Computing and Software for Big Science (2021) 5:26
https://doi.org/10.1007/s41781-021-00073-z

ORIGINAL ARTICLE

Charged Particle Tracking via Edge-Classifying Interaction Networks

Gage DeZoort[1] · Savannah Thais[1] · Javier Duarte[2] · Vesal Razavimaleki[2] · Markus Atkinson[3] · Isobel Ojalvo[1] · Mark Neubauer[3] · Peter Elmer[1]

Received: 12 July 2021 / Accepted: 13 October 2021 / Published online: 15 November 2021

Abstract
Recent work has demonstrated that geometric deep learning methods such as graph neural networks (GNNs) are well suited to address a variety of reconstruction problems in high-energy particle physics. In particular, particle tracking data are naturally represented as a graph by identifying silicon tracker hits as nodes and particle trajectories as edges, given a set of hypothesized edges, edge-classifying GNNs identify those corresponding to real particle trajectories. In this work, we adapt the physics-motivated interaction network (IN) GNN toward the problem of particle tracking in pileup conditions similar to those expected at the high-luminosity Large Hadron Collider. Assuming idealized hit filtering at various particle momenta thresholds, we demonstrate the IN's excellent edge-classification accuracy and tracking efficiency through a suite of measurements at each stage of GNN-based tracking: graph construction, edge classification, and track building. The proposed IN architecture is substantially smaller than previously studied GNN tracking architectures; this is particularly promising as a reduction in size is critical for enabling GNN-based tracking in constrained computing environments. Furthermore, the IN may be represented as either a set of explicit matrix operations or a message passing GNN. Efforts are underway to accelerate each representation via heterogeneous computing resources towards both high-level and low-latency triggering applications.

Keywords Graph neural networks · Tracking · Particle physics

Introduction

Charged particle tracking is essential to many physics reconstruction tasks including vertex finding [1, 2], particle reconstruction [3, 4], and jet flavor tagging [5–7]. Current tracking algorithms at the CERN Large Hadron Collider (LHC) experiments [2, 8] are typically based on the combinatorial Kalman filter [9–12] and have been shown to scale worse than linearly with increasing beam intensity and detector occupancy [13]. The high-luminosity phase of the LHC (HL-LHC) will see an order of magnitude increase in luminosity [14], highlighting the need to develop new tracking algorithms demonstrating reduced latency and improved performance in high-pileup environments. To this end, ongoing research focuses on both accelerating current tracking algorithms via parallelization or dedicated hardware and developing new tracking algorithms based on machine learning (ML) techniques.

Geometric deep learning (GDL) [15–18] is a growing sub-field of ML focused on learning representations on non-Euclidean domains, such as sets, graphs, and manifolds. Graph neural networks (GNNs) [19–24] are the subset of GDL algorithms that operate on graphs, data represented as a set of nodes connected by edges, and have been explored for a variety of tasks in high energy physics [25, 26]. Particle tracking data are naturally represented as a graph; detector hits form a 3D point cloud and the edges between them represent hypotheses about particle trajectories. Recent progress by the Exa.TrkX project and other collaborations has

S. T. and V. R. are supported by IRIS-HEP through the U.S. National Science Foundation (NSF) under Cooperative Agreement OAC-1836650. J. D. is supported by the U.S. Department of Energy (DOE), Office of Science, Office of High Energy Physics Early Career Research program under Award No. DE-SC0021187. G. D. is supported by DOE Award No. DE-SC0007968.

✉ Gage DeZoort
jdezoort@princeton.edu

✉ Savannah Thais
sthais@princeton.edu; savannah.jennifer.thais@cern.ch

[1] Princeton University, Princeton, NJ, USA

[2] University of California San Diego, La Jolla, CA, USA

[3] University of Illinois at Urbana-Champaign, Champaign, IL, USA

demonstrated that edge-classifying GNNs are well suited to particle tracking applications [27–30]. Tracking via edge classification typically involves three stages. In the graph construction stage, silicon tracker hits are mapped to nodes and an edge-assignment algorithm forms edges between certain nodes. In the edge classification stage, an edge-classifying GNN infers the probability that each edge corresponds to a true track segment meaning that both hits (nodes connecting the edge) are associated to the same truth particle, as discussed further in Sect. 4.1. Finally, in the track building step, a track-building algorithm leverages the edge weights to form full track candidates.

In this work, we present a suite of measurements at each of these stages, exploring a range of strategies and algorithms to facilitate GNN-based tracking. We focus in particular on the interaction network (IN) [22], a GNN architecture frequently used as a building block in more complicated architectures [27, 29, 31, 32]. The IN itself demonstrates powerful edge-classification capability and its mathematical formulations are the subject of ongoing acceleration studies [33]. In Sect. 2, we first present an overview of particle tracking and graph-based representations of track hits. In Sect. 3, we introduce INs and describe the mathematical foundations of our architecture. In Sect. 4, we present specific graph construction, IN edge classification, and track building measurements on the open-source TrackML dataset. Additionally, we present IN inference time measurements, framing this work in the context of ongoing GNN acceleration studies. In Sect. 5, we summarize the results of our studies and contextualize them in the broader space of ML-based particle tracking. We conclude in the same section with outlook and discussion of future studies, in particular highlighting efforts to accelerate INs via heterogeneous computing resources.

Theory and Background

Particle Tracking

In collider experiments, such as the LHC, charged particle trackers are composed of cylindrical detector layers immersed in an axially-aligned magnetic field. The detector geometry is naturally described by cylindrical coordinates (r, ϕ, z), where the z-axis is aligned with the beamline. Pseudorapidity is a measure of angle with respect to the beamline, defined as $\eta := -\log\tan\frac{\theta}{2}$ where θ is the polar angle. Charged particles produced in collision events move in helical trajectories through the magnetic field, generating localized hits in the tracker layers via ionization energy deposits. Track reconstruction consists of "connecting the dots," wherein hits are systematically grouped to form charged particle trajectories. We refer to a pair of hits that belong to the same particle as a track segment, such that the line extending between the hits is a linear approximation of the particle's trajectory. Note that in a high-pileup scenario, track hits might correspond to multiple overlapping particle trajectories. Reconstructed tracks are defined by their respective hit patterns and kinematic properties, which are extracted from each track's helix parameters. Specifically, initial position and direction follow directly from helical fits and the transverse momentum p_T is extracted from the track's curvature (see Fig. 1) [34].

In this work, we focus specifically on track building in the pixel detector (see Fig. 2), the innermost subdetector of the tracker. Many tracking algorithms run "inside out," where track seeds from the pixel detector are used to estimate initial track parameters and propagated through the full detector [2]. Improving the seeding stage of the

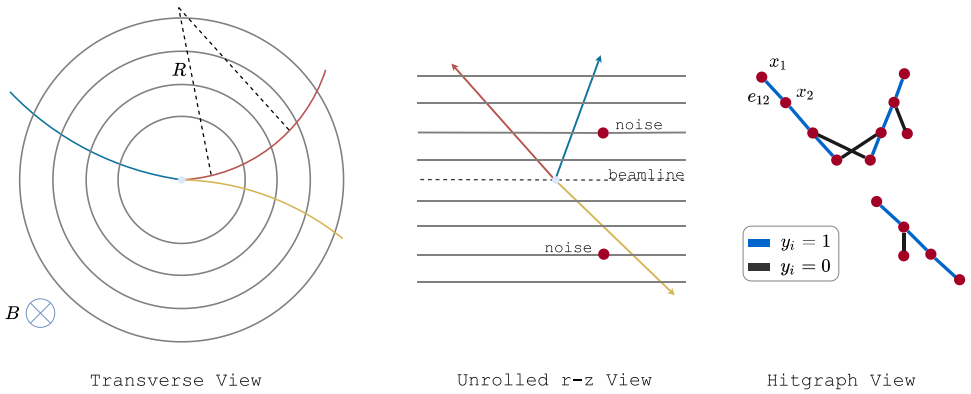

Fig. 1 (Left) A transverse view of a generic particle tracker, where the z-axis points out of the page. Here, we see a set of four cylindrical detector layers with three particles traversing them. The magnetic field (of strength B) is aligned with the z-axis such that tracks move with a radius of curvature R in the transverse plane, yielding measurements of transverse momentum via $p_T = 0.3\,[\frac{\text{GeV}}{\text{T·m}}]\,BR$. (Middle) The four cylindrical tracker layers are "unrolled" in the r–z plane to show the full event contents: three particles plus additional noise hits. (Right) The corresponding hitgraph is shown with example node and edge labels

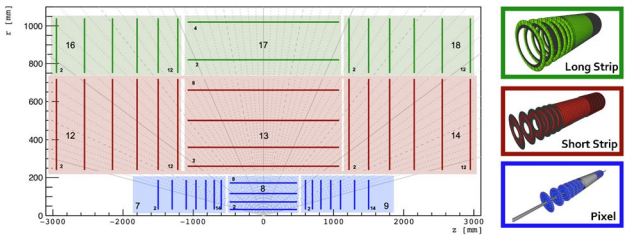

Fig. 2 Here we depict a particle tracker geometry similar to tracker designs proposed for the HL-LHC era. This "generic tracker" geometry is used in the TrackML dataset (see Sect. 4). The generic tracker is composed of three sub-detectors named by the shape of their silicon modules: the pixel detector, the short strip detector, and the long strip detector. Each sub-detector is divided into volumes (numbered 7-18); each volume contains a set of detector layers. Volumes 8, 13, and 17 above are referred to as the *barrel* of the detector because their layers sit at a constant cylindrical radius. Volumes 7, 9, 12, 14, 16, and 18 comprise the tracker's *endcaps* because of their disk-like shape; endcap layers are positioned at a single point along the z-axis. The above figure is adapted from a figure in Ref. [35] and the TrackML detector diagram accompanying Kaggle's TrackML dataset

tracking pipeline is an important step towards enabling efficient tracking at the HL-LHC; this approach is complimentary to other GNN-based tracking efforts that focus on the full tracker barrel (without including endcaps) using graph segmentation [28].

