]. The interaction of the generated particles with the detector, and its response, are implemented using the Geant4 toolkit [25] as described in [26]. We use simulated minimum bias samples corresponding to typical LHCb datataking conditions between 2022 and 2025, with an average of 5.3 inelastic p - p collisions which produce at least one particle with momentum above 2 GeV in the LHCb detector acceptance per event. In our sample, there are, on average, 150 particles in the VELO acceptance, and 2,200 hits, in each event. In the simulation used for this paper around 15% of the hits are spilled over from prior events and therefore noise. The embedding network and GNN, introduced in Section 5, are trained on 700,000 events adhering to the selection criteria below.

1. **Removal of non-linear particle tracks:** The hits of the particle tracks that are not sufficiently linear due to multiple scattering (largely soft electrons) are removed. This is assessed by fitting a line to the particle hits and applying an upper limit to the average squared distance between the hits and the line. This criterion improves the physics performance during the training but also excludes 2.5% of the tracks reconstructible in the VELO.
2. **Minimum number of VELO hits:** A minimum of 500 genuine VELO hits is required.
3. **Exclusion of tracks with insufficient hits:** Tracks with fewer than 3 hits are excluded.

These criteria are not imposed on the test samples. Allen’s existing tracking algorithms, which are used by LHCb in 2024 datataking, are used as a reference for both the physics and computational performance of ETX4VELO, described in detail in Sections 6 and 7 respectively.

5 The ETX4VELO Pipeline

In general, a graph is a pair $G = (V, E)$, where V is a finite set of vertices (or nodes), and E is the set of connections (known as edges) between these nodes. In our problem, the hits left by charged particles in the VELO detector form a point cloud in 3-dimensional space. However, they can also be represented as a graph, in which the successive hits (or nodes) of each particle traversing the detector are connected to each other. This graph can be thought of as the “truth graph” that perfectly describes the VELO event. Conceptually, the end-goal of the ETX4VELO pipeline is to produce a graph that closely approximates this truth graph.

More specifically, the basic idea of our GNN pipeline is to build an initial graph of possible connections between hits in the detector, accurately classify these connections as correct (true/genuine) or incorrect (fake), and then, after discarding the fake ones, transform them into a set of track objects, containers of the hits of each track, which can then be understood and used by the rest of the algorithms in LHCb’s real-time pipeline. Since a graph with N nodes consisting of all possible node connections would have $C(N, 2) = N(N - 1)/2$ edges, and thus would be prohibitively large, a key challenge is to construct this initial graph in such a way that nearly all initial connections made are part of the final graph, while as many fake connections as possible are not. To illustrate this, for a typical VELO event of $N = 2,200$ hits, a maximally connected graph would have $C(N, 2) = 5 \times 10^6$ edges, while our method, described in the following paragraphs, creates on average 30,000 edges.

Our pipeline consists of 7 steps. We summarize the steps here, and then give further details on each step in dedicated subsections. Starting with the hits alone, the first two steps of the ETX4VELO pipeline construct a rough graph, $G_{\text{rough}}^{\text{hit}}$. Hits are first embedded into a Euclidean space using a Multi-Layer Perceptron (MLP). The MLP is trained to position hits that are likely to be connected by an edge close to each other in the embedding space. Subsequently, k -Nearest Neighbours (k -NN) algorithms are applied in the embedding space to collect edges that are most likely to be genuine. Nodes are considered at most two planes apart when identifying edge candidates.

The next four steps involve the GNN. A novelty of the ETX4VELO pipeline is the classification of edge connections in addition to individual edges, which allows for the separation of tracks that share hits. This involves transitioning from a graph of connected hits, G^{hit} , to a graph of edges, G^{edge} . Edge connections are referred to as triplets because they are formed by three hits. To minimise the number of edge connections, which increases exponentially with the number of edges, it is essential to first filter out fake edges.

First, a GNN encodes each edge in $G_{\text{rough}}^{\text{hit}} (i \rightarrow j)$ into a high-dimensional vector $e_{i \rightarrow j}$, with dual training targets: edge classification and triplet classification. Second, the *edge classifier* network transforms these encodings into edge scores ranging from 0 (fake) to 1 (genuine). Edges with scores below $s_{\text{edge, min}}$ are discarded, resulting in the purified hit graph $G_{\text{purified}}^{\text{hit}}$. Third, the graph of edges G^{edge} is built from the purified hit graph $G_{\text{purified}}^{\text{hit}}$. Finally, the *triplet classifier* network transforms the pairs of edge encodings (triplets) into triplet scores. Triplets scoring below $s_{\text{edge, min}}$ are discarded, resulting in the purified edge graph $G_{\text{purified}}^{\text{edge}}$.

The final step involves constructing tracks from the purified edge graph $G_{\text{purified}}^{\text{edge}}$. The Exa.TrkX pipeline applies a Weakly Connected Component (WCC) algorithm [27] to the purified hit graph $G_{\text{purified}}^{\text{hit}}$ to interpreted sets of connected hits as tracks. The classification of edge connections rather than only edges allows the WCC algorithm to be modified in such a way as to allow tracks to share multiple hits, which is particularly important for the efficiency of reconstructing electron-positron pairs.

All these steps are illustrated in Figure 3.

5.1 Hit Embedding and Rough Graph Construction

To construct the graph $G_{\text{rough}}^{\text{hit}}$, one could connect each hit to all hits on the next two planes, allowing for the possibility of a missing plane due to pixel inefficiencies. However, this approach results in an excessive number of edges, thereby increasing the GNN’s inference time and memory usage. To enhance throughput, it is essential to minimise the graph size at this stage. We start by describing the operation of our method and then the training process and loss function in Eqs. (2) and (3).

