Title

Performance of a geometric deep learning pipeline for HL-LHC particle tracking

Journal

European Physical Journal C, 81(10)

ISSN

1434-6044

Authors

Ju, Xiangyang

Murnane, Daniel

Calafiura, Paolo

et al.

Publication Date

2021-10-01

DOI

10.1140/epjc/s10052-021-09675-8

Peer reviewed

Physics and Computing Performance of the Exa.TrkX TrackML Pipeline

Xiangyang Ju[1], *Daniel* Murnane[1], *Paolo* Calafiura[1,*], *Nicholas* Choma[1], *Sean* Conlon[1], *Steve* Farrell[1], *Yaoyuan* Xu[1], *Maria* Spiropulu[2], *Jean-Roch* Vlimant[2], *Adam* Aurisano[3], *Jeremy* Hewes[3], *Giuseppe* Cerati[4], *Lindsey* Gray[4], *Thomas* Klijnsma[4], *Jim* Kowalkowski[4], *Markus* Atkinson[5], *Mark* Neubauer[5], *Gage* DeZoort[6], *Savannah* Thais[6], *Aditi* Chauhan[7], *Alex* Schuy[7], *Shih-Chieh* Hsu[7], *Alex* Ballow[8], and *Alina* Lazar[8]

[1]Lawrence Berkeley National Laboratory, Berkeley, CA, USA
[2]California Institute of Technology, Pasadena, CA USA
[3]University of Cincinnati, Cincinnati, OH USA
[4]Fermi National Accelerator Laboratory, Batavia, IL USA
[5]University of Illinois at Urbana-Champaign, Urbana, IL, USA
[6]Princeton University, Princeton, NJ, USA
[7]University of Washington, Seattle, WA, USA
[8]Youngstown State University, Youngstown, OH, USA

Abstract. The Exa.TrkX project has applied geometric learning concepts such as metric learning and graph neural networks to HEP particle tracking. The Exa.TrkX tracking pipeline clusters detector measurements to form track candidates and filters them. The pipeline, originally developed using the TrackML dataset (a simulation of an LHC-like tracking detector), has been demonstrated on various detectors, including the DUNE LArTPC and the CMS High-Granularity Calorimeter. This paper documents new developments needed to study the physics and computing performance of the Exa.TrkX pipeline on the full TrackML dataset, a first step towards validating the pipeline using ATLAS and CMS data. The pipeline achieves tracking efficiency and purity similar to production tracking algorithms. Crucially for future HEP applications, the pipeline benefits significantly from GPU acceleration, and its computational requirements scale close to linearly with the number of particles in the event.

1 Introduction

Charged particle tracking plays an essential role in High-Energy Physics (HEP), including particle identification and kinematics, vertex finding, lepton reconstruction, and flavor jet tagging. At the core of particle tracking there is a pattern recognition algorithm that must associate a list of 2D or 3D position measurements from a tracking detector (known as *hits* or *spacepoints* in literature) to a list of particle track candidates (*tracks*).

The number of particle track candidates varies significantly from one experiment setup to another. For example, in a High-Luminosity LHC (HL-LHC) [1] collision *event*, due to the *pile-up* of multiple proton-proton collision per bunch crossing, there are typically 5,000 tracks and 100,000 spacepoints, about 50 % of which are associated to particles of interest.

*e-mail: pcalafiura@lbl.gov

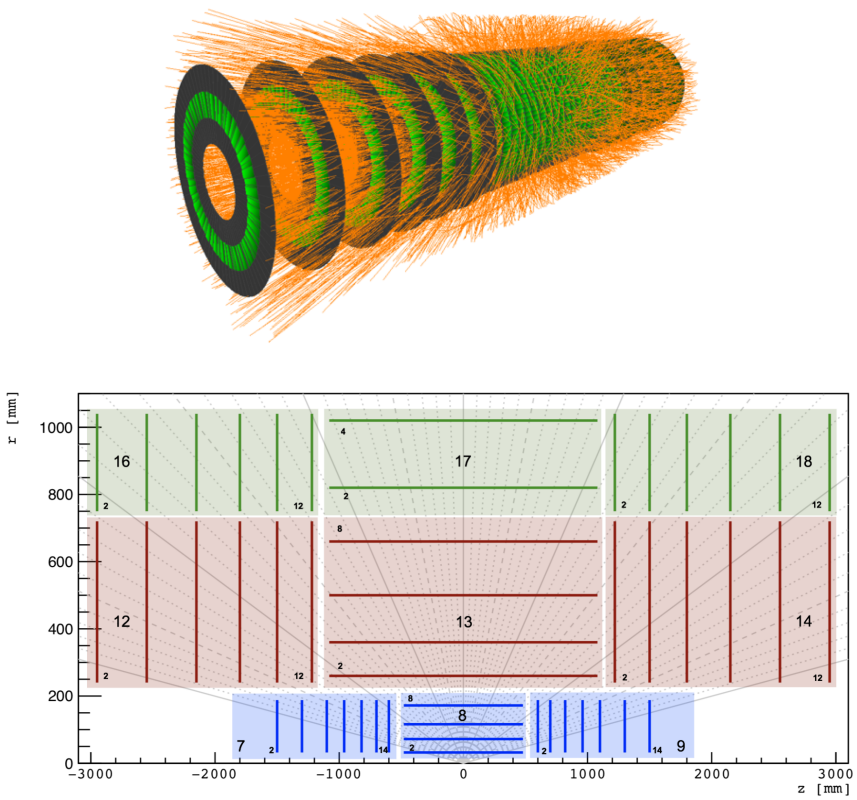

Figure 1. A simulated HL-LHC collision event (top) as seen by the TrackML tracking detector [2]. The detector schematic (bottom) shows the top half of the detector projected on the r-z plane. The z-axis is along the beam direction.

A typical HEP offline tracking algorithm [3–5] has four stages: spacepoint formation, track seeding, track following, and track fitting. The spacepoint formation stage combines the detector readout cell raw data in clusters from which the spacepoint 3D coordinates, and their uncertainties, are determined. Track seeding combines spacepoints in *doublet* or *triplet seeds*. Each seed provides an initial track direction, origin, and possibly a curvature, with associated uncertainties. The track following stage adds more spacepoints to the seed by looking for matching spacepoints along the extrapolated trajectory. Finally a track fitting stage, which may be combined with the track following, fits a trajectory through the track spacepoints to assess the track quality and measure the particle's physical and kinematic properties (charge, momentum, origin, etc). To avoid biasing physics results, each stage of the algorithm must have high *efficiency*, meaning it must identify e.g. >90% of the charged particles within a fiducial region (e.g. $p_{\mathrm{T}} > 1$ GeV, $\eta < 4$) as track candidates. Track seeding and track filtering must also have high *purity*, meaning that e.g. >60% of the track seeds and track candidates must correspond to charged particles. High purity allows to keep the number of track candidates, and the associated computational costs, under control.