Tracker Hits as Graphs

Tracking data is naturally represented as a graph by identifying hits as nodes and track segments as (in general) directed edges (see Fig. 1). In this scheme, nodes have cylindrical spatial features $x_k = (r_k, \phi_k, z_k)$ and edges are defined by the nodes they connect. We employ two different edge representations: 1) binary incidence matrices $R_i, R_o \in \{0,1\}^{n_{\text{edges}} \times n_{\text{nodes}}}$ in incoming/outgoing (IO) format and 2) hit index pair lists $I \in \mathbb{N}^{2 \times n_{edges}}$ in coordinate (COO) format [36]. Specifically, the incidence matrix elements $(R_i)_{e,h}$ are 1 if edge e is incoming to hit h and 0 otherwise; R_o is defined similarly for outgoing edges. COO entries $I_{0,e}$ and $I_{1,e}$ are the hit indices from which edge e is outgoing from and incoming to, respectively. Each edge is assigned a set of geometric features $a_{ij} = (\Delta r_{ij}, \Delta\phi_{ij}, \Delta z_{ij}, \Delta R_{ij})$, where $\Delta R_{ij} = \sqrt{\Delta\eta_{ij}^2 + \Delta\phi_{ij}^2}$ is the edge length in η-ϕ space. Node and edge features are stacked into matrices $X = [x_k] \in \mathbb{R}^{n_{\text{nodes}} \times 3}$ and $R_a = [a_{ij}] \in \mathbb{R}^{n_{\text{edges}} \times 4}$. Accordingly, we define *hitgraphs* representing tracking data as $\mathcal{G}_{\text{IO}} := (X, R_a, R_i, R_o)$ and $\mathcal{G}_{\text{COO}} := (X, R_a, I)$. The corresponding training target is the vector $y \in \mathbb{R}^{n_{\text{edges}}}$, whose components y_e are 1 when edge e connects two hits associated to the same particle and 0 otherwise.

Interaction Networks

The IN is a physics-motivated GNN capable of reasoning about objects and their relations [22]. Each IN forward-pass involves a relational reasoning step, in which an interaction is computed, and an object reasoning step, in which interaction effects are aggregated and object dynamics are applied. The resulting predictions have been shown to generate next-timestep dynamics consistent with various physical principles. We adapt the IN to the problem of edge classification by conceptualizing each hitgraph as a complex network of hit "objects" and edge "relations." In this context, the relational and object reasoning steps correspond to edge and node re-embeddings, respectively. In an edge classification scheme, the IN must determine whether or not each edge represents a track segment. Accordingly, we extend the IN forward pass to include an additional relational reasoning step, which produces an edge weight for each edge in the hitgraph. We consider two formulations of the IN: (1) the matrix formulation, suitable for edge-classification on $\mathcal{G}_{\text{IO}}$ defined via PyTorch [37] and (2) the message passing formulation, suitable for edge-classification on $\mathcal{G}_{\text{COO}}$ defined via PyTorch Geometric (PyG) [36]. These formulations are equivalent in theory, but specific implementations and training procedures can vary their computational and physics performance. In particular, the COO encoding of the edge adjacency can greatly reduce the memory footprint for training. For this reason, the measurements performed in this paper are based on the message passing formulation. In Sect. 3.1, we review the matrix formulation as presented in the original IN paper [22], subsequently expanding the notation to describe the message passing IN formulation in Sect. 3.2.

Matrix Formulation

The original IN was formulated using simple matrix operations interpreted as a set of physical interactions and effects [22]. The forward pass begins with an input hitgraph $\mathcal{G}_{IO} = (X, R_a, R_i, R_o)$. The hits receiving an incoming edge are given by $X_i := R_i X \in \mathbb{R}^{n_{\text{edges}} \times 3}$; likewise, the hits sending an outgoing edge are given by $X_o := R_o X \in \mathbb{R}^{n_{\text{edges}} \times 3}$. Interaction terms are defined by the concatenation $m(\mathcal{G}_{IO}) := [X_i, X_o, R_a] \in \mathbb{R}^{n_{\text{edges}} \times 10}$, known as the marshalling step. A relational network $\phi_{R,1}$ predicts an effect for each interaction term, $E := \phi_{R,1}\left(m(\mathcal{G}_{IO})\right) \in \mathbb{R}^{n_{\text{edges}} \times 4}$. These effects are aggregated via summation for each receiving node, $A := a(\mathcal{G}_{IO}, E) = R_i^T E \in \mathbb{R}^{n_{\text{nodes}} \times 4}$, and concatenated with X to form a set of expanded hit features $C := [X, A] \in \mathbb{R}^{n_{\text{nodes}} \times 7}$. An object network ϕ_O re-embeds the hit positions as $\tilde{X} := \phi_O(C) \in \mathbb{R}^{n_{\text{nodes}} \times 3}$. At this point, the

traditional IN inference is complete, having re-embedded both the edges and nodes. Accordingly, we denote the re-embedded graph $\mathrm{IN}(\mathcal{G}_{IO}) = \tilde{\mathcal{G}}_{IO} = (\tilde{X}, E, R_i, R_o)$.

To produce edge weights, an additional relational reasoning step is performed on $\tilde{\mathcal{G}}_{IO}$. Re-marshalling yields new interaction terms $m(\tilde{\mathcal{G}}_{IO}) = [\tilde{X}_i, \tilde{X}_o, E] \in \mathbb{R}^{n_{\text{edges}} \times 10}$ and a second relational network $\phi_{R,2}$ predicts edge weights for each edge: $W(\mathcal{G}_{IO}) := \phi_{R,2}(m(\tilde{\mathcal{G}}_{IO})) \in (0,1)^{n_{\text{edges}}}$. Summarily, we have a full forward pass of the edge classification IN:

$$W(\mathcal{G}_{IO}) = \phi_{R,2}\Big[m\big(\mathrm{IN}(\mathcal{G}_{IO})\big)\Big]. \tag{1}$$

Message Passing Formulation

The message passing NN (MPNN) framework summarizes the behavior of a range of GNN architectures including the IN [21]. In general, MPNNs update node features by aggregating "messages," localized information derived from the node's neighborhood, and propagating them throughout the graph. This process is iterative; given a message passing time T indexed by $t \in \mathbb{N}$, a generic message passing node update can be written as follows:

$$x_i^{(t)} = \phi_{\text{node}}^{(t)}\left(x_i^{(t-1)}, \square_{j \in N(i)} \phi_{\text{message}}^{(t)}\left(x_i^{(t-1)}, x_j^{(t-1)}, a_{ij}^{(t-1)}\right)\right). \tag{2}$$

Here, $N(i)$ is neighborhood of node i. The differentiable function $\phi_{\text{message}}^{(t)}$ calculates messages for each $j \in N(i)$, which are aggregated across $N(i)$ by a permutation-invariant function $\square$. A separate differentiable function $\phi_{\text{node}}^{(t)}$ leverages the aggregated messages to update the node's features. Given this generalized MPNN, the IN follows from the identifications $\phi_{\text{message}} \to \phi_{R,1}$, $\square_{j \in N(i)} \to \sum_{j \in N(i)}$, and $\phi_{\text{node}} \to \phi_O$ for a single timestep ($T = 1$):

$$a_{ij}^{(1)} = \phi_{R,1}\left(x_i^{(0)}, x_j^{(0)}, a_{ij}^{(0)}\right), \tag{3}$$

$$x_i^{(1)} = \phi_O\left(x_i^{(0)}, \sum_{j \in N(i)} a_{ij}^{(1)}\right). \tag{4}$$

An additional relational reasoning step gives edge weights

$$w_{ij}^{(1)} := \phi_{R,2}\left(x_i^{(1)}, x_j^{(1)}, a_{ij}^{(1)}\right). \tag{5}$$

In this way, we produce edge weights $W(\mathcal{G}_{\text{COO}}) = [w_{ij}^{(1)}]$ from the re-embedded graph with node features $\tilde{X} = [x_i^{(1)}]$ and edge features $E = [a_{ij}^{(1)}]$. This formulation is easily generalized to $T > 1$ by applying Eqs. 3 and 4 in sequence at each time step before finally calculating edge weights via Eq. 5 at time T. In the following studies, we focus on the simplest case of nearest-neighbor message passing ($T = 1$).

Measurements

TrackML Dataset

The TrackML dataset is a simulated set of proton-proton collision events originally developed for the TrackML Particle Tracking Challenge [35]. TrackML events are generated with 200 pileup interactions on average, simulating the high-pileup conditions expected at the HL-LHC. Each event contains 3D hit position and truth information about the particles that generated them. In particular, particles are specified by particle IDs (p_{ID}) and three-momentum vectors ($\mathbf{p}$). Each simulated hit has a unique identifier assigned that gives the true hit position and which particle created the hit. For this truth assignment, no merging of reconstructed hits is considered as merging of hits occurs in less than 0.5% of the cases and the added complexity was deemed unnecessary for the original challenge. Other simplifications in this dataset include a simple geometry with modules arranged in cylinders and disks, instead of a more complex geometry with cones, no simulation of electronics, cooling tubes, and cables, and only one type of physics process (top quark-antiquark pairs) instead of a variety of processes.