Most VELO tracks are produced directly from the initial proton-proton interactions, which occur in a

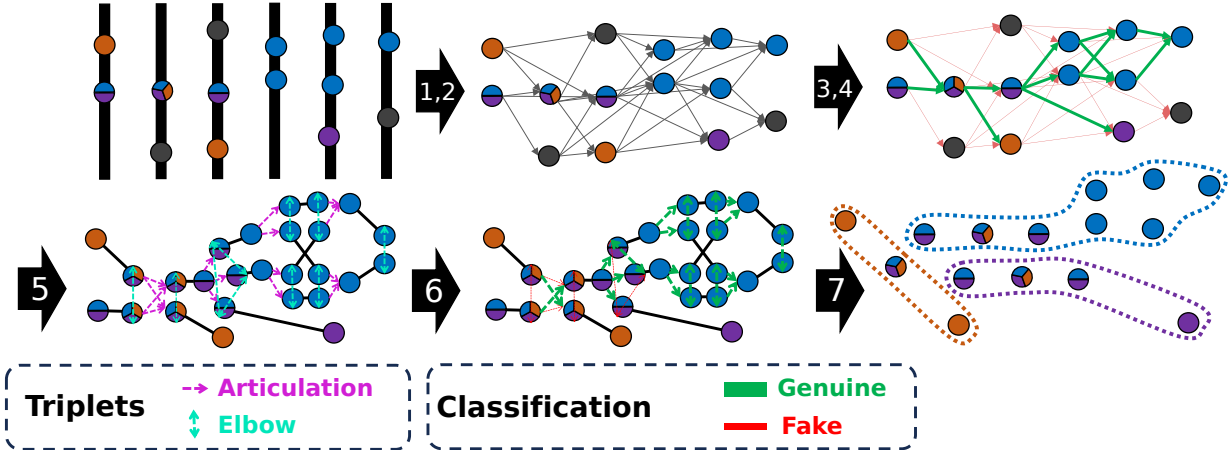


Figure 3: Illustration of the pipeline’s 7 steps, beginning with hits from the minimalist example in Figure 2. Steps entail (1) embedding the hits, (2) building a rough hit graph, (3) encoding the edges, (4) classifying the edges and discarding fakes (in red), (5) constructing the edge graph with edge-to-edge connections called triplets, (6) classifying and removing fake triplets, and finally, (7) producing the tracks.

relatively narrow interaction region with a spread of around 45 mm in z and around $30 \mu\text{m}$ in x and y . This fact strongly constrains which edges have to be considered when constructing our graph. The embedding MLP captures this by accepting the hits h as input and embedding them into an n_{dim} -dimensional space $e_h \in \mathbb{R}^{n_{\text{dim}}}$. This is done by passing the normalised⁵ cylindrical (r, ϕ, z) coordinates to the MLP. In this space, likely connected hits are positioned close together based on a reference squared distance $m = 1^6$, while unlikely connections are spaced apart. Here, the squared distance $d^2(a, b)$ between two hits a and b is defined as the usual Euclidean distance

$$d^2(a, b) = \|e_a - e_b\|^2 = \sum_{i=1}^{n_{\text{dim}}} (e_{a,i} - e_{b,i})^2. \quad (1)$$

Using this trained embedding MLP, a hit on plane p is connected to hits on the next two planes, $p + 1$ and $p + 2$, if they are within a squared distance of d_{max}^2 . To avoid an excessive number of edges, a maximum of k_{max} edges per node is imposed. Consequently, the rough graph $G_{\text{rough}}^{\text{hit}}$ is constructed by applying a k_{max} -NN algorithm on plane $p \in \llbracket 0, n_{\text{planes}} - 1 \rrbracket$ to the next two planes, $p + 1$ and $p + 2$, under a maximum squared distance of d_{max}^2 . The k-NN implementation from `faiss` [28] is used for this purpose. The values of the hyperparameters k_{max} and d_{max}^2 are determined post-training to balance the tradeoff between efficiency and clone rate, metrics described in Section 6. Here these values are chosen to be $k_{\text{max}} = 50$ and $d_{\text{max}}^2 = 0.9$.

In the training process, each step corresponds to one event, with noise hits removed as they are considered random and unrelated. The training set T is composed of hit pairs from a query node $q \in Q$ to another node a on the next two planes, representing $q \rightarrow a$ edge candidates. To focus on significant particles, a hit must belong to a reconstructible particle within acceptance and not be an electron to qualify as a query node. Almost all of the edges from electrons are identified by the network without training specifically on them and thus electrons are excluded. The training set $T = T_{\text{genuine}} \cup T_{\text{fake}}$ includes both connected pairs T_{genuine} and disconnected pairs T_{fake} . It is constructed by merging three sets of pairs:

Hard-negative mining: Fake pairs are generated using the same $k_{\text{max}}^{\text{training}}$ -NN procedure with $(d_{\text{max}}^{\text{training}})^2$ as during inference, representing fake pairs that would be classified as genuine during inference. The values $k_{\text{max}}^{\text{training}} = 50$ and $(d_{\text{max}}^{\text{training}})^2 = 1.5$ are used.

⁵The inputs of a neural network are normalised using the mean and standard deviation calculated from a representative set of events.

⁶Using the squared distance instead of the distance avoids the computational cost of square root calculations.

Random pairs: For each query point, $n_{\text{random}} = 1$ pairs are included.

Genuine edges: All genuine edges from the query points are added to the training set.

To train the embedding MLP to reduce the distance of genuine pairs and increase that of fake pairs, the following loss function is minimised:

$$\mathcal{L} = \mathcal{L}_{\text{fake}} + w_{\text{genuine}} \times \mathcal{L}_{\text{genuine}}, \quad (2)$$

where $\mathcal{L}_{\text{genuine}}$ and $\mathcal{L}_{\text{fake}}$ are the normalised pairwise hinge embedding losses [29] for genuine and fake examples, respectively, defined as:

$$\mathcal{L}_{\text{genuine}} = \frac{1}{|T_{\text{genuine}}|} \sum_{(q,a) \in T_{\text{genuine}}} d^2(q,a) \quad \text{and} \quad \mathcal{L}_{\text{fake}} = \frac{1}{|T_{\text{fake}}|} \sum_{(q,b) \in T_{\text{fake}}} \max(0, m - d^2(q,b)), \quad (3)$$

The margin m , representing a squared distance threshold, is fixed at $m = 1$. The parameter $w_{\text{genuine}} > 1$ reduces the likelihood of excluding true edges within this margin, thus favouring the inclusion of genuine edges over the exclusion of false ones. The values m and w , simily to k_{max} and d_{max}^2 , are chosen based on an elementary hyperparameter exploration.