Online tracking algorithms may use different pattern recognition algorithms (including Hough transforms [6, 7], cellular automata [8, 9]) to create and filter track seeds and candi-

dates, but share the same high efficiency requirements. Online application also have stringent computing requirements (e.g. latency $O(10)$ μs for LHC triggers).

The computational cost of current tracking algorithms grows worse than linearly with beam intensity and detector occupancy, as demonstrated in Figure 2. This is due to the combinatorial increase in the number of track seeds and spacepoints to match as the density of detector measurements increases. Given the order-of-magnitude increase expected for beam intensity at HL-LHC, charged particle pattern recognition algorithms might well limit the discovery potential of HL-LHC experiments.

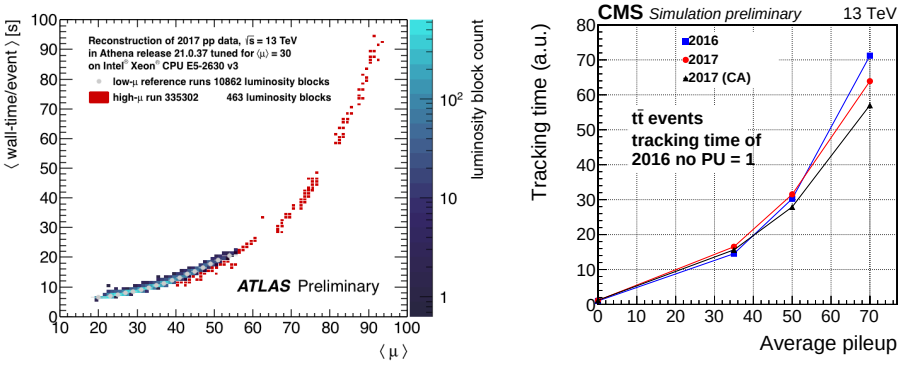

Figure 2. Reconstruction wall time per event as a function of the average number of interactions per bunch crossing $\langle \mu \rangle$. Left: ATLAS Run 2 Inner Detector reconstruction with default configurations [10]. Right: CMS time spent in tracking sequence for 2016 tracking, 2017 tracking with conventional seeding, and 2017 tracking with Cellular Automaton (CA) seeding [11].

Over the last two decades, tracking computational challenges arising from the increased number of combinations have been addressed by tightening fiducial regions for charged particles, developing highly optimized tracking algorithms [4, 5], and even optimizing the geometry of tracking detectors. These optimizations brought order-of-magnitude gains in tracking computational performance with limited impact on physics. While these efforts continue [12], it is unlikely that another order of magnitude can be gained through incremental optimization without impacting physics performance. Furthermore, given the computational complexity and iterative nature of current track following and filtering algorithms, it is challenging to run them efficiently on data parallel architectures like GPUs.

The TrackML challenge [2] jump-started the exploration of deep learning pattern recognition methods applied to HEP tracking. The HEP.TrkX pilot project [13] proposed the use of graph networks to filter track doublet and triplet seeds [14]. Building on that work, the Exa.TrkX project [15] has demonstrated the applicability of the *Geometric Deep Learning* (GDL) [16] – specifically metric learning and Graph Neural Networks (GNN) – to particle tracking [17]. GDL is concerned with learning representations of data that have complex geometrical relationships and no natural ordering, like detector spacepoints. GDL models are computationally regular, naturally parallel and therefore well-suited to run on accelerators.

This work describes new developments that enabled the first study of the computing and physics performance of the Exa.TrkX pipeline on the entire TrackML detector at HL-LHC design luminosity, a step towards the validation of the pipeline on ATLAS and CMS data.

2 Related work

Early on, the Hep.TrkX pilot project attempted to assign and regress track parameters to single spacepoints using image processing models. Subsequent attempts at estimating track parameters using image processing and recurrent networks showed promising results [18] in a simplified environment. A similar realization of the method is reported in [19] where a model processing image from successive pixel detector layers is used to produce tracklets, seeds to classical pattern recognition. The method yields superior seeding efficiency for tracks within jets in dense environments. The concept of using LSTM [20] to supplement the Kalman Filter method for track following developed by HEP.TrkX [14, 18, 21] was later found in one of the promising solutions of the accuracy phase [22] of the TrackML challenge. The task of particle tracking was addressed with a hit-to-track assignment method using gated recurrent unit [23] (GRU), producing promising result in sparse environments [21]. This approach was constrained computationally due to the use of recurrent models.

Ref. [24] applies the track finding approach developed in [25] to the whole detector by exploiting a new data-driven graph construction method and large model support in Tensorflow [26]. Ref. [27] applies a similar GNN model to the task of particle-flow reconstruction. The model has a classification objective, followed by a partial regression of generator-level particle candidate kinematics. The method performs at least as well as a classical particle-flow algorithm in HL-LHC-like collision conditions. As part of the Exa.TrkX project, graph networks are used for LArTPC track reconstruction [28]. Ref. [29] explores the opportunity to implement Exa.TrkX-inspired graph networks on FPGAs. Starting from the input stage of the Exa.TrkX pipeline, Ref. [30] studies the impact of pixel cluster shape information on track seeding performance. In Ref. [31], metric learning is used to improve the purity in spacepoints buckets formed using similarity hashing. With the advent of quantum computer of increasing size came the development of quantum machine learning techniques, also applied in particle physics [32]. In particular, inspired by the use of GNN for charged particle tracking of the Exa.TrkX team, quantum graph networks have been tested on the same problem [33–35].

3 Methodology

3.1 Input Data

This study is based on the TrackML dataset that uses a Montecarlo simulation of top quark pair production from proton-proton collisions at the HL-LHC. To simulate the effect of event pileup and produce realistic detector occupancy, a Poisson random number (with $\mu = 200$) of QCD "minimum bias" events are overlaid on top of the $t\bar{t}$ collisions.