The TrackML detector is designed as a generalized LHC tracker; it contains discrete layers of sensor arrays immersed in a strong magnetic field. We focus specifically on the pixel layers, a highly-granular set of four barrel and fourteen endcap layers in the innermost tracker regions. The pixel layers are shown in Fig. 2. We note that constraining our studies to the pixel layers reduces the size of the hitgraphs such that they can be held in memory and processed by the GNN without segmentation.

Graph Construction

In the graph construction stage, each event's tracker hits are converted to a hitgraph through an edge selection algorithm. Typically, a set of truth filters are applied to hits before they are assigned to graph nodes. For example, p_{T} *filters* reject hits generated by particles with $p_{\text{T}} < p_{\text{T}}^{\text{min}}$, *noise filters* reject noise hits, and *same-layer filters* reject all but one hit per layer for each particle. These truth filters are used to modulate the number of hits present in each hit graph to make it more feasible to apply GNN methods and can be thought of as an idealized hit filtering step (see Table 1). One goal of future R&D is to lower or remove this truth-based filter or replace it with a realistic hit filtering step that could be applied in a high-pileup experimental setting. After initial hit filtering yields a set of nodes, edge-assignment algorithms extend edges between certain nodes. These edges are inputs to the

Table 1 The p_T, noise, and same-layer filters are used as a handle on graph size by reducing the number of hits allowed into the graph. Here, we profile 100 events from the TrackML `train_1` sample; these events have an average of $N(\text{total}) = 56751 \pm 6070$ hits in the pixel detector. Denote the hits removed by the p_T, noise, and same-layer filters as $N(p_T < p_T^{\text{min}})$, $N(\text{noise})$ and $N(\text{same}-\text{layer})$, respectively. The noise filter is observed to remove $N(\text{noise}) = 3702 \pm 56$ hits, roughly 6.5% of the detector occupancy. The p_T and same-layer filters remove hits as a function of p_T^{min}; these values are reported in the table below. We define $N(\text{remaining}) := N(\text{total}) - N(p_T < p_T^{\text{min}}) - N(\text{same}-\text{layer}) - N(\text{noise})$ to be the hits remaining after these filters are applied; $N(\text{remaining})$ corresponds to n_{nodes} constructed in the hitgraph

p_T^{min} [GeV]	$N(p_T < p_T^{\text{min}})$	$N(\text{same}-\text{layer})$	$N(\text{remaining})$	$N(\text{remaining})/N(\text{total})$ [%]
2.0	51520 ± 5848	439 ± 87	1090 ± 156	1.9 ± 0.3
1.5	49880 ± 5617	921 ± 159	2248 ± 155	4.0 ± 0.5
1.0	45501 ± 5057	2233 ± 333	5315 ± 152	9.4 ± 1.0
0.9	43839 ± 4855	2735 ± 395	6475 ± 151	11.4 ± 1.2
0.8	41677 ± 4583	3396 ± 480	7976 ± 150	14.1 ± 1.5
0.7	38778 ± 4242	4278 ± 582	9993 ± 148	17.6 ± 1.9
0.6	34951 ± 3798	5448 ± 714	12650 ± 146	22.3 ± 2.4
0.5	29830 ± 3265	7025 ± 875	16194 ± 144	28.5 ± 3.1

inference stage and must therefore represent as many true track segments as possible. Naively, one might return a fully-connected hitgraph. However, this strategy yields $\frac{1}{2} n_{\text{nodes}}(n_{\text{nodes}} - 1)$ edges, which for $n_{\text{nodes}} \sim \mathcal{O}(1000)$ gives $n_{\text{edges}} \sim \mathcal{O}(500,000)$. This represents a fundamental tradeoff between different edge-assignment algorithms: they must simultaneously maximize *efficiency*, the fraction of track segments represented as true edges, and *purity*, the fraction of true edges to total edges in the hitgraph.

In this work, we compare multiple graph construction algorithms, each of which determines whether or not to extend an edge with features a_{ij} between hits i and j. In all methods, only pixel detector hits are considered, pseudorapidity is restricted to $\eta \in [-4, 4]$, and the noise and same-layer hit filters are applied. Each method has the same definition of graph construction efficiency ($N_{\text{true}}^{\text{reconstructed}} / N_{\text{true}}^{\text{possible}}$) and purity ($N_{\text{true}}^{\text{reconstructed}} / N_{\text{total}}^{\text{reconstructed}}$). The denominator quantity $N_{\text{true}}^{\text{possible}}$ is independent of the graph construction algorithm such that one may directly compare the efficiencies of the various methods. On the other hand, the denominator $N_{\text{total}}^{\text{reconstructed}}$ depends on the specific graph construction routine; for this reason, it is important to study purity in the context of efficiency. The same-layer filter introduces an ambiguity in defining edges between the barrel and innermost endcap layers. Specifically, barrel hits generated by the same particle could produce multiple true edges incoming to a single endcap hit. The resulting triangular edge pattern conflicts with the main assumption of the same-layer filter, that only one true track segment exists between each subsequent layer. For this reason, a *barrel intersection* cut was developed, in which edges between a barrel layer and an innermost endcap layer are rejected if they intersect with any intermediate barrel layers (see Fig. 3).

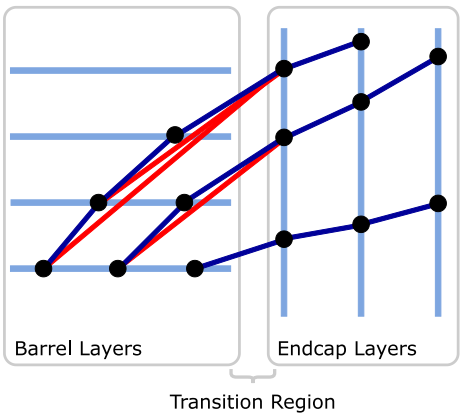

Fig. 3 The transition region between the barrel and endcaps introduces an ambiguity in truth-labeling edges crossing from barrel to endcap layers. Specifically, one may draw multiple possible edges between hits in barrel layers and the innermost endcap layer. Only one such edge can be true; the others (labeled red) should be rejected. The barrel intersection cut rejects any edges between a barrel layer and an innermost endcap layer that intersect an intermediate barrel layer. Accordingly, the red edges would be rejected by the intersecting line cut and the blue edges would not

In addition to the barrel intersection cut, edges must also satisfy p_T^{min}-dependent constraints on the geometric quantities $z_0 = z_i - r_i \frac{z_j - z_i}{r_j - r_i}$ and $\phi_{\text{slope}} = \frac{\phi_j - \phi_i}{r_j - r_i}$. These selections form the basis of each of the following graph construction algorithms:

1. *Geometric* Edges must satisfy the barrel intersection cut and z_0 and ϕ_{slope} constraints.
2. *Geometric and preclustering* In addition to all geometric selections, edges must also belong to the same cluster in

η-ϕ space determined by the density-based spatial clustering of applications with noise (DBSCAN) algorithm [38].

3. *Geometric and data-driven* In addition to all geometric selections, edges must connect detector modules that have produced valid track segments in an independent data sample; this data-driven strategy is known as the *module map* method originally developed in [39].

Truth-labeled example graphs and key performance metrics for each graph construction algorithm are shown in Figs. 4 and 5, respectively. For each method, $p_{\mathrm{T}}^{\mathrm{min}}$-dependent values of ϕ_{slope} and z_0 are chosen to keep the efficiency at a constant $\mathcal{O}(99\%)$. We observe a corresponding drop in purity to $\mathcal{O}(1\%)$ as $p_{\mathrm{T}}^{\mathrm{min}}$ is decreased and graphs become denser. At high values of $p_{\mathrm{T}}^{\mathrm{min}}$, preclustering hits in η–ϕ space yields a significant increase in purity over the purely geometric construction. This effect disappears as $p_{\mathrm{T}}^{\mathrm{min}}$ decreases below 1.5 GeV, as tracks begin to overlap non-trivially with higher detector occupancy. On the other hand, the data-driven module map yields a significant boost in purity for the full range of $p_{\mathrm{T}}^{\mathrm{min}}$. Accordingly, the module map method is most suited to constrained computing environments in which graph size or processing time is limited. It should be noted, however, that purer graphs do not necessarily lead to higher edge classification accuracies.