5.2 Graph Neural Network and Classifiers

The Graph Neural Network (GNN) is employed to derive edge encodings used for both edge and triplet classification. The architecture of the GNN closely follows that of the Exa.TrkX collaboration, with minor deviations, as illustrated in Figure 4. The node and edge encoders, the node and edge networks and the edge and triplet classifiers are all MLPs, and should not be confused with the embedding network described in Section 5.1, which also happens to be an MLP.

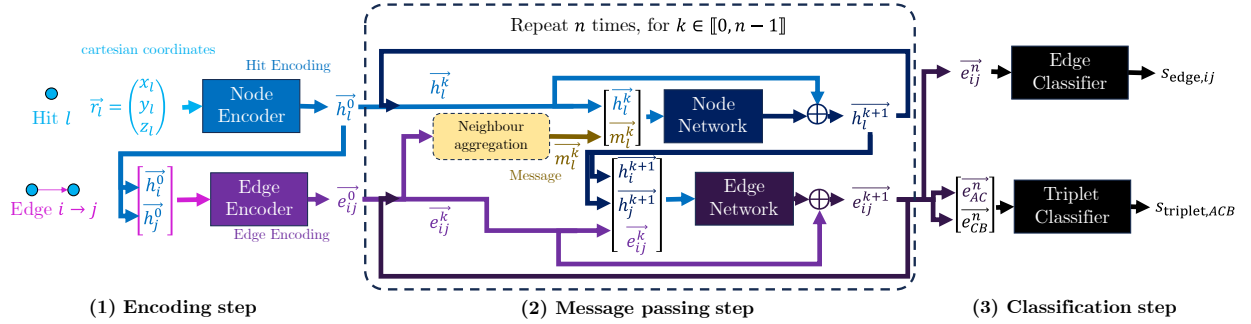


Figure 4: Schematic of the GNN architecture, highlighting: (1) hit and edge encodings, (2) n message passing steps, and (3) subsequent edge and triplet classifications. The node and edge encoders, the node and edge networks and the edge and triplet classifiers are all MLPs.

Initially, the hits l are encoded from their normalised Cartesian coordinates $\vec{r}_l = (x_l, y_l, z_l)$ into an n_h -dimensional space $\vec{h}_l^0 \in \mathbb{R}^{n_h}$ using the node encoder. The concatenated node features \vec{h}_i^0 and \vec{h}_j^0 of edges $i \rightarrow j$ are input to the edge encoder, producing the edge encodings $\vec{e}_{ij}^0 \in \mathbb{R}^{n_e}$ in an n_e -dimensional space.

The hit and edge encodings are then iteratively updated over n message passing steps. These steps allow the encodings to incorporate information from distant neighbours. During each message passing step $k \in \{0, \dots, n-1\}$, a message m_l^k is computed for each hit l by aggregating the encodings of the edges connected to and from hit l . The message is computed as follows:

$$m_l^k = \left[\sum_{j \text{ s.t. } l \rightarrow j \text{ exists}} e_{lj}^k, \quad \sum_{i \text{ s.t. } i \rightarrow l \text{ exists}} e_{il}^k \right] \quad (4)$$

where $[\cdot, \cdot]$ denotes concatenation. This operation of aggregating edge encodings by summing them, using terminology from deep learning frameworks, will be referred to as `scatter_add`. The node network updates the hit encodings h_l^{k+1} using the previous hit encodings h_l^k and the message m_l^k , incorporating a residual connection. Similarly, the edge encodings are updated to e_{ij}^{k+1} using the previous edge encodings e_{ij}^k and the updated hit encodings h_i^{k+1} and h_j^{k+1} , also with a residual connection.

Edge encodings are sufficient for both the edge classifier and triplet classifier. Therefore, the hit encodings are only utilised during the encoding and message passing steps to compute and update the edge encodings. The GNN is trained to classify both edges and triplets by minimizing the sum of the edge and triplet losses:

$$\mathcal{L} = \mathcal{L}_{\text{edges}} + \mathcal{L}_{\text{triplets}}. \quad (5)$$

We use sigmoid focal losses [30] for the edge and triplet classification since, for the purposes of our pipeline, it outperforms the traditional binary cross-entropy loss. To ensure the GNN focuses on relevant triplets, edges with scores below 0.5 are discarded before triplet building and classification.

Several optimizations have been applied to the GNN to improve performance and efficiency. The size of the GNN was significantly reduced, with node and edge encodings now residing in a 32-dimensional space ($n_h = n_e = 32$), down from the initial 256 dimensions. The number of graph iterations was reduced to $n = 5$. Despite the reduction in network size, several changes and corrections were made to maintain reasonable physics performance. Notably, the node and edge networks used at each message passing step are distinct, making the GNN non-recurrent [31]. This approach increases the number of trainable parameters but keeps the throughput unchanged while greatly improving the physics performance.

5.3 Triplet Building

Edge connections, or triplets [15], are constructed from the purified hit graph $G_{\text{purified}}^{\text{hit}}$. Each triplet consists of exactly three hits: one common hit C and two other hits, A and B . There are only three types of triplets that can be formed, as illustrated in Figure 5.

Articulation: Two consecutive edges, $A \rightarrow C$ and $C \rightarrow B$, with the common hit in the middle.

Left Elbow: Edges $C \rightarrow A$ and $C \rightarrow B$, with the common hit on the left.

Right Elbow: Edges $A \rightarrow C$ and $B \rightarrow C$, with the common hit on the right.

It was found that using separate triplet classifiers for articulations and elbows led to better performance. Each of these classifiers is an MLP ending with a layer with 1 unit, and a sigmoid activation function.

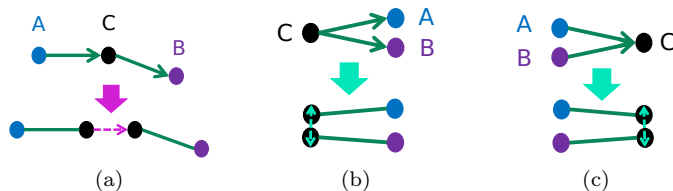


Figure 5: Visual representation of the three triplet configurations in the edge graph: (a) the articulation, (b) the left elbow and (c) the right elbow.