The TrackML detector is a set of concentric cylindrical layers of pixelated sensors (the *barrell*) complemented by a set of circular disks (the *endcaps*) to ensure nearly 4π coverage in solid angle, as pictured in Figure 1. Figure 3 shows the spatial distribution of the spacepoints of a typical event. One notable feature of this dataset is the inclusion of “noise” spacepoints, added as a proxy for various low-momentum particle interactions and detector effects which would otherwise require more expensive and detailed simulations.

3.2 The Exa.TrkX TrackML Pipeline

This paper updates the methodology in Ref. [17, 25] to a fully-learned end-to-end pipeline, where both graph construction and graph classification are trained. This section describes the

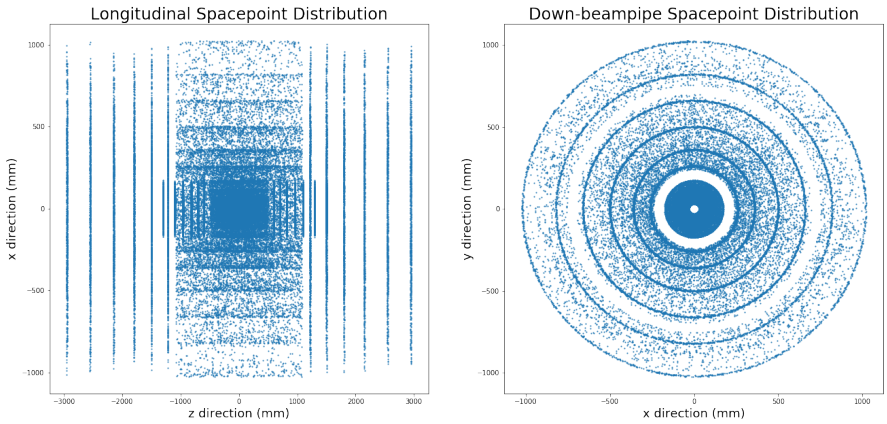

Figure 3. A typical event distribution of spacepoints projected on the x-z plane, parallel to the beam direction (left), and the x-y plane, orthogonal to the beam direction (right).

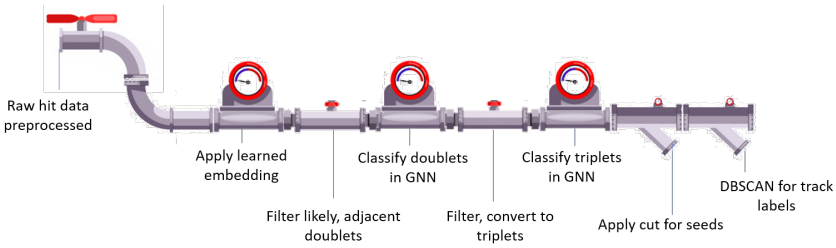

Figure 4. Stages of the TrackML track formation pipeline.

pipeline (represented schematically in Figure 4) used to obtain the results in § 4. Details of the model design and parameter choices are discussed in § 5.

First, the dataset is processed into a format suitable for model training. This includes calculating directional information and summary statistics from the charge deposited in each spacepoint. These values are appended to the cylindrical co-ordinates of each spacepoint to form an input feature vector to the pipeline. To apply a graph neural network to this set of data, it is necessary to arrange them into a graph. One can apply various geometric heuristics to define which spacepoints are likely to be connected by an edge (i.e. belong to the same track), but a useful technique is to train a model on the geometry of connected tracks. Thus, our second stage is to train an Embedding Network – a multi-layer perceptron (MLP) which embeds each spacepoint into an N-dimensional latent space. The graph is constructed by connecting neighboring spacepoints within a radius $r_{\text{embedding}}$, in the latent space. We train this embedding with a pairwise hinge loss, to encourage spacepoints that belong to the same track to be close in the embedded space, according to the Euclidean metric. This allows for a highly efficient edge construction, since we do not rely on any heuristics of the detector geometry that may lead to missed edges.

The edge selection at this stage is close to 100% efficient but $O(1)$% pure, with a graph size of $O(10^5)$ nodes and $O(10^7)$ edges (the purity-efficiency trade-off can be tuned with the choice of $r_{\text{embedding}}$). Before running training or inference on the memory-intensive GNN, we filter these edges down with another MLP. The input to this third stage is the concatenated features on either side of each edge. That is, the Filter Network is a binary classifier applied

to the set of edges. Constraining efficiency to remain high (above 96%) leads to much sparser graphs, of $O(10^6)$ edges.

The fourth stage of the pipeline is the training and inference of the graph neural network. The results presented in this work are predominantly obtained from the Interaction Network architecture, first proposed in Ref. [36]. This varietal of GNN includes hidden features on both nodes and edges, which are propagated around the graph (called "message passing") with consecutive concatenations along edges and aggregations of messages at receiving nodes. In the final layer of the network, a binary classification is obtained for each edge as true or fake, and trained on a cross-entropy loss.

The final stage of the TrackML pipeline involves task-specific post-processing. If our goal is track formation, we can place a threshold on edge scores produced by the GNN and partition the graph into connected components. If our goal is track seeding, we can directly sample the classified edges for high likelihood combinations of connected triplets, or convert the entire graph to a *triplet graph* and train this on a second GNN to classify the triplets. A triplet graph is formed by taking all edges in the original (*doublet*) graph and assigning them as nodes in the new triplet graph. The nodes in this triplet graph are connected if they share a hit in the doublet graph. Applying a GNN to this structure produces highly pure sets of seeds as shown in Ref. [17].

Many of these techniques are common to other applications being explored in the Exa.TrkX collaboration. In particular, the pattern of nearest-neighbor graph-building and GNN edge classification has shown its potential for neutrino experiments [28] and CMS High Granularity Calorimeter [25]. Indeed, these applications build on the TrackML pipeline and extend it, for example by adding the particle type as an edge feature.

4 Results

4.1 Tracking Performance of the TrackML pipeline

4.1.1 Tracking Efficiency and Purity

The performance of a tracking pipeline is mainly characterized by tracking efficiency and purity. For efficiency calculations, only charged particles that satisfy $|\eta| < 4.0$ and $p_{\mathrm{T}} > 100$ MeV are considered. These *selected* particles, $N_{particles}$(selected), are hereafter referred to as *particles*.

The overall tracking efficiency, known as *physics efficiency*, is defined as the fraction of particles that are *matched* to at least one reconstructed track. A particle is considered to be matched to a reconstructed track when 1) the majority of spacepoints in the reconstructed track belong to the same true track, and 2) the majority of spacepoints in the matched true particle track are found in the reconstructed track[1].