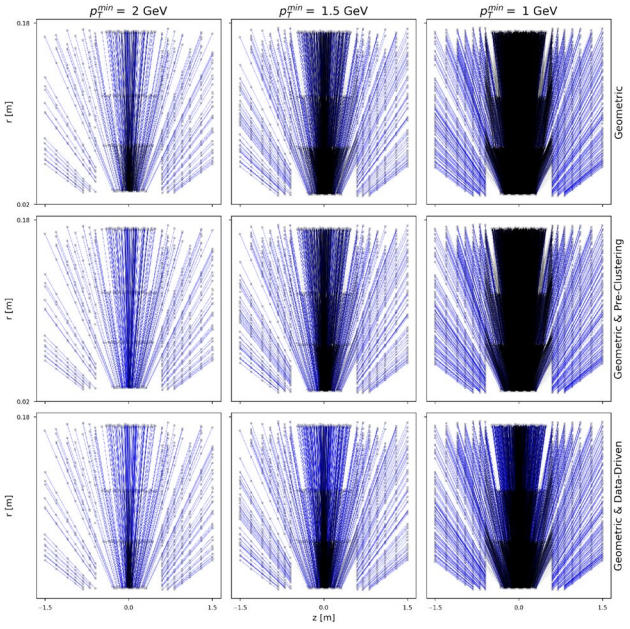

Fig. 4 Edge colors indicate truth labels; blue edges are true track segments and back edges are false. Varying $p_{\mathrm{T}}^{\mathrm{min}}$ modulates the graph size. As $p_{\mathrm{T}}^{\mathrm{min}}$ is decreased, graphs are increasingly composed of false edges. Preclustering and data-driven edge selections reduce the fraction of false edges in the graphs when compared to simple geometric selections

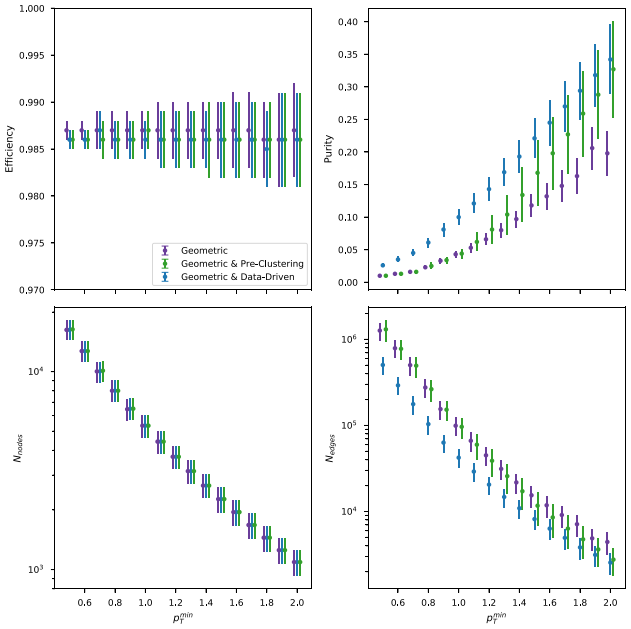

Fig. 5 Graph construction efficiency, purity, node counts, and edge counts are reported for a range of $p_{\mathrm{T}}^{\mathrm{min}}$ calculated using 100 random graphs from the `train_1` sample

Edge Classification

As detailed in Sect. 3, we have implemented the IN in PyTorch [37] as a set of explicit matrix operations and in PyG [36] as a MPNN. Both implementations are available in the Git repository accompanying this paper [40]. In the following studies, we limit our focus to the MPNN implementation trained on graphs built using geometric cuts only. Because PyG accommodates the sparse $\mathcal{G}_{\mathrm{COO}}$ edge representation, the MPNN implementation is significantly faster and more flexible than the matrix implementation (see 4.5). The full forward-pass, composed of edge and node blocks used to predict edge weights, is shown in Fig. 6. The functions $\phi_{R,1}$, $\phi_{R,2}$, and ϕ_O are approximated as multilayer perceptrons (MLPs) with rectified linear unit (ReLU) activation functions [41, 42]. The ReLU activation function behaves as an identity function for positive inputs and saturates at 0 for negative inputs. Notably, the $\phi_{R,2}$ outputs have a sigmoid activation $\sigma(\cdot) \in (0, 1)$, such that they represent probabilities, or edge weights, $W(\mathcal{G}_{\mathrm{COO}}) \in (0, 1)^{n_{\mathrm{edges}}}$ that each edge is a track segment. We therefore seek to optimize a binary cross-entropy (BCE) loss between the truth targets $y_k = \{0, 1\}$ and edge weights $w_k \in (0, 1)$, which henceforth are re-labeled by the edge index k:

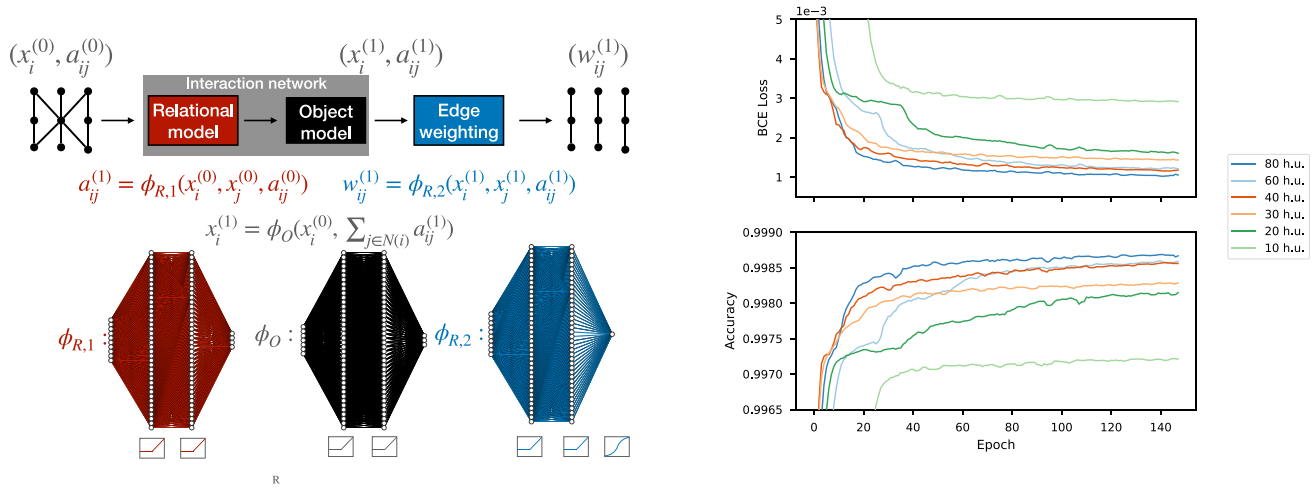

Fig. 6 (Left) The complete IN forward-pass with the relational and object models approximated as MLPs. (Right) An example hyperparameter scan in which a models with varying numbers of hidden units (h.u.) were trained on $p_{\mathrm{T}}^{\mathrm{min}} = 0.7\,\mathrm{GeV}$ graphs

$$\ell\left(y_n, W_n(\mathcal{G})\right) = -\sum_{k=1}^{n_{\mathrm{edges}}} \left(y_k \log w_k + (1-y_k)\log(1-w_k)\right) \tag{6}$$

Here, n is the sample index so that the total loss per epoch is the average BCE loss $L(\{\mathcal{G}_n, y_n\}_{n=1}^{N}) = \frac{1}{N}\sum_{n=1}^{N} \ell\left(y_n, W_n(\mathcal{G})\right)$. Throughout the following studies, the architecture in Fig. 6 is held at a constant size of 6,448 trainable parameters, corresponding to 40 hidden units (h.u.) per layer in each of the MLPs. Validation studies indicate that even this small network rapidly converged to losses of $\mathcal{O}(10^{-3})$, similar to its larger counterparts (see Fig. 6). Assuming every MLP layer has the same number of h.u., 40 h.u. per layer is sufficient to recover the maximum classification accuracy with models trained on $p_{\mathrm{T}}^{\mathrm{min}} = 1\,\mathrm{GeV}$ graphs. In the following studies, models are trained on graphs built with $p_{\mathrm{T}}^{\mathrm{min}}$ ranging from 0.6–2 GeV. At each value of $p_{\mathrm{T}}^{\mathrm{min}}$, 1500 graphs belonging to the TrackML `train_1` sample are randomly divided into 1000 training, 400 testing, and 100 validation sets. The Adam optimizer is used to facilitate training [43]. It is configured with learning rates of $3.5\text{–}8\times10^{-3}$, which are decayed by a factor of $\gamma = 0.95$ for $p_{\mathrm{T}}^{\mathrm{min}} \leq 1\,\mathrm{GeV}$ and $\gamma = 0.8$ for $p_{\mathrm{T}}^{\mathrm{min}} > 1\,\mathrm{GeV}$ every 10 epochs.

To evaluate the IN edge-classification performance, it is necessary to define a threshold δ such that each edge weight $w_k \in W(\mathcal{G}_{COO})$ satisfying $w_k \geq \delta$ or $w_k < \delta$ indicates that edge k was classified as true or false, respectively. Here, we define δ^* as the threshold at which the true positive rate (TPR) equals the true negative rate (TNR). In principle, δ^* may be calculated individually for each graph. However, this introduces additional overhead to the inference step, which is undesirable in constrained computing environments. We instead determine δ^* during the training process by minimizing the difference $|\mathrm{TPR} - \mathrm{TNR}|$ for graphs in the validation set. The resulting δ^*, which is stored for use in evaluating the testing sample, represents the average optimal threshold for the validation graphs. Accordingly, we define the model's accuracy at δ^* as $(n_{\mathrm{TP}} + n_{\mathrm{TN}})/n_{\mathrm{edges}}$, where n_{TP} (n_{TN}) is the number of true positives (negatives), and note that the BCE loss is independent of δ^*.

As shown in Fig. 7, the training process results in smooth convergence to excellent edge-classification accuracy for a range of $p_{\mathrm{T}}^{\mathrm{min}}$. Classification accuracy degrades slightly as $p_{\mathrm{T}}^{\mathrm{min}}$ is lowered below 1 GeV; hyperparameter studies indicate that larger networks improve performance on lower $p_{\mathrm{T}}^{\mathrm{min}}$ graphs (see Fig. 6). A transfer learning study was conducted in which models trained on graphs at a specific $p_{\mathrm{T}}^{\mathrm{min}}$ were tested on graph samples at a range of $p_{\mathrm{T}}^{\mathrm{min}}$. The results are summarized in Fig. 8, which shows that the models achieve relatively robust performance on a range of graph sizes. These results suggest it may be possible to train IN models in simplified scenarios and apply them to more complex realistic scenarios (e.g. without a $p_{\mathrm{T}}^{\mathrm{min}}$ cut).