5.4 Track Building

To build tracks from the purified edge graph $G_{\text{purified}}^{\text{edge}}$, simply applying the WCC algorithm would identify sets of connected edges, allowing for the reconstruction of tracks that share exactly one hit. However, electron-positron pairs are created with a very small opening angle between the two tracks and therefore

frequently share multiple hits at the beginning of the tracks. In order to account for this, the process of building tracks from triplets involves 4 steps as illustrated in Figure 6.

First, the left elbows and right elbows are connected, leaving only the articulations to connect. Duplicate edges resulting from elbow connections are removed, so that when two tracks share their initial hits, it is equivalent to two articulations sharing an edge. Second, a WCC algorithm is applied to the edge graph, excluding these shared articulations. The shared articulations now act as connections between two sets of connected edges, with one set being shared. Third, a new track is formed for each remaining articulation, effectively duplicating the shared set of connected edges. Finally, edges are replaced by their corresponding hits, converting the sets of connected edges into sets of connected hits, thereby representing the tracks.

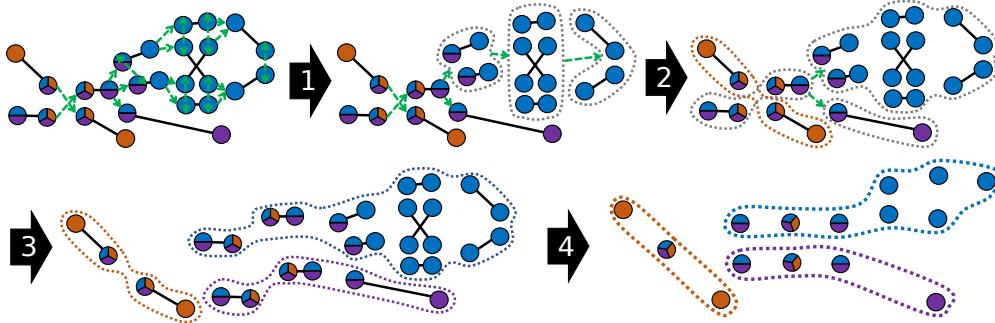


Figure 6: Illustration of the four phases of track construction from the purified edge graph using articulations and elbows defined in Section 5.3. (1) Connecting left and right elbows, (2) applying a WCC while omitting articulations with shared edges, (3) designating each residual link as a unique track, and (4) substituting edges with hits.

6 Physics Performance

The conventions and definitions for track-finding performance in LHCb are outlined in [22]. A track is matched to a particle when at least 70% of its hits are associated with that particle, forming a set of *matching candidates* (track, particle). The metrics used to assess the track-finding performance are presented in Table 1. The uncertainties associated with track-finding efficiency are determined via the Bayesian method with a uniform prior [32] and computed using the ROOT software [33].

Metric	Definition	Formula
Efficiency	Proportion of matched particles	$\frac{\# \text{ matched particles}}{\# \text{ particles}}$
Clone rate	Proportion of redundant candidates	$\frac{\# \text{ candidates} - \# \text{ matched particles}}{\# \text{ candidates}}$
fake rate	Proportion of unmatched tracks	$\frac{\# \text{ unmatched tracks}}{\# \text{ tracks}}$
Hit efficiency	Average proportion of matched hits per particle	$\left\langle \frac{\# \text{ matched hits}}{\# \text{ hits on particle}} \right\rangle_{\text{candidates}}$
Hit purity	Average proportion of matched hits per track	$\left\langle \frac{\# \text{ matched hits}}{\# \text{ hits on track}} \right\rangle_{\text{candidates}}$

Table 1: Metrics for track-finding performance. Efficiency, clone rate, and fake rate encompass all events, while hit efficiency and hit purity are averaged across matching candidates.

Within the VELO’s track-finding context, Figure 1 illustrates two primary particle categories:

VELO-only particles: particles whose tracks are reconstructible in the VELO but not in the SciFi.

Long particles: particles whose tracks are reconstructible both in the VELO and in the SciFi.

Long track trajectories are bent between the UT and the SciFi due to the magnetic field, which enables momentum measurements. Consequently, reconstructing these tracks is crucial for LHCb physics analyses. The aforementioned categories can be further broken down into three sub-categories:

No electrons: All particles except for electrons.

Electrons: Only electrons, which are more challenging to reconstruct due to a higher chance of scattering or photon radiation (bremsstrahlung).

From strange: Non-electron particles originating from a decay chain with an s -quark hadron, excluding electrons. These typically represent tracks originating near the end of the VELO detector, making them harder to reconstruct.

The physics performance of the ETX4VELO pipeline is evaluated on a sample of 1000 events. This performance is compared to that of the default search by triplet algorithm used for track finding in the VELO within the Allen framework. The performance metrics for ETX4VELO and the search by triplet algorithm are summarised in Table 2 for long particles, Table 3 for VELO-only particles, and Table 4 for the fake track rate. These tables also present the performance of the ETX4VELO pipeline without the triplet approach, where tracks are obtained by applying a WCC on the purified graph of hits $G_{\text{purified}}^{\text{hit}}$, as currently implemented in C++/CUDA.

The physics performance is also compared between the ETX4VELO pipeline and Allen for long particles, excluding electrons, as a function of the occupancy of the detector, i.e. the number of hits in each event, in Fig. 7. Various track-finding performance metrics are plotted against the occupancy. Events are split into bins based on their occupancy, and the evaluation of the tracking algorithms is done on the events of each bin. The error bars for the efficiency are binomial errors.

The ETX4VELO pipeline demonstrates performance comparable to the Allen framework, with a fake rate reduced by more than half. Track quality is significantly better across all categories, with consistently higher hit efficiency and hit purity. It excels particularly in reconstructing electron tracks. However, it performs slightly less well for particles originating from strange decays. This shortfall is likely because these particles tend to be more tilted relative to the beam line and may not have been adequately included during the graph building stage due to the chosen squared maximum distance d_{max}^2 .