To measure the efficiency of the tracking pipeline itself, we also define the *technical efficiency* as the fraction of *reconstructable* particles that are matched to at least one reconstructed track. Reconstructable particles have a trajectory that leaves at least five spacepoints in the detector. Tracking purity is defined as the fraction of reconstructed tracks that match a selected particle[2].

[1] This nomenclature and the associated definitions broadly follow [2, 37].

[2] HEP tracking literature often quotes fake rate = 1 − purity

$$\text{Physics Efficiency} = \frac{N_{particles}(\text{selected, matched})}{N_{particles}(\text{selected})} \tag{1}$$

$$\text{Technical Efficiency} = \frac{N_{particles}(\text{selected, reconstructable, matched})}{N_{particles}(\text{selected, reconstructable})} \tag{2}$$

$$\text{Purity} = \frac{N_{tracks}(\text{selected,matched})}{N_{tracks}(\text{selected})} \tag{3}$$

Averaged over 50 testing events from the TrackML dataset, the physics efficiency for particles with $p_T > 500$ MeV is 88.7 ± 0.3% and the technical efficiency is 97.6 ± 0.3%. Without any fiducial p_T cut, the physics efficiency becomes 67.2 ± 0.1% and the technical efficiency 91.3 ± 0.2%. The tracking purity is 58.3 ± 0.6%. Using the TrackML challenge scoring system, we obtained a score of 0.901 ± 0.004. The errors quoted are statistical only.

Figure 5 shows the p_T distribution of particles as well as the tracking efficiency as a function of particle p_T. The physics efficiency for particles with p_T of [100, 300] MeV is 43%, therefore, is not displayed in the plot. The physics efficiency for particles with $p_T > 700$ MeV is above 88%. The technical efficiency is 82% for particles with p_T of [100, 300] MeV, and increases to above 97% for particles with $p_T > 700$ MeV. Figure 5 also shows the η distribution of particles with $p_T > 500$ MeV as well as the tracking efficiency as a function of the particle η. The physics efficiency is higher in the *barrel* region of the detector (volumes 8,13,17 in Figure 1), while the technical efficiency is almost flat across the η range. In Figure 5 the p_T and η of the matched truth particle were used, rather than the p_T and η of the reconstructed track. We leave a study of track quality and detector resolution effects for future work.

4.1.2 Systematic Studies

Before using a tracking algorithm in production, it is necessary to measure its sensitivity to systematic effects, including pile-up, noise and digitization errors, and uncertainties in the measurement of detector properties (alignment, rotation, magnetic field map, etc.).

Measuring precisely the impact of pile-up collisions on tracking performance is beyond the scope of this work, but we can estimate pile-up's impact on the pipeline tracking performance by plotting efficiency and purity as a function of the number of spacepoints in the detector. Figure 6 shows that the effect of the increased detector occupancy is a smooth, and relatively small performance degradation.

The impact of noise spacepoints can be estimated using the TrackML dataset by studying the inference performance of the tracking pipeline, trained without any noise spacepoints, as a function of the fraction of noise spacepoints (up to a maximum of 20% of the total). Table 1 shows the technical tracking efficiency and purity for different noise levels. The efficiency decreases by $\simeq 1.6\%$ and the purity by $\simeq 5.4\%$ when 20% of noise spacepoints are presented. The loss of efficiency happens primarily for particles with $p_T < 500$ MeV (Figure 7).

Detector misalignment effects are approximated by shifting by up to 1 mm the x-axis of all spacepoints in the inner-most TrackML barrel detector layer or the four innermost layers (volume 8 in Figure 1). In both cases, the impact on the tracking efficiency is less than 0.1%. However, studying in depth misalignments, and other detector effects, requires access to experiment detailed detector simulation data. We leave these studies as future work to be performed in collaboration with each experiment.

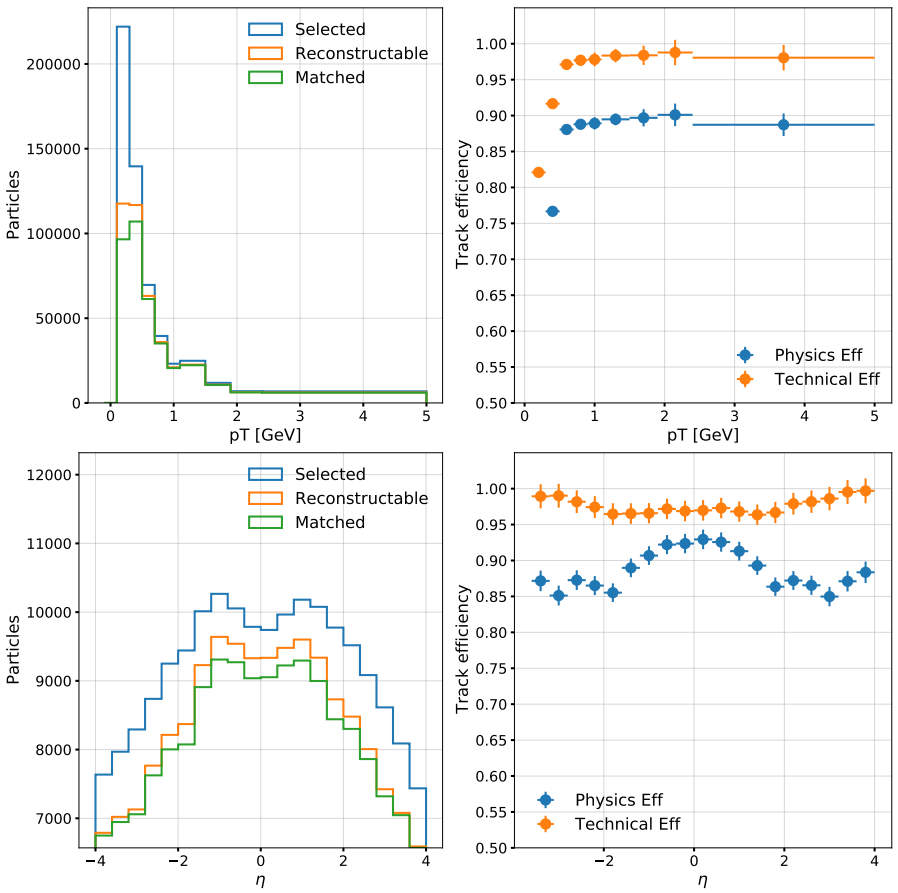

Figure 5. Top row: selected, reconstructable, and matched particles (left) and tracking efficiency (right) as a function of p_T for particles with $|\eta| < 4$. Bottom row: selected, reconstructable, and matched particles (left) and tracking efficiency (right) as a function of η for $p_T > 0.5$ GeV. The definition of "selected", "reconstructable", and "matched" can be found in § 4.1.1