Track Building

In the track building step, the predicted edge weights $w_k \in W(\mathcal{G}_{\mathrm{COO}})$ are used to infer that edges satisfying $w_k \geq \delta^*$ represent true track segments. If the edge weight mask perfectly reproduced the training target (i.e. $\mathtt{int}(W(\mathcal{G}_{\mathrm{COO}}) \geq \delta^*) = y$), the edge-classification step would produce $n_{\mathrm{particles}}$ disjoint subgraphs, each corresponding to a single particle. Imperfect edge-classification leads to spurious connections between these subgraphs, prompting the need for more sophisticated track-building algorithms. Here, we use the union-find algorithm [44] and DBSCAN to cluster hits in the edge-weighted graphs. Hit clusters are then considered to be reconstructed tracks candidates; the track

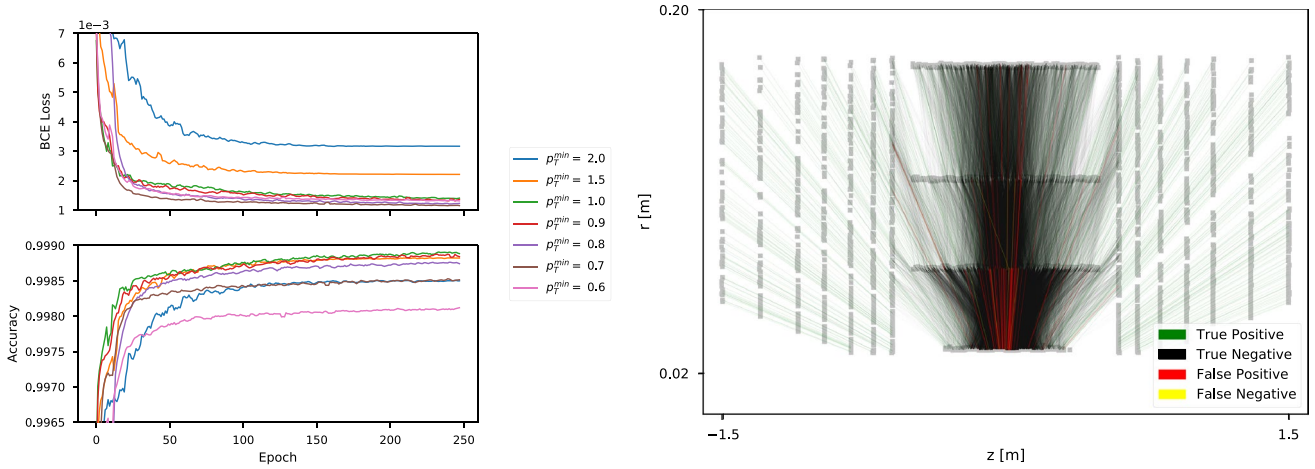

Fig. 7 (Left) Loss convergence for models trained on various $p_{\mathrm{T}}^{\mathrm{min}}$ graphs. (Right) A model trained on $p_{\mathrm{T}}^{\mathrm{min}} = 1\,\mathrm{GeV}$ graphs was used to evaluate an unseen $p_{\mathrm{T}}^{\mathrm{min}} = 1\,\mathrm{GeV}$ graph, yielding a loss of 1.52×10^{-3} and accuracy of 99.9%. 98 out of 95,160 edges were incorrectly classified; these erroneous classifications are magnified in the figure

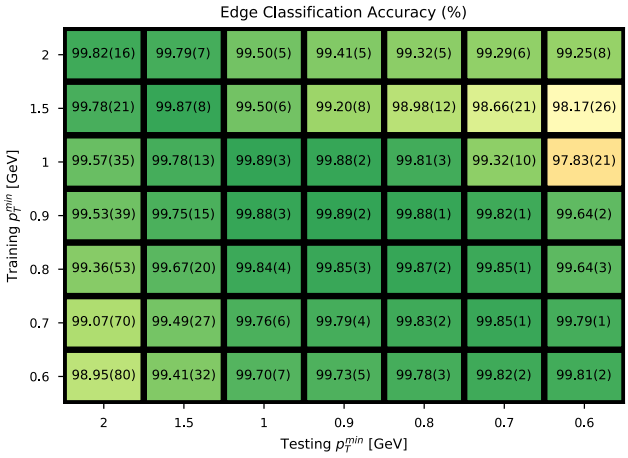

Fig. 8 Models trained on various $p_{\mathrm{T}}^{\mathrm{min}}$ graphs in the `train_1` sample were tested on 400 graphs from the `train_3` sample at various $p_{\mathrm{T}}^{\mathrm{min}}$ thresholds

candidates are subsequently matched to simulated particles (when possible). In a full tracking pipeline, these track candidates would then be fit to extract track parameters, such as p_{T} and η; in this work, we use truth information for matched particles to get the track parameters. Tracking efficiency metrics measure the relative success of the clustering and matching process using various definitions. We define three tracking efficiency measurements using progressively tighter requirements to allow comparison with current tracking algorithm efficiencies and other on-going HL-LHC tracking studies:

1. *LHC match efficiency* The number of reconstructed tracks containing over 75% of hits from the same particle, divided by the total number of particles.
2. *Double-majority efficiency* The number of reconstructed tracks containing over 50% of hits from the same particle and over 50% of that particle's hits, divided by the total number of particles.
3. *Perfect match efficiency* The number of reconstructed tracks containing only hits from the same particle and every hit generated by that particle, divided by the number of particles.

We note that the perfect match efficiency is not commonly used by experiments as 100% is not realistically achievable, but we present it to demonstrate the absolute performance of the GNN tracking pipeline.

Figure 9 shows each of these tracking efficiencies as a function of particle p_{T} and η for both the DBSCAN and union-find clustering approaches. Additionally, Table 2 shows the corresponding fake rates, or fractions of unmatched clusters relative to all clusters, across the full p_{T} and η range. The efficiencies and fake rates are calculated with $p_{\mathrm{T}}^{\mathrm{min}} = 0.9\,\mathrm{GeV}$ graphs. Tracking performance is relatively stable at low p_{T} but degrades for higher p_{T} particles; similar effects have been noted in other edge-weight-based hit clustering schemes [39]. The tracking efficiencies are lowest in the neighborhood of $\eta = 0$, indicating that performance is worst in the pixel barrel region. This is consistent with the observation that most edge classification errors occur in the barrel, where the density of detector modules is significantly higher [35]. Tracking efficiency loss around $|\eta| \approx 2.5$ corresponds to the transition region between barrel and endcap layers. DBSCAN demonstrates higher tracking efficiency than union-find across all p_{T} and η values and efficiency definitions. This performance gap is likely due to the additional spatial information used in DBSCAN's clustering routine.