Long	Efficiency			Clone rate			Hit efficiency			Hit purity		
	Allen	ETX4VELO	(97.96)	Allen	ETX4VELO	(0.88)	Allen	ETX4VELO	(98.42)	Allen	ETX4VELO	(99.95)
No electrons	99.35	99.35	(97.96)	2.61	1.23	(0.88)	96.34	98.58	(98.42)	99.78	99.92	(99.95)
Electrons	95.21	98.10	(51.82)	3.31	3.35	(0.93)	95.69	97.33	(96.46)	98.37	99.55	(95.05)
From strange	97.53	97.43	(92.23)	2.70	1.62	(0.61)	95.85	97.95	(96.39)	99.44	99.59	(99.77)

Table 2: Track-finding performance (in percentages) of the search by triplet algorithm in Allen versus ETX4VELO for long particles. The values in parentheses correspond to the performance of the ETX4VELO pipeline without the triplet approach, as currently implemented in C++/CUDA.

7 C++/CUDA Implementation and Throughput Measurement

The ETX4VELO pipeline is implemented in C++/CUDA using the Allen framework, the implementation of the first-level of the LHCb trigger on GPUs, in order to measure its computational performance. Our implementation leverages Allen’s existing implementations for tasks such as memory management, event loading and dispatching, and VELO hit decoding. The classical Allen reconstruction pipeline is benchmarked dispatching 500 events to each of 16 CUDA streams and allocating 500 MB of GPU memory per stream. The ETX4VELO pipeline must adhere to these constraints to ensure optimal performance. Computational

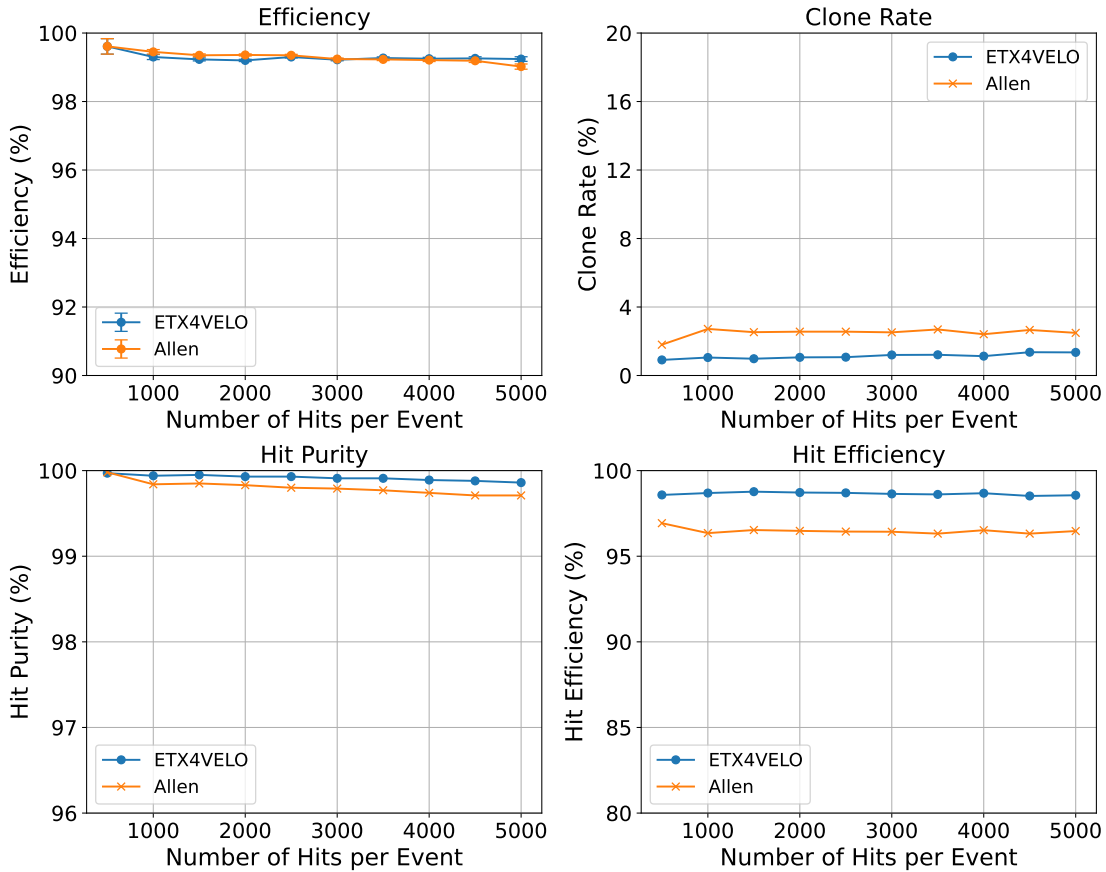


Figure 7: Track-finding performance comparison for the search by triplet algorithm in Allen versus ETX4VELO for long particles, excluding electrons, as a function of the occupancy of the detector.

VELO-only	Efficiency		Clone rate		Hit efficiency		Hit purity	
	Allen	ETX4VELO	Allen	ETX4VELO	Allen	ETX4VELO	Allen	ETX4VELO
No electrons	97.03	97.05 (96.28)	3.65	1.46 (0.87)	94.07	97.68 (97.93)	99.51	99.81 (99.92)
Electrons	67.84	83.60 (49.93)	9.65	6.71 (3.51)	79.57	90.83 (85.25)	97.62	99.17 (98.25)
From strange	94.25	93.69 (84.33)	5.16	4.09 (1.35)	90.33	97.95 (90.79)	99.43	99.49 (99.72)

Table 3: Track-finding performance (in percentages) of the search by triplet algorithm in Allen versus ETX4VELO for VELO-only particles. The values in parentheses correspond to the performance of the ETX4VELO pipeline without the triplet approach, as currently implemented in C++/CUDA.

throughput is measured using the Nvidia GeForce RTX 2080Ti and GeForce RTX 3090 cards, with 50 repetitions of the pipeline in order to make the impact of I/O overhead on the throughput measurement negligible. The following steps of the pipeline have been implemented: (1) embedding network inference, (2) k-NN algorithm, (3) GNN inference up to the edge classifier, and (4) WCC algorithm. The implementation of track building from edge triplets is left for the future, but is not expected to have a major impact on the computational performance of the pipeline.