4.2 Distributed Training Performance

It takes about 1.5 days to train the Exa.TrkX pipeline on one Nvidia A100 GPU for one set of hyper-parameters. It is therefore desirable to use distributed training to parallelize model training and hyper-parameter optimization (HPO). This study relied on data parallel training [38] implemented using Horovod [39] and Tensorflow's TF.DISTRIBUTED framework [40]. Horovod supports distributed training across multiple nodes, while tf.distributed allows to use the same code across CPUs, TPUs, and GPUs.

For this study, the TrackML pipeline is trained on up to 64 Nvidia V100 GPUs across eight NERSC Cori-GPU computing nodes. Using the Horovod framework (Figure 8), training time is reduced from 22 minutes, with 1 GPU, to 0.5 minutes with 64 GPUs[3]. The strong scaling efficiency[4] is about 90% with 2 GPUs and 75% with 8 GPUs. This deviation from ideal scaling is due to the model setup time and data movement costs.

[3] All measurements in this section were taken training on spacepoints from the barrel region of the TrackML detector. For comparison, training with spacepoints from the whole detector takes ≃70 minutes per epoch on one Nvidia A100 GPU

[4] defined as $t_1/(N \times t_N) * 100\%$ where t_N is the time to train on a fixed total number of events across N GPUs.

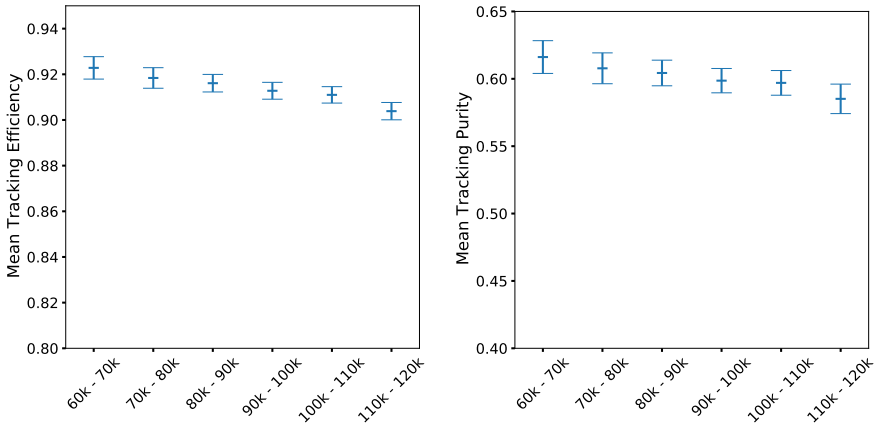

Figure 6. Mean and standard deviation of the technical efficiency (left) and purity (right) as a function of the total number of spacepoints in an event.

Noise	Efficiency	Purity
0	91.5	59.3
4%	91.5	59.3
8%	91.1	58.0
12%	90.9	56.8
16%	92.2	54.8
20%	89.9	53.9

Table 1. Technical efficiency and purity for different noise fractions $(N_{\text{spp}}^{\text{noise}}/N_{\text{spp}}) * 100\%$

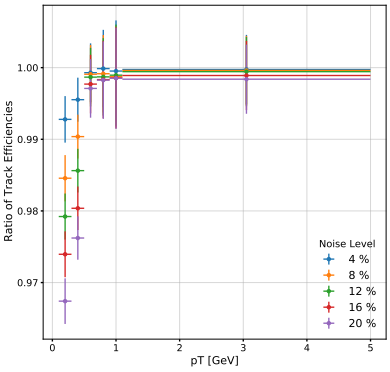

Figure 7. *Relative* technical efficiency as a function of p_{T}. Each curve shows the ratio of eff(noise = N%)/eff(noise = 0).

Figure 8 also shows the scaling behaviour of the TF.DISTRIBUTED implementation. Since TF.DISTRIBUTED requires all input data to be of the same size, we have to pad all input graphs to a fixed size. This essentially doubles the time needed to train one epoch, that increases from 22 minutes for dynamic input graph sizes to 41 minutes for constant graph sizes. Leaving aside this fixed overhead, TF.DISTRIBUTED appears to scale better than Horovod, achieving $\simeq$ 85% strong scaling efficiency with 8 GPUs.

4.3 Inference performance on CPU and GPU

It is crucial to characterize the computational cost of the end-to-end learned tracking algorithm. Our algorithm is optimized for running inference so that the whole inference pipeline can be run in GPUs with the help of PYTORCH, TENSORFLOW, and CUGRAPH. The execution time for the inference pipeline has been measured on two hardware platforms: Nvidia V100 GPUs with 16 GB on-board memory, and Intel Xeon 6148s (Skylake) CPUs with 40 cores