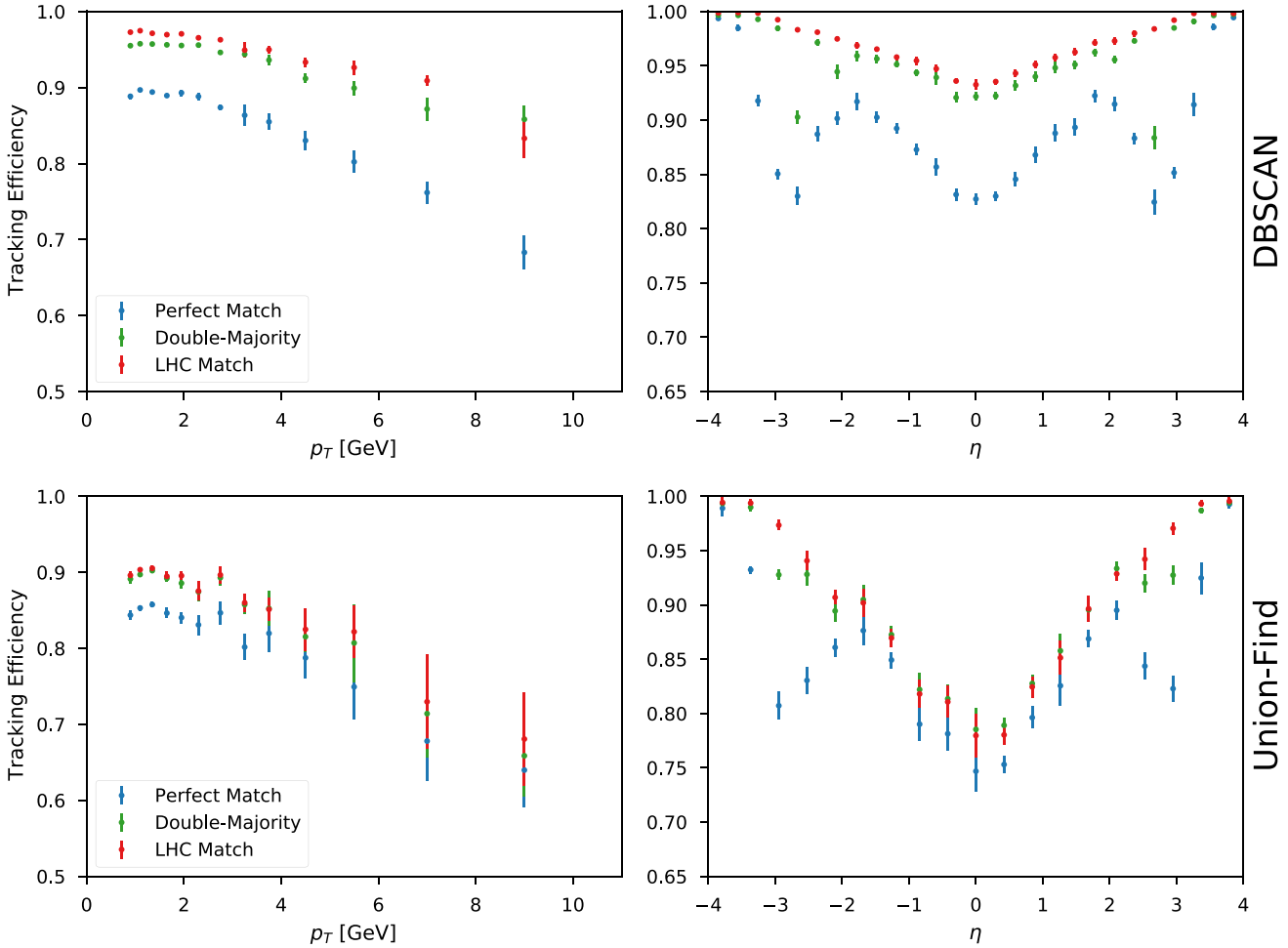

Fig. 9 The track-building performance of DBSCAN and union-find is measured as a function of particle p_T and η for three tracking efficiency definitions at $p_T^{min} = 0.9\,GeV$

Table 2 Overall fake rates of union-find and DBSCAN track-building for three tracking efficiency definitions for $p_T^{min} = 0.9\,GeV$

Efficiency definition	Union-find	DBSCAN
LHC match	0.0471 ± 0.008	0.0275 ± 0.005
Double majority	0.0934 ± 0.01	0.0891 ± 0.01
Perfect match	0.0910 ± 0.01	0.1242 ± 0.01

Moving forward, additional tracking performance may be recovered by leveraging the specific values of each edge weight to make dynamic hit clustering decisions. The fake rates are relatively low for both track-building methods, and as expected roughly increase for increasingly tight efficiency definitions. Interestingly, DBSCAN demonstrates a lower fake rate for LHC match efficiency while union-find demonstrates a lower fake rate for the perfect match efficiency; DBSCAN also has a larger drop in tracking efficency between the double match and perfect match definitions, indicating that while DBSCAN identifies more track candidates, union-find builds tracks more precisely.

Inference Timing

An important advantage of GNN-based approaches over traditional methods for HEP reconstruction is the ability to natively run on highly parallel computing architectures. The PyG library supports graphics processing units (GPUs) to parallelize the algorithm execution. Moreover, the model was prepared for inference by converting it to a TorchScript program [45]. For the IN studied in this work, the average CPU and GPU inference times per graph for a variety of minimum p_T cuts are shown in Table 3. For this test, the graphs are constructed using the geometric selections as described in Section 4.2. Moreover, we use bidirectional graphs, which means both directed edges (outward and inward from the primary vertex) are present in the edge list. As can be seen, inference can be significantly sped up

Table 3 CPU and GPU inference time estimates for each p_T threshold. The model was prepared for inference by converting it to a TorchScript program. The timing tests were performed with an Nvidia Titan Xp GPU with 12 GB RAM and a 12-core Intel Xeon CPU E5-2650 v4 @ 2.20 GHz. Inference is performed with a batch size of one graph. Graphs are constructed using geometric restrictions with bidirectional edges (both edge directions are present). The inference is repeated 100 times (after some warm-up) for 5 iterations and the best time per inference over the 5 iterations is found. The mean and standard deviation of the best inference time derived for 5 random graphs in the testing dataset are then reported. The mean and standard deviation of the number of nodes and edges are also reported for 100 graphs. We find a significant speedup with the GPU versus the CPU, which is greater at lower p_T^{min} because of the higher multiplicity and thus the greater gain from parallelization on the GPU

p_T^{min} [GeV]	CPU [ms]	GPU [ms]	$\overline{n}_{nodes}$	$\overline{n}_{edges}$
2	3.83 ± 0.89	0.95 ± 0.01	1090.6 ± 192.8	9080.7 ± 3027.1
1.5	7.96 ± 1.44	0.95 ± 0.03	2247.0 ± 363.9	30980.6 ± 9468.6
1	33.96 ± 11.24	3.61 ± 0.91	5309.3 ± 765.5	200910.1 ± 55825.9
0.9	52.44 ± 14.15	5.36 ± 1.37	6468.5 ± 912.2	312809.5 ± 85441.3
0.8	91.86 ± 24.42	9.60 ± 2.61	7970.5 ± 1100.0	556417.7 ± 151482.8
0.7	168.40 ± 41.34	17.70 ± 4.39	9982.7 ± 1341.5	1011884.4 ± 268706.0
0.6	273.20 ± 62.09	28.84 ± 6.65	12640.0 ± 1648.2	1585883.3 ± 409146.8
0.5	437.00 ± 97.99	44.66 ± 7.91	16178.6 ± 2019.1	2535979.6 ± 628297.1

with heterogeneous resources like GPUs. For instance, for a 0.5GeVminimum p_T cut, the inference time can be reduced by approximately a factor of 10 using the GPU with respect to the CPU. In general, the speedup is greater at lower p_T^{min} because of the higher multiplicity and thus the greater gain from parallelization on the GPU versus the CPU. Other heterogeneous computing resources specialized for inference may be even more beneficial. This speed-up may benefit the experiments' computing workflows by accessing these resources as an on-demand, scalable service [46–48].

Work has also been done to accelerate the inference of deep neural networks with heterogeneous resources beyond GPUs, like field-programmable gate arrays (FPGAs) [49–57]. This work extends to GNN architectures [29, 58]. Specifically, in Ref. [29], a compact version of the IN was implemented for $p_T > 2$ GeV segmented geometric graphs with up to 28 nodes and 37 edges, and shown to have a latency less than 1 μs, an initiation interval of 5 ns, reproduce the floating-point precision model with a fixed-point precision of 16 bits or less, and fit on a Xilinx Kintex UltraScale FPGA.

While this preliminary FPGA acceleration work is promising, there are several limitations of the current FPGA implementation of the IN:

1. This fully pipelined design cannot easily scale to beyond $\mathcal{O}(100)$ nodes and $\mathcal{O}(1000)$ edges. However, if the initiation interval requirements are loosened, it can scale up to $O(10,000)$ nodes and edges.
2. The neural network itself is small, and while it is effective for $p_T > 2$ GeV graphs, it may not be sufficient for lower-p_T graphs.
3. The FPGA design makes no assumptions about the possible graph connectivity (e.g. layer 1 nodes are only connected to layer 2 nodes), and instead allows all nodes to potentially participate in message passing. However by taking this additional structure into account, the hardware resources can be significantly reduced.
4. Quantization-aware training [55, 59–68] using QKeras [56, 69] or Brevitas [50, 70], parameter pruning [71–76], and general hardware-algorithm codesign can significantly reduce the necessary FPGA resources by reducing the required bit precision and removing irrelevant operations.
5. The design can be made more flexible, configurable, and reusable by integrating it fully with a user-friendly interface like `hls4ml` [77].

Summary and Outlook

In this work, we have shown that the physics-motivated interaction network (IN), a type of graph neural network (GNN), can successfully be applied to the task of charged particle tracking across a range of hitgraph sizes. Through a suite of graph construction, edge classification, and track building measurements, we have framed the IN's performance in the context of a GNN-based tracking pipeline following a truth-based hit filtering preselection in which hits associated with particles whose transverse momentum (p_T) is below a certain threshold (p_T^{min}) are removed. The graph construction measurements demonstrate that geometric cuts, hit clustering, and data-driven strategies are effective in constructing highly-efficient graphs from pixel barrel and endcap layers; in constrained computing environments, the parameters of each strategy allow a trade-off between graph efficiency and purity. In particular, for a fixed graph construction efficiency of $\mathcal{O}(99\%)$, we show that geometric

cuts alone produce reasonably pure graphs ($\sim 4\%$ purity at $p_{\mathrm{T}}^{\mathrm{min}} = 1\,\mathrm{GeV}$) but that the module-map method produces the most pure graphs for the entire range of $p_{\mathrm{T}}^{\mathrm{min}}$ ($\sim 10\%$ purity at $p_{\mathrm{T}}^{\mathrm{min}} = 1\,\mathrm{GeV}$). With efficiency held constant, purity is more-or-less a comparison of graph sizes, indicating that the module map method is most suited for graph construction in constrained computing environments. Though high graph construction efficiency is desirable in a global sense, graph purity is non-trivially related to downstream physics performance; in particular, many message passing GNN architectures may benefit from less-pure graphs due to higher edge connectivity.

The lightweight IN models trained in the edge classification step demonstrate extremely high edge classification efficiency for a range of $p_{\mathrm{T}}^{\mathrm{min}}$. Significantly, we find models trained in simpler scenarios (larger $p_{\mathrm{T}}^{\mathrm{min}}$) generalize to more complex scenarios (smaller $p_{\mathrm{T}}^{\mathrm{min}}$). Track building measurements performed on these edge-weighted graphs showed that DBSCAN's spatial clustering outperformed union-find clustering across a variety of efficiency definitions.