	Allen	ETX4VELO	
		With triplets	Without triplets
fake rate	2.18%	1.01%	2.07%

Table 4: Fake rate of the search by triplet algorithm in Allen versus ETX4VELO, for the full pipeline with triplets and the pipeline excluding the triplet approach, as implemented in C++/CUDA.

7.1 Network Inference

To infer the embedding network and the GNN trained in PyTorch, inference engines are utilised. We have tried both ONNX Runtime (ORT) [34] with a CUDA backend and TensorRT (TRT) [35]. Both engines require the corresponding PyTorch networks to be exported to the ONNX open-source format. For TensorRT, memory allocation can be easily delegated to the Allen framework, making it well-suited for production and real-time inference scenarios. In contrast, ONNX Runtime does not readily support this feature. However, ONNX Runtime has the advantage of a CPU backend, making it easy to switch the pipeline from GPU to CPU when needed.

For embedding inference, since Allen dispatches 500 events to each CUDA stream, all the hits from these 500 events are provided directly to ONNX Runtime and TensorRT, maximizing parallelization within the available memory constraints. For GNN inference, events are grouped into batches with a maximum of 2^{20} hits and 2^{22} edges, which is the maximum that GPU memory allows, to accommodate varying event sizes.

Support for the `scatter_add` operation during message building posed challenges for both ONNX Runtime and TensorRT. Fortunately, the latest version of ONNX Runtime (v18) supports this operation. However, for TensorRT, a custom plugin had to be implemented. Although TensorRT 10.0 and later versions support `scatter_add`, the support is limited to specific type combinations. For example, the operation on INT8 types is not supported.

7.2 k-NN Implementation

For each node in plane p , a sequential loop iterates over the hits in planes $p + 1$ and $p + 2$ to compute the squared distance in the embedding space. These iterations are performed in parallel for different hits on a single plane and in parallel for different planes. If the distance is below the squared maximum distance d_{\max}^2 , the hit is added to the list of k_{\max} nearest neighbours. If this list is already full, the maximum distance in the list is found and replaced if the new hit is a closer neighbour, though this situation is infrequent. In fact, less than 0.1% of hits have more than 50 neighbours.

7.3 WCC Implementation

A custom implementation of the Weakly Connected Component (WCC) algorithm is used, leveraging the plane structure of the VELO detector. The purpose of the WCC is to assign a connected component label to each node, indicating the track to which each node belongs. Each node is initially assigned a unique label, typically its own node index. Next the label of each node in plane p is updated in parallel to the lowest label of its connected nodes on the left, sequentially from plane 1 to plane $n_{\text{planes}} - 1 = 25$ (the planes being numbered from 0 to $n_{\text{planes}} - 1$). If a node on the right is connected to more than two nodes on the left, it will only be updated to connect to one of these two nodes, leaving one of the nodes on the left without a proper label. Therefore, it is necessary to repeat the process in reverse, from plane $n_{\text{planes}} - 2 = 24$ to plane 0, considering the connected nodes on the right.

7.4 Quantization

For the quantization to INT8 precision of the embedding MLP model, Nvidia’s `pytorch-quantization`⁷ library, targeting the TensorRT backend, was used. The 8-bit tensor cores, instead of the standard CUDA cores of

⁷<https://docs.nvidia.com/deeplearning/tensorrt/pytorch-quantization-toolkit/docs/index.html>

an Nvidia GPU, are utilized for matrix-multiplication operations yielding more compute throughput. The quantization parameters of the model were calibrated using 5,000 events. The physics performance of the pipeline using the INT8 version of the embedding is shown in Table 5. The fake rate is at 1.72%. For the INT8 case, the rest of the pipeline remains in FP32 precision. The embedding ends up creating roughly an extra 5-10% of edges and thus dilutes the performance of the pipeline.

The quantization of the GNN has not yet been achieved due to the `scatter_add` operation not being natively supported by TensorRT. The custom plugin for this operation was only implemented for single precision. However, in order to use INT8 quantization for the GNN, the plugin has to support this precision also. The quantization of the GNN currently holds the highest promise for throughput gains.

Long	Efficiency		Clone rate		Hit efficiency		Hit purity	
	INT8	FP32	INT8	FP32	INT8	FP32	INT8	FP32
No electrons	97.66	97.96	0.77	0.88	99.95	98.42	98.23	99.95
Electrons	58.50	51.82	2.41	0.93	96.39	96.46	92.39	95.05
From strange	89.03	92.23	1.27	0.61	99.73	96.39	94.07	99.77

Table 5: Track-finding performance (in percentages) of the ETX4VELO pipeline for long particles using the FP32 embedding MLP vs the INT8 version. For the INT8 case, the rest of the pipeline remains in FP32 precision.

7.5 Throughput Results

The current throughput of the pipeline is shown in Table 6 and Table 7 for two different GPU cards. For a visual comparison of the throughput evolution with the algorithm steps on one of these cards, and its comparison to Allen, see Fig. 8. The quoted results are from Allen’s built-in throughput timer. The throughput is measured after various steps of the pipeline, following the pipeline up to a specific step, hence the “Up to step” column. We also give information about the number of streams and the memory used by each stream. The throughput of the ETX4VELO can be separated in 3 columns, “ORT FP32”, “TRT FP32” and “TRT INT8”, where for each we specify the inference engine and precision that was used for the inference of the ML models.

Firstly, the TensorRT implementation in FP32 is significantly faster than the corresponding one in ONNX Runtime. The difference is especially evident in the embedding step, where TensorRT has a throughput of 260k events per second, while ONNX Runtime can process only 46k events per second. Additionally, TensorRT has a lower memory footprint than ONNX Runtime, enabling the GNN to run on multiple streams simultaneously. However, after the k-NN and the GNN, both implementations drop to below 100k, and 1k respectively. On the other hand, the TensorRT implementation with the quantised embedding MLP with INT8 precision, after the embedding step, attains 540k throughput, which is a promising result. After the k-NN however, we see the throughput dropping significantly, namely down to 67k. At present our k-NN implementation is suboptimal as it lacks parallelisation over the neighbours, so there is reason to hope that this throughput loss can be mitigated in the future.