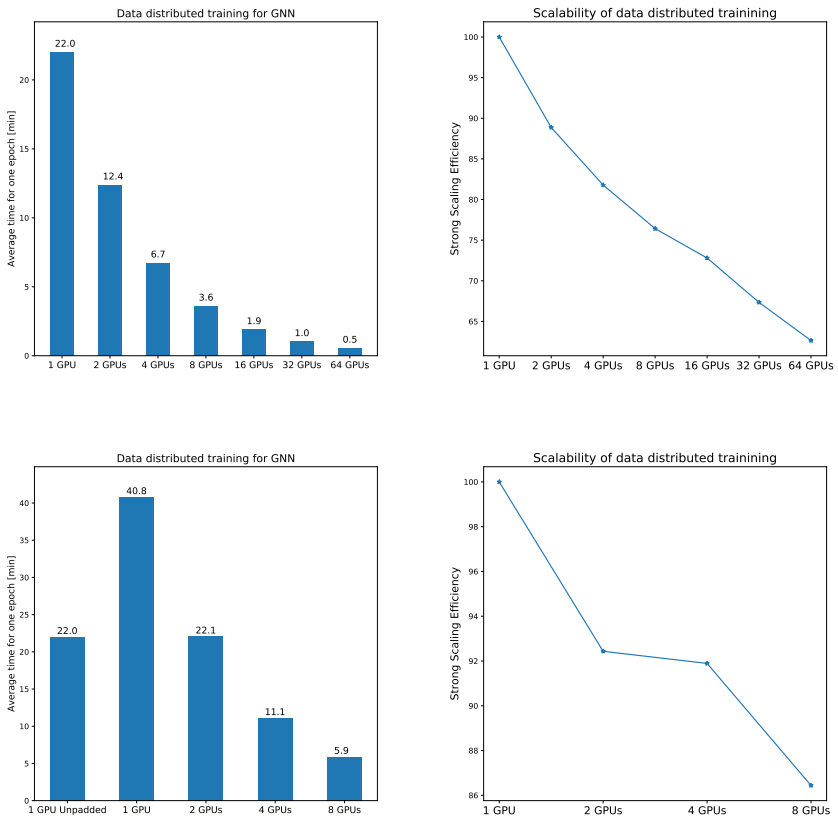

Figure 8. Time per training epoch (left) and Strong scaling efficiency (right) for GNN's distributed training. The top row refers to the Horovod implementation, the bottom row to the TF.DISTRIBUTED one. The first bin in the bottom left diagram refers to the serial case, in which the input graph is not padded.

and 192 GB memory per node. The inputs to the filtering step do not fit into the GPU memory. Therefore, edge filtering for one event is executed in mini-batches with a fixed batch size of 800k edges. Typically, the inputs to the filtering from one event are split into seven batches, leading to additional computational cost for moving data from host to GPU. The peak GPU memory consumption is about 15.7 GB as obtained from the Nvidia profiling tool.

Averaging over 500 events, it takes 2.2 ± 0.3 wall-clock seconds per event (as measured by measured by the python module *time*) to run the inference pipeline on the GPU and 202 ± 35 seconds to run it on a single CPU core. This total execution time includes every step of the calculation, and in particular the time needed to move data from host to GPU. Table 2 breaks down the wall-clock time for the most significant steps of the pipeline. For these step-by-step measurements, we force the pipeline to execute serially by calling `torch.cuda.synchronize` after each step. The results show how the graph creation and filtering steps are the biggest targets for further optimization.

In addition, Figure 9 shows how the total inference time depends almost linearly on the number of spacepoints in the event for both CPUs and GPUs. The step-like dispersion in the GPU case is due to the splitting of the inputs to the filtering step into mini-baches. A step-like jump indicates one more mini-batch is added.

	Wall time [s] on Xeon 6148s single core	Wall time [s] on Nvidia V100 synchronous
Data Loading	0.0049 ± 0.0153	0.0023 ± 0.0003
Embedding	3.02 ± 0.39	0.024 ± 0.003
Build Edge	66 ± 13	0.76 ± 0.10
Filtering	99 ± 19	1.57 ± 0.34
GNN	27 ± 2	0.45 ± 0.06
Labeling	3.23 ± 0.34	0.08 ± 0.01
Total (sync)	202 ± 35	3.3 ± 0.5

Table 2. Average inference time for synchronous execution of the TrackML pipeline benchmarked on CPUs and GPUs. The total inference time comprises all the steps including ones not listed in the table.

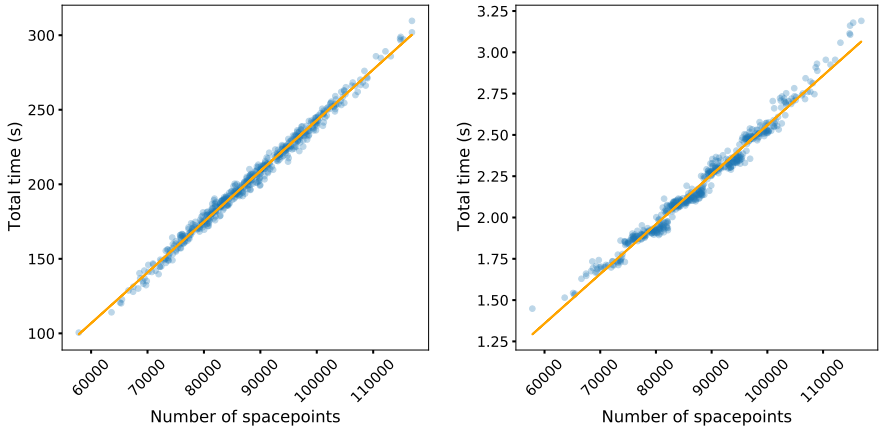

Figure 9. Total inference time as a function of number of spacepoints in each event for CPUs (left) and GPUs (right).

Many optimizations were introduced to the pipeline in order to achieve these GPU timings, which before optimization took over 20 seconds per event. These improvements include porting all data processing to the GPU-accelerated CuPy library [41], writing custom sparse operations for graph processing (e.g. doublet-to-triplet conversion [42], graph intersection methods), using FAISS [43] for large-k NN graph construction, and performing track labelling with CuGraph on GPU [44]. These improvements are specific to the inference pipeline; training optimizations will be discussed in the following section, and ongoing developments in § 6.

5 Discussion

The performance given above is the result of experimentation across various feature sets, architectures, model configurations and hyperparameters. It has also been necessary to overcome a variety of training hurdles in terms of memory and computational availability. We describe here training and inference details that should allow a reader to reproduce these results on the provided codebase.

5.1 Feature Set

The input dataset includes both spatial co-ordinates and highly granular pixel cluster information. Graph construction (the second pipeline step in Figure 4, that includes learned embedded space model and edge filter model) appears to benefit significantly from the cluster shape information, approximately doubling the purity for a held fixed high efficiency. The summary cluster shape statistics include the number of channels and the total charge deposited, as well as local and global representations of the cluster as a high-level feature vector. Details about the calculation of this feature vector as well as a thorough exploration of the effect of cluster shape information on seeding performance are provided in Ref. [30]. Cluster shape information does not appear to improve the performance of the GNN, and in fact seems to degrade it. This suggests that the width of the GNN hidden layers is not great enough to capture the functional relationship of cluster information between nodes. Scaling to a width that properly explores this question would require more memory than available on the Nvidia V100 GPUs used for this study.