The IN architecture presented here is substantially smaller than previous GNN tracking architectures, which may enable its use in constrained computing environments. Accordingly, we have compared the IN's CPU and GPU inference times and discussed related work on accelerating INs with FPGAs. As described in Section 4.5, there are several limitations to the current FPGA implementation of the IN and addressing these concerns is the subject of ongoing work.

Another important aspect of GNN-based tracking is reducing the time it takes to construct graphs. Ongoing efforts are dedicated to studying how best to accelerate graph construction using heterogeneous resources. Alternative GNN approaches that do not require an input graph structure, such as dynamic graph convolutional neural networks [24], distance-weighted GNNs [78], attention-based transformers [79], reformers [80], and performers [81], may be fruitful avenues of investigation as well.

In summary, geometric deep learning methods can be naturally applied to many physics reconstruction tasks, and our work and related studies establish GNNs as an extremely promising candidate for tracking at the high luminosity LHC.

Acknowledgements We gratefully acknowledge the input and discussion from the Exa.TrkX collaboration.

References

1. Piacquadio G, Prokofiev K, Wildauer A (2008) Primary vertex reconstruction in the ATLAS experiment at LHC. J Phys Conf Ser 119:032033. https://doi.org/10.1088/1742-6596/119/3/032033
2. CMS Collaboration (2014) Description and performance of track and primary-vertex reconstruction with the CMS tracker. JINST 9(10):P10009. https://doi.org/10.1088/1748-0221/9/10/P10009. arXiv:1405.6569
3. ATLAS Collaboration (2017) Jet reconstruction and performance using particle flow with the ATLAS detector. Eur Phys J C 77:466. https://doi.org/10.1140/epjc/s10052-017-5031-2. arXiv:1703.10485
4. Collaboration CMS (2017) Particle-flow reconstruction and global event description with the CMS detector. JINST 12:P10003. https://doi.org/10.1088/1748-0221/12/10/P10003. arXiv:1706.04965
5. Larkoski AJ, Moult I, Nachman B (2020) Jet substructure at the large hadron collider: a review of recent advances in theory and machine learning. Phys Rep 841:1. https://doi.org/10.1016/j.physrep.2019.11.001. arXiv:1709.04464
6. CMS Collaboration (2018) Identification of heavy-flavour jets with the CMS detector in pp collisions at 13 TeV. JINST 13(05):P05011. https://doi.org/10.1088/1748-0221/13/05/P05011. arXiv:1712.07158
7. ATLAS Collaboration (2018) Measurements of b-jet tagging efficiency with the ATLAS detector using $t\bar{t}$ events at $\sqrt{s} = 13$ TeV. JHEP 08:089. https://doi.org/10.1007/JHEP08(2018)089. arXiv:1805.01845
8. ATLAS Collaboration (2017) Performance of the ATLAS track reconstruction algorithms in dense environments in LHC run 2. Eur Phys J C 77(10):673. https://doi.org/10.1140/epjc/s10052-017-5225-7. arXiv:1704.07983
9. Billoir P (1989) Progressive track recognition with a Kalman-like fitting procedure. Comput Phys Commun 57:390. https://doi.org/10.1016/0010-4655(89)90249-X
10. Billoir P, Qian S (1990) Simultaneous pattern recognition and track fitting by the Kalman filtering method. Nucl Instrum Methods A 294:219. https://doi.org/10.1016/0168-9002(90)91835-Y
11. Mankel R (1997) A concurrent track evolution algorithm for pattern recognition in the hera-b main tracking system. Nucl Instrum Methods A 395:169. https://doi.org/10.1016/S0168-9002(97)00705-5
12. Frühwirth R (1987) Application of Kalman filtering to track and vertex fitting. Nucl Instrum Methods A 262:444. https://doi.org/10.1016/0168-9002(87)90887-4
13. CMS Collaboration (2018) Expected performance of the physics objects with the upgraded CMS detector at the HL-LHC, CMS Note CMS-NOTE-2018-006. CERN-CMS-NOTE-2018-006
14. Apollinari G et al (eds) (2017) High-luminosity large hadron collider (HL-LHC): technical design report V. 0.1, vol 4/2017. CERN. https://doi.org/10.23731/CYRM-2017-004
15. Bronstein MM et al (2017) Geometric deep learning: going beyond Euclidean data. IEEE Signal Process Mag 34:18. https://doi.org/10.1109/MSP.2017.2693418. arXiv:1611.08097
16. Zhang Z, Cui P, Zhu W (2020) Deep learning on graphs. A survey. IEEE Trans Knowl Data Eng (1):1. https://doi.org/10.1109/TKDE.2020.2981333. arXiv:1812.04202

17. Zhou J et al (2020) Graph neural networks: a review of methods and applications.AI Open 1:57-81. https://doi.org/10.1016/j.aiopen.2021.01.001. arXiv:1812. 08434
18. Wu Z et al (2020) A comprehensive survey on graph neural networks. IEEE Trans Neural Netw Learn Syst. https://doi.org/10.1109/TNNLS.2020.2978386. arXiv:1901.00596
19. Scarselli F et al (2009) The graph neural network model. IEEE Trans Neural Netw 20(1):61. https://doi.org/10.1109/TNN.2008.2005605
20. Qi CR, Su H, Mo K, Guibas LJ (2017) PointNet: deep learning on point sets for 3D classification and segmentation. In: 2017 IEEE conference on computer vision and pattern recognition (CVPR). https://arxiv.org/abs/1612.00593. https://doi.org/10.1109/CVPR.2017.16
21. Gilmer J et al (2017) Neural message passing for quantum chemistry. In: Proceedings of the 34th international conference on machine learning, volume 70 of Proceedings of machine learning research, p 1263. https://arxiv.org/abs/1704.01212
22. Battaglia PW et al (2016) Interaction networks for learning about objects, relations and physics. In: Lee D et al (eds) Advances in neural information processing systems, vol 29, p 4502. Curran Associates, Inc. https://arxiv.org/abs/1612.00222
23. Battaglia PW et al (2018) Relational inductive biases, deep learning. and graph networks. https://arxiv.org/abs/1806.01261
24. Wang Y et al (2019) Dynamic graph CNN for learning on point clouds. ACM Trans Graph. https://doi.org/10.1145/3326362. https://arxiv.org/abs/1801.07829
25. Duarte J, Vlimant J-R (2020) Graph neural networks for particle tracking and reconstruction. In: Calafiura P, Rousseau D, Terao K (eds) Artificial intelligence for high energy physics. World Scientific Publishing, p 12. https://arxiv.org/abs/2012.01249. https://doi.org/10.1142/12200**(Submitted to Int. J. Mod. Phys. A)**
26. Shlomi J, Battaglia P, Vlimant J-R (2020) Graph neural networks in particle physics. Mach Learn Sci Technol 2:021001. https://doi.org/10.1088/2632-2153/abbf9a. arXiv:2007.13681
27. Farrell S et al (2018) Novel deep learning methods for track reconstruction. In: 4th international workshop connecting the dots 2018. https://arxiv.org/abs/1810.06111
28. Ju X et al (2019) Graph neural networks for particle reconstruction in high energy physics detectors. In: Machine learning and the physical sciences workshop at the 33rd annual conference on neural information processing systems. https://arxiv.org/abs/2003.11603
29. Heintz A et al (2020) Accelerated charged particle tracking with graph neural networks on FPGAs. In: 3rd machine learning and the physical sciences workshop at the 34th annual conference on neural information processing systems, vol 12. https://arxiv.org/abs/2012.01563
30. Ju X et al (2021) Performance of a geometric deep learning pipeline for HL-LHC particle tracking, vol 3. https://arxiv.org/abs/2103.06995**(Submitted to Eur. Phys. J. C)**
31. Moreno EA et al (2020) Interaction networks for the identification of boosted $H \to b\bar{b}$ decays. Phys Rev D 102:012010. https://doi.org/10.1103/PhysRevD.102.012010. https://arxiv.org/abs/1909.12285
32. Moreno EA et al (2020) JEDI-net: a jet identification algorithm based on interaction networks. Eur Phys J C 80:58. https://doi.org/10.1140/epjc/s10052-020-7608-4. https://arxiv.org/abs/1908.05318
33. Besta M et al (2019) Graph processing on FPGAs: taxonomy, survey, challenges. arXiv:1903.06697
34. Strandlie A, Frühwirth R (2010) Track and vertex reconstruction: from classical to adaptive methods. Rev Mod Phys 82:1419. https://doi.org/10.1103/RevModPhys.82.1419
35. Amrouche S et al (2020) The tracking machine learning challenge: accuracy phase. In: The NeurIPS '18 competition, p 231. https://arxiv.org/abs/1904.06778. https://doi.org/10.1007/978-3-030-29135-8_9
36. Fey M, Lenssen JE (2019) Fast graph representation learning with PyTorch Geometric. In: Representation learning on graphs and manifolds workshop at the 7th international conference on learning representations. arXiv:1903.02428
37. Paszke A et al (2019) PyTorch: an imperative style, high-performance deep learning library. In: Wallach H et al (eds) Advances in neural information processing systems, vol 32. Curran Associates, Inc. arXiv:1912. 01703
38. Ester M, Kriegel H-P, Sander J, Xu X (1996) A density-based algorithm for discovering clusters in large spatial databases with noise. In: Proceedings of the second international conference on knowledge discovery and data mining, p 226. AAAI Press
39. Biscarat C et al (2021) Towards a realistic track reconstruction algorithm based on graph neural networks for the HL-LHC. In: 25th international conference on computing in high-energy and nuclear physics, vol 3. arXiv:2103. 00916
40. DeZoort G, Duarte J (2021) GageDeZoort/interaction_network_paper: interaction networks for GNN-based particle tracking. https://doi.org/10.5281/zenodo.5558032. https://github.com/GageDeZoort/interaction_network_paper. Accessed 08 Oct 2021
41. Nair V, Hinton GE (2010) Rectified linear units improve restricted Boltzmann machines. In: 27th international conference on international conference on machine learning, ICML'10, p 807. Omnipress, Madison
42. Glorot X, Bordes A, Bengio Y (2011) Deep sparse rectifier neural networks. In: Gordon G, Dunson D, Dudík M (eds) 14th international conference on artificial intelligence and statistics, vol 15, p 315. JMLR, Fort Lauderdale, FL, USA, 4
43. Kingma DP, Ba J (2015) Adam: a method for stochastic optimization. In: Bengio Y, LeCun Y (eds) 3rd international conference on learning representations. https://arxiv.org/abs/1412.6980
44. Patwary MMA, Blair J, Manne F (2010) Experiments on union-find algorithms for the disjoint-set data structure. In: Festa P (ed) Experimental algorithms. Springer, Berlin, pp 411–423
45. The PyTorch Team, TorchScript (2021). https://pytorch.org/docs/stable/jit.html. Accessed 08 Oct 2021
46. Krupa J et al (2021) Gpu coprocessors as a service for deep learning inference in high energy physics. Mach Learn Sci Technol 2(3):035005. https://doi.org/10.1088/2632-2153/abec21. https://arxiv.org/abs/2007.10359
47. Rankin DS et al (2020) FPGAs-as-a-service toolkit (FaaST). In: 2020 IEEE/ACM international workshop on heterogeneous high-performance reconfigurable computing (H2RC). https://arxiv.org/abs/2010.08556. https://doi.org/10.1109/H2RC51942.2020.00010
48. Wang M et al (2021) GPU-accelerated machine learning inference as a service for computing in neutrino experiments. Front Big Data 3:48. https://doi.org/10.3389/fdata.2020.604083. https://arxiv.org/abs/2009.04509
49. Umuroglu Y et al (2017) FINN: a framework for fast, scalable binarized neural network inference. In: Proceedings of the 2017 ACM/SIGDA international symposium on field-programmable gate arrays, p 65. ACM, New York, NY, USA. https://arxiv.org/abs/1612.07119. https://doi.org/10.1145/3020078.3021744
50. Blott M et al (2018) FINN-R: An end-to-end deep-learning framework for fast exploration of quantized neural networks. ACM Trans Reconfig Technol Syst 11(12). https://doi.org/10.1145/3242897. https://arxiv.org/abs/1809.04570
51. Shawahna A, Sait SM, El-Maleh A (2019) FPGA-based accelerators of deep learning networks for learning and classification: a review. IEEE Access 7:7823. https://doi.org/10.1109/ACCESS.2018.2890150. https://arxiv.org/abs/1901.00121