Up to step	# streams	Memory per stream (MB)	Throughput (events/s)		
			ORT FP32	TRT FP32	TRT INT8
VELO decoding	16	500		770k	
Embedding	16	500	46k	260k	540k
k-NN	16	500	28k	53k	67k
GNN	4 (1)	2,000 (9,600)	0.32k	0.86k	-
WCC (VELO tracks)	4 (1)	2,000 (9,600)	0.32k	0.85k	-

Table 6: Throughput of the GPU implementation of ETX4VELO on Nvidia GeForce RTX 2080Ti. The number of streams and memory used for the GNN and WCC step by the ORT pipeline is shown in parentheses. These throughputs should be compared to 530k for the full Allen pipeline ending in VELO tracks.

Up to step	# streams	Memory per stream (MB)	Throughput (events/s)		
			ORT FP32	TRT FP32	TRT INT8
VELO decoding	16	500		1,400k	
Embedding	16	500	54k	330k	820k
k-NN	16	500	38k	81k	93k
GNN	8 (1)	2,500 (9,600)	0.46k	1.4k	-
WCC (VELO tracks)	8 (1)	2,500 (9,600)	0.45k	1.3k	-

Table 7: Throughput of the GPU implementation of ETX4VELO on Nvidia GeForce RTX 3090. The number of streams and memory used for the GNN and WCC step by the ORT pipeline is shown in parentheses. These throughputs should be compared to 860k for the full Allen pipeline ending in VELO tracks.

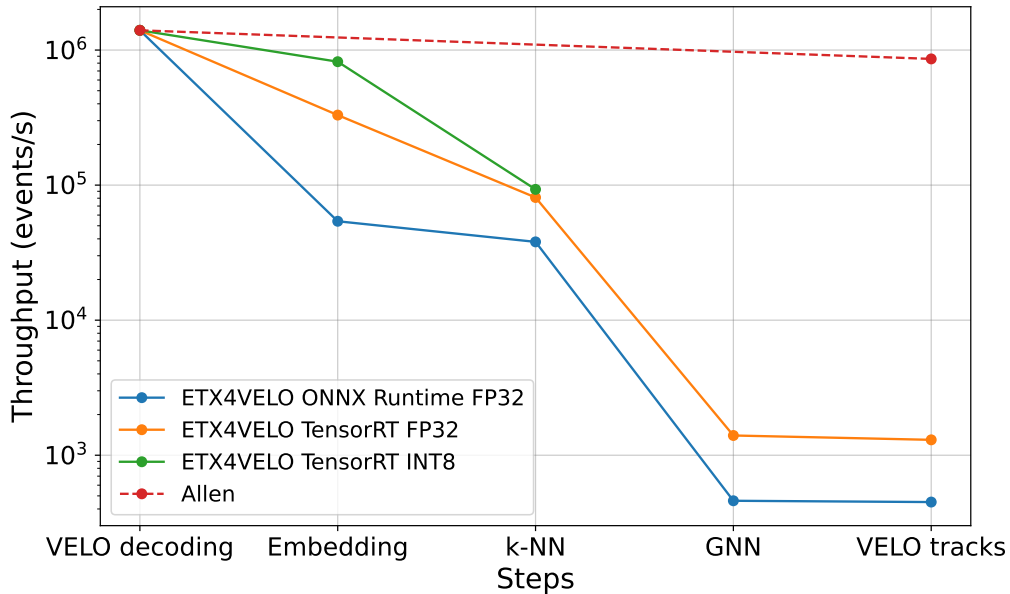


Figure 8: Throughput comparison of track reconstruction in the VELO on an Nvidia GeForce RTX 3090.

8 Conclusions

This work introduced ETX4VELO, a GNN-based pipeline for track reconstruction in the LHCb vertex detector. ETX4VELO has the ability to reconstruct tracks sharing hits through an novel triplet-based approach. When juxtaposed with the default traditional algorithm, ETX4VELO not only matched its efficiency but in some instances, surpassed it.

The focus has now shifted to deployment, and the GNN-based pipeline has been integrated into Allen. The immediate priority is to optimise the inference time, in order to accommodate the high-rate environment that LHCb operates in. This integration is aiming to provide a comprehensive comparison with the default traditional algorithm. Also, it is expected to offer insights that will guide the refinement of various hyperparameters including the MLP and GNN architectures. Our pipeline and general network architecture should be applicable to any LHCb tracking algorithm, although the graph construction and eventual quantisation will vary as a function of the occupancy and number of readout channels in the different parts of the detector.

The code for training and testing ETX4VELO is accessible at <https://gitlab.cern.ch/gdl4hep/etx4velo>. Similarly, the code for the GPU implementation can be found at https://gitlab.cern.ch/gdl4hep/etx4velo_cuda.

ACKNOWLEDGEMENTS

This work is part of the SMARTHEP network and it is funded by the European Union’s Horizon 2020 research and innovation programme, call H2020-MSCA-ITN-2020, under Grant Agreement n. 956086. It is also supported by the ANR-BMBF project ANN4EUROPE ANR-21-FAI1-0011, in collaboration with the Frankfurt Institute for Advanced Studies (FIAS).

The authors extend their sincere gratitude to the LIP6 laboratory for granting access to their versatile GPU cluster, where the majority of the trainings were conducted. We are grateful to the ANN4Europe group at FIAS, led by Ivan Kisel, for their insightful discussions. We thank LHCb’s Real-Time Analysis project for its support, for many useful discussions, and for reviewing an early draft of this manuscript. We also thank Roel Aaij and the LHCb engineering teams for the help in setting up the environment dependencies for performing the inference of the models in Allen as well as the LHCb computing and simulation teams for producing the simulated LHCb samples used to benchmark the performance of the algorithm presented in this paper. The development and maintenance of LHCb’s nightly testing and benchmarking infrastructure which our work relied on is a collaborative effort and we are grateful to all LHCb colleagues who contribute to it.