Depending on the final goal of the pipeline, further features can be included in the loss calculation in order to bias the model towards desired regions. For example, if our aim is to maximize the TrackML score (described in Ref. [2]) — a weighting function s_i that places more importance on a spacepoint i from a longer and higher p_T track, and in the first and last sets of detector layers – we can weight-up true edges by this function, normalized to have a mean of weight = 1. To measure the performance of models trained to this goal, we introduce a *weighted* purity measure. Weighted purity is defined as a function the TrackML weights w_{ij} and the truth $y_{ij} \in \{0, 1\}$ of each edge connecting spacepoint i and spacepoint j,

$$\mathrm{pur}_{\mathrm{weighted}} = \frac{\sum_{ij} w_{ji} y_{ij}}{\sum_{ij} w_{ij}}, \qquad w_{ij} = \begin{cases} \frac{1}{2}(s_i + s_j), \text{ if } y_{ij} = 1 \\ 1, \text{ if } y_{ij} = 0 \end{cases} \tag{4}$$

We see significant improvements in this metric when validating on the weighted model: the Embedding Network improves from a weighted purity of 1.7% ± 0.2% to 2.0% ± 0.3%, while the Filter Network improves from a weighted purity of 8.4% ± 0.6% to 11.7% ± 1.0%. Given this weighting, the model learns to prioritize higher p_T and longer tracks, while disregarding less informative tracks. Using this bias, we can achieve the same TrackML score with a constructed graph size reduced by approximately 25%. Using this technique to improve the TrackML score is an ongoing work.

5.2 Graph Construction

Having chosen a feature set, to train the learned embedding space we use a training paradigm commonly referred to as a Siamese Network [45], where a particular spacepoint - called the *source* - is run through an MLP, here 6 layers each with 512 hidden channels, hyperbolic tan activations, and layer normalization. The final layer of the MLP takes the features to an 8-dimensional latent space. A different, comparison spacepoint - called the *target* - is also run through this same Embedding Network, and the L2 norm distance d in the latent space between the source and target enters a comparative hinge loss

$$\mathcal{L}_{\mathrm{hinge}} = \begin{cases} d^p, \text{ if } y_{ij} = 1 \\ \max(0, 1 - d^p), \text{ if } y_{ij} = 0 \end{cases} \tag{5}$$

where p is a hyperparameter that we choose to be 2.

If the source i and target j spacepoints share an edge in the event's truth graph [5], we designate them as neighbours with $y_{ij} = 1$, otherwise they are designated $y_{ij} = 0$. In this way, the hinge loss draws together truth graph neighbors and repels non-neighbors.

Training performance of the Embedding Network is highly dependent on choice of source-target example pairs. In early epochs, it is enough to choose random pairs. However, at some point, many random pairs will contribute no gradient to the loss, as they will be separated by a distance greater than the margin. At that point, it is useful to implement hard negative mining [46]. We run a GPU-optimised k-nearest-neighbor (KNN) algorithm [6] to mine examples around each source vector, within the hinge margin $d = 1$. The computational overhead of the KNN step is significantly offset by the examples mined which all contribute to the loss.

A similar technique is used in the Filter Network, where the vast majority of the edges produced from the graph construction in the embedded space are easy to classify as fake. This is already a highly imbalanced dataset, with around 98.5% of edges fake. Again, within several epochs, the Filter Network is able to classify many of these as fake, so we balance each batch with all true edges, the same number of hard negatives (i.e. negatives the filter is unsure of) and the same number of easy negatives (to maintain performance on these edges). The Filter Network is a MLP that takes the 24-feature concatenated edge features and feeds forward through 3 layers of 1024 hidden channels, to a binary cross-entropy loss function.

5.3 GNN Edge Classification

In choosing the best GNN architecture, memory usage remains a significant constraint. The Interaction Network (IN) [36] presented in these results does appear to marginally attain the best performance against Attention Graph Neural Networks (AGNN) [14, 48] – the other class of GNN considered for the pipeline. However, both of these networks require gradients to be retained in memory for every graph edge. Indeed, this anisotropic treatment of edges (i.e. a node is able to receive the messages of each of its neighbors in a non-uniform way) is what allows these two architectures to be so expressive. Depending on hardware availability, we have found two solutions to the memory constraint. Access to next-generation Nvidia A100 GPUs allowed an IN to be trained with 8 steps of message passing, aggregating edge features at each node, and each node and concatenated edge features passing through two-layer MLPs of [128, 64] hidden features and ReLU activations [49]. Choice of aggregation function should be permutation invariant. In this work, we take it to be a summation.

For lower-memory GPUs, such as the Nvidia V100, we attained similar performance training the AGNN architecture, with [64, 64, 64]-channel MLPs applied to each edge and node. Adding residuals [50] across the 8 message passing steps greatly improved performance in this case. To fit full-event training on a single V100, it was necessary to employ various techniques, such as mixed precision training and gradient checkpointing. The latter stores only the input of each layer, not the gradients. On the backward pass, gradients are recalculated on the fly, allowing for a 4x reduction in memory usage for an 8-iteration GNN. Another technique explored is to split the events piecemeal and train on each piece as a standalone batch. There is a noticeable impact on performance due to messages being interrupted at the graph edges. In future work, we will present ongoing efforts to parallelise these graph pieces across multiple GPUs, retaining the high performance that full-event training allows.

[5] one can also designate $y_{ij} = 1$ for source and target in the same track, rather than *immediate neighbors* in the track. This does lead to similar performance in later stages of the pipeline, but the more lax concept of truth leads to graphs around three times more dense than the strict track neighbor definition.

[6] We use two high-performance libraries, FAISS [43] and Pytorch3D [47], depending on number of nearest neighbors k. Fastest performance is obtained with FAISS for $k > 35$, Pytorch3D for $k <= 35$.

5.4 Physics-inspired data augmentation

Preliminary work on using coordinate transforms to augment the training data has been explored with varying degrees of success. In this study, focused on track seeding, only the innermost detector layers (volumes 7-9 in Figure 1) were used.

One promising approach is to make a copy of each graph in the training set that has been reflected across the phi-axis [51]. The phi reflection creates the charge conjugate graph and helps to balance any asymmetry between positive and negatively charged particles within the training set. This performance boost comes at the cost of doubling the training time.

A second promising trick is to use a Hough Transform [6, 7] on the graph to create edge features. Using the Hough parameters as edge features boosts efficiency by $\simeq 2\%$ and purity by $\simeq 1\%$. A further efficiency boost of $\simeq 3\%$ (and $\simeq 2\%$ to purity) comes from using the Hough accumulator to extract an edge weight. This edge weight effectively pools information from every node, and therefore comes at a large computational cost (filling the accumulator in Hough space). On the other hand, the Hough parameters can be computed quickly from the two nodes that define the edge.

6 Conclusion and Future Work

This works shows how a machine learning tracking pipeline based on geometric learning can achieve state-of-the-art computing performance on commercial GPUs. Crucially the computing performance scales linearly with the number of spacepoints, showing great promise for the next generation of HEP experiments.

Within the simplifying assumptions of the TrackML dataset, we have shown how the Exa.TrkX pipeline could meet the tracking performance requirements of current collider experiments. Preliminary studies suggest that this performance should be robust against systematic effects like detector noise, misalignment, and pile-up.

Much remains to be done to validate these promising results in the "real world." To this end, the Exa.TrkX project is collaborating with physicists from ATLAS [52], CMS [53], DUNE [54], ICARUS [55], and MuonE [56].

The goal is to adapt the Exa.TrkX pipeline to each experiment's needs and simulated datasets, measure its performance and robustness against systematic effects according to the experiment metrics. For example, it is crucial for HL-LHC experiments to study the performance of tracking algorithms in dense environments, like high-p_{T} jets. Given the interest in long-lived particle observation at the HL-LHC, it will also be important to study the performance of the Exa.TrkX pipeline for tracks coming from a displaced vertex[7].

On the computational side, there are several optimization opportunities that need to be explored systematically including mixed precision training, multi-GPU training and inference with graph data parallelisation (that is, one event spread across multiple GPUs) [57]; locality sensitive hashing to speed-up KNN/graph construction stage [58], model quantization, operator fusion and other improvements with TensorRT [59], clustering of final node embeddings rather than hard connected components method with GravNet-style architectures [60].

The distributed training results presented in this work are promising but still preliminary. To fully exploit the capabilities of upcoming HPC systems and to further reduce training time while potentially pushing further on model size, it will be beneficial to perform further studies on large scale training of GNNs for track reconstruction. Given the size of the input graphs, this problem may be amenable to training techniques which parallelise the processing of input graphs across multiple GPUs in training.

[7] it may be worth noticing that in LArTPC applications [28] all tracks come from a displaced vertex.

Finally, it will be interesting to measure the computing performance of (parts of) the Exa.TrkX pipeline on domain-specific accelerators like Google TPU [61] and GraphCore IPU [62], comparing power consumption, latency and throughput with "traditional" GPUs.

7 Software availability

A growing number of groups are currently studying the application of graph networks to HEP reconstruction (see [63] for a recent review). Some of these works [24, 27–31, 33–35] have strong connections with the Exa.TrkX project. To promote collaboration and reproducibility, the Exa.TrkX software is available from the HEP Software Foundation's Trigger and Reconstruction GitHub[8]. A pipeline of re-usable modules is implemented within the Pytorch Lightning system, which allows for uncluttered and simple model definitions. As each stage of the pipeline is dependent, logging utilities are integrated that allow a specific combination of stages and hyperparameters to be trackable and reproducible. Extensive documentation is provided to help track reconstruction groups start exploring geometric learning. The roadmap for this repository includes adding performance metrics to the codebase; a taxonomy of model features; and short tutorials in each of the available applications.

Acknowledgements

This research was supported in part by the U.S. Department of Energy's Office of Science, Office of High Energy Physics, of the US Department of Energy under Contracts No. DE-AC02-05CH11231 (CompHEP Exa.TrkX) and No. DE-AC02-07CH11359 (FNAL LDRD 2019.017); and by the National Science Foundation under Cooperative Agreement OAC-1836650. This research was supported in part by the Exascale Computing Project (17-SC-20-SC), a joint project of the Office of Science and National Nuclear Security Administration.

This research used resources of the National Energy Research Scientific Computing Center (NERSC), a U.S. Department of Energy Office of Science User Facility located at Lawrence Berkeley National Laboratory, operated under Contract No. DE-AC02-05CH11231. We are grateful to Google Co. for providing early access to Nvidia A100 instances in the context of the US ATLAS/Google Cloud Platform collaboration.

Finally, we thank Marcin Wolter (IFJ PAN), Kesheng Wu and Alex Sim (LBNL) for the useful discussions.

References

[1] I.B. Alonso, O. Brüning, P. Fessia, M. Lamont, L. Rossi, L. Tavian, M. Zerlauth, CERN Yellow Report **10** (2020)

[2] S. Amrouche, L. Basara, P. Calafiura, V. Estrade, S. Farrell, D.R. Ferreira, L. Finnie, N. Finnie, C. Germain, V.V. Gligorov et al., in *The NeurIPS 2018 Competition* (Springer International Publishing, 2019), pp. 231–264, 1904.06778

[3] A. Strandlie, R. Frühwirth, Rev. Mod. Phys. **82**, 1419 (2010)

[4] ATLAS Collaboration, Eur. Phys. J. C **77**, 673 (2017), 1704.07983

[5] S. Chatrchyan et al. (CMS), JINST **9**, P10009 (2014), 1405.6569

[6] R.O. Duda, P.E. Hart, Commun. ACM **15**, 11–15 (1972)

[7] J. Gradin, M. Mårtensson, R. Brenner, JINST **13**, P04019 (2018), 1709.01034

[8] https://hsf-reco-and-software-triggers.github.io/Tracking-ML-Exa.TrkX

[8] D. Funke, T. Hauth, V. Innocente, G. Quast, P. Sanders, D. Schieferdecker, J. Phys. Conf. Ser. **513**, 052010 (2014)

[9] D. Rohr, S. Gorbunov, M.O. Schmidt, R. Shahoyan, EPJ Web Conf. **214**, 01050 (2019), 1905.05515

[10] ATLAS Collaboration, *Computing and Software Public Results* (2017), https://twiki.cern.ch/twiki/bin/view/AtlasPublic/ComputingandSoftwarePublicResults

[11] CMS Collaboration, *CMS Tracking POG Performance Plots For 2017 with PhaseI pixel detector* (2017), https://twiki.cern.ch/twiki/bin/view/CMSPublic/TrackingPOGPerformance2017MC

[12] ATLAS Collaboration, Tech. Rep. ATL-PHYS-PUB-2019-041, CERN, Geneva (2019), https://cds.cern.ch/record/2693670

[13] HEP.TrkX, *HEP advanced tracking algorithms with cross-cutting applications* (2016), https://heptrkx.github.io/

[14] S. Farrell et al., *Novel deep learning methods for track reconstruction*