52. Wang T, Wang C, Zhou X, Chen H (2019) An overview of FPGA based deep learning accelerators: challenges and opportunities. In: 2019 IEEE 21st international conference on high performance computing and communications; IEEE 17th international conference on smart city; IEEE 5th international conference on data science and systems (HPCC/SmartCity/DSS), p 1674. https://arxiv.org/abs/1901.04988. https://doi.org/10.1109/HPCC/SmartCity/DSS.2019.00229
53. Duarte J et al (2018) Fast inference of deep neural networks in FPGAs for particle physics. JINST 13:P07027. https://doi.org/10.1088/1748-0221/13/07/P07027. https://arxiv.org/abs/1804.06913
54. Summers S et al (2020) Fast inference of boosted decision trees in FPGAs for particle physics. JINST 15:P05026. https://doi.org/10.1088/1748-0221/15/05/p05026. https://arxiv.org/abs/2002.02534
55. Ngadiuba J et al (2020) Compressing deep neural networks on FPGAs to binary and ternary precision with `hls4ml`. Mach Learn Sci Technol 2(1):015001. https://doi.org/10.1088/2632-2153/aba042. https://arxiv.org/abs/2003.06308
56. Coelho CN et al (2021) Automatic heterogeneous quantization of deep neural networks for low-latency inference on the edge for particle detectors. Nat Mach Intell. https://doi.org/10.1038/s42256-021-00356-5. https://arxiv.org/abs/2006.10159
57. Åarrestad T et al (2021) Fast convolutional neural networks on FPGAs with `hls4ml`. Mach Learn Sci Technol 2(4):045015. https://doi.org/10.1088/2632-2153/ac0ea1. https://arxiv.org/abs/2101.05108
58. Iiyama Y et al (2021) Distance-weighted graph neural networks on FPGAs for real-time particle reconstruction in high energy physics. Front Big Data 3:44. https://doi.org/10.3389/fdata.2020.598927. https://arxiv.org/abs/2008.03601
59. Moons B, Goetschalckx K, Berckelaer NV, Verhelst M (2017) Minimum energy quantized neural networks. In: 2017 51st asilomar conference on signals, systems, and computers, p 1921. https://arxiv.org/abs/1711.00215. https://doi.org/10.1109/ACSSC.2017.8335699
60. Courbariaux M, Bengio Y, David J-P (2015) BinaryConnect: training deep neural networks with binary weights during propagations. In: Cortes C et al (eds) Advances in Neural Information Processing Systems, vol 28, p 3123. Curran Associates, Inc. https://arxiv.org/abs/1511.00363
61. Zhang D, Yang J, Ye D, Hua G (2018) LQ-nets: learned quantization for highly accurate and compact deep neural networks. In: Proceedings of the European conference on computer vision (ECCV), p 365. https://arxiv.org/abs/1807.10029
62. Li F, Liu B (2016) Ternary weight networks. https://arxiv.org/abs/1605.04711
63. Zhou S et al (2016) DoReFa-Net: training low bitwidth convolutional neural networks with low bitwidth gradients. https://arxiv.org/abs/1606.06160
64. Hubara I et al (2018) Quantized neural networks: training neural networks with low precision weights and activations. J Mach Learn Res 18(187):1. https://arxiv.org/abs/1609.07061
65. Rastegari M, Ordonez V, Redmon J, Farhadi A (2016) XNOR-Net: ImageNet classification using binary convolutional neural networks. In: 14th European conference on computer vision (ECCV). Springer International Publishing, Cham, p 525. https://doi.org/10.1007/978-3-319-46493-0_32. https://arxiv.org/abs/1603.05279
66. Micikevicius P et al (2018) Mixed precision training. In: 6th international conference on learning representations. https://arxiv.org/abs/1710.03740
67. Zhuang B et al (2018) Towards effective low-bitwidth convolutional neural networks. In: 2018 IEEE/CVF conference on computer vision and pattern recognition, p 7920. https://arxiv.org/abs/1711.00205. https://doi.org/10.1109/CVPR.2018.00826
68. Wang N et al (2018) Training deep neural networks with 8-bit floating point numbers. In: Bengio S et al (eds) Advances in neural information processing systems, vol 31, p 7675. Curran Associates, Inc. https://arxiv.org/abs/1812.08011
69. Coelho C (2019) QKeras. https://github.com/google/qkeras. Accessed 08 Oct 2021
70. Pappalardo A (2020) Xilinx/brevitas. https://doi.org/10.5281/zenodo.3333552
71. LeCun Y, Denker JS, Solla SA (1990) In: Touretzky DS (ed) Optimal brain damage. In: Advances in neural information processing systems, vol 2. Morgan-Kaufmann, p 598
72. Han S, Mao H, Dally WJ (2016) Deep compression: compressing deep neural networks with pruning, trained quantization and Huffman coding. In: 4th international conference on learning representations. https://arxiv.org/abs/1510.00149
73. Frankle J, Carbin M (2019) The lottery ticket hypothesis: training pruned neural networks. In: 7th international conference on learning representations. https://arxiv.org/abs/1803.03635
74. Zhou H, Lan J, Liu R, Yosinski J (2019) Deconstructing lottery tickets: zeros, signs, and the supermask. In: Wallach H et al (eds) Advances in neural information processing systems, vol 32, p 3597. Curran Associates, Inc. https://arxiv.org/abs/1905.01067
75. Blalock D, Ortiz JJG, Frankle J, Guttag J (2020) What is the state of neural network pruning? In: 4th conference on machine learning and systems. https://arxiv.org/abs/2003.03033
76. Hawks B et al (2021) Ps and Qs: quantization-aware pruning for efficient low latency neural network inference. Front AI 4:94. https://doi.org/10.3389/frai.2021.676564. https://arxiv.org/abs/2102.11289
77. Loncar V et al (2021) fastmachinelearning/hls4ml. https://doi.org/10.5281/zenodo.4447439
78. Qasim SR, Kieseler J, Iiyama Y, Pierini M (2019) Learning representations of irregular particle-detector geometry with distance-weighted graph networks. Eur Phys J C 79:608. https://doi.org/10.1140/epjc/s10052-019-7113-9. https://arxiv.org/abs/1902.07987
79. Vaswani A et al (2017) Attention is all you need. In: Guyon I et al (eds) Advances in neural information processing systems, vol 30, p 5998. Curran Associates, Inc. https://arxiv.org/abs/1706.03762
80. Kitaev N, Kaiser Ł, Levskaya A (2020) Reformer: the efficient transformer. In: 8th international conference on learning representations. https://arxiv.org/abs/2001.04451
81. Choromanski K et al (2021) Rethinking attention with performers. In: 9th international conference on learning representations. arXiv:2009. 14794