References

- [1] R. Aaij et al., *Allen: A high level trigger on GPUs for LHCb*, *Comput. Softw. Big Sci.* **4** (2020) 7 [1912.09161].
- [2] LHCb Collaboration, *LHCb Upgrade GPU High Level Trigger Technical Design Report*, 2020. 10.17181/CERN.QDVA.5PIR.
- [3] ALICE collaboration, *ALICE HLT high speed tracking on GPU*, *IEEE Trans. Nucl. Sci.* **58** (2011) 1845.
- [4] ALICE collaboration, *GPU-accelerated track reconstruction in the ALICE High Level Trigger*, *J. Phys. Conf. Ser.* **898** (2017) 032030 [1712.09430].
- [5] ALICE collaboration, *The O2 software framework and GPU usage in ALICE online and offline reconstruction in Run 3*, *EPJ Web Conf.* **295** (2024) 05022 [2402.01205].
- [6] ATLAS collaboration, *Technical Design Report for the Phase-II Upgrade of the ATLAS TDAQ System*, 2017. 10.17181/CERN.2LBB.4IAL.

- [7] C. Collaboration, *The Phase-2 Upgrade of the CMS Data Acquisition and High Level Trigger*, 2021.
- [8] CMS collaboration, *The Phase-2 Upgrade of the CMS Level-1 Trigger*, 2020.
- [9] D.H.C. Pérez, N. Neufeld and A.R. Núñez, *Search by triplet: An efficient local track reconstruction algorithm for parallel architectures*, *Journal of Computational Science* **54** (2021) 101422.
- [10] R. Frühwirth and R.K. Bock, *Data analysis techniques for high-energy physics experiments*, vol. 11, Cambridge University Press (2000).
- [11] T. Golling et al., *TrackML : a tracking Machine Learning challenge*, *PoS ICHEP2018* (2019) 159.
- [12] P. Calafiura et al., *TrackML: A High Energy Physics Particle Tracking Challenge*, in *14th International Conference on e-Science*, p. 344, 2018, DOI.
- [13] S. Amrouche et al., *The Tracking Machine Learning challenge : Accuracy phase*, in *The NeurIPS '18 Competition: From Machine Learning to Intelligent Conversations* (2019), DOI [1904.06778].
- [14] S. Amrouche et al., *The Tracking Machine Learning Challenge: Throughput Phase*, *Comput. Softw. Big Sci.* **7** (2023) 1 [2105.01160].
- [15] N. Choma, D. Murnane, X. Ju, P. Calafiura, S. Conlon, S. Farrell et al., *Track Seeding and Labelling with Embedded-space Graph Neural Networks*, June, 2020. 10.48550/arXiv.2007.00149.
- [16] S. Caillou, C. Collard, C. Rougier, J. Stark, H. Torres and A. Vallier, *Novel fully-heterogeneous GNN designs for track reconstruction at the HL-LHC*, *EPJ Web Conf.* **295** (2024) 09028.
- [17] X. Ju, D. Murnane, P. Calafiura, N. Choma, S. Conlon, S. Farrell et al., *Performance of a geometric deep learning pipeline for HL-LHC particle tracking*, *The European Physical Journal C* **81** (2021) 876.
- [18] A. Correia, F.I. Giasemis, N. Garroum, V.V. Gligorov and B. Granado, *Graph Neural Network-Based Pipeline for Track Finding in the Velo at LHCb*, in *Connecting The Dots 2023 (CTD 2023)*, (Toulouse, France), pp. PROC-CTD2023-34, Oct., 2023, <https://hal.science/hal-04614040>.
- [19] LHCb Collaboration, *Framework TDR for the LHCb Upgrade: Technical Design Report*, 2012.
- [20] LHCb Collaboration, *LHCb VELO Upgrade Technical Design Report*, 2013.
- [21] LHCb Collaboration, *LHCb Tracker Upgrade Technical Design Report* (2014).
- [22] P. Li, E. Rodrigues and S. Stahl, *Tracking Definitions and Conventions for Run 3 and Beyond*, 2021.
- [23] T. Sjöstrand, S. Mrenna and P. Skands, *A brief introduction to PYTHIA 8.1*, *Computer Physics Communications* **178** (2008) 852.
- [24] D.J. Lange, *The EvtGen particle decay simulation package*, *Nucl. Instrum. Meth. A* **462** (2001) 152.
- [25] J. Allison, K. Amako, J. Apostolakis, H. Araujo, P. Arce Dubois, M. Asai et al., *Geant4 developments and applications*, *IEEE Transactions on Nuclear Science* **53** (2006) 270.
- [26] M. Clemencic, G. Corti, S. Easo, C.R. Jones, S. Miglioranzi, M. Pappagallo et al., *The LHCb Simulation Application, Gauss: Design, Evolution and Experience*, *Journal of Physics: Conference Series* **331** (2011) 032023.
- [27] R. Tarjan, *Depth-first search and linear graph algorithms*, *SIAM Journal on Computing* **1** (1972) 146 [<https://doi.org/10.1137/0201010>].
- [28] J. Johnson, M. Douze and H. Jegou, *Billion-Scale Similarity Search with GPUs*, *IEEE Transactions on Big Data* **7** (2021) 535.

- [29] C. Cortes and V. Vapnik, *Support-vector networks*, *Mach. Learn.* **20** (1995) 273–297.
- [30] T.-Y. Lin, P. Goyal, R. Girshick, K. He and P. Dollár, *Focal Loss for Dense Object Detection*, Feb., 2018. 10.48550/arXiv.1708.02002.
- [31] ATLAS collaboration, J.D. Burlison, S. Caillou, P. Calafiura, J. Chan, C. Collard, X. Ju et al., *Physics Performance of the ATLAS GNN4ITk Track Reconstruction Chain*, 2023.
- [32] M. Paterno, *Calculating efficiencies and their uncertainties*, Dec., 2004. 10.2172/15017262.
- [33] R. Brun and F. Rademakers, *ROOT — An object oriented data analysis framework*, *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* **389** (1997) 81.
- [34] ONNX Runtime developers, *ONNX Runtime*, 2021.
- [35] TensorRT developers, *NVIDIA TensorRT*, 2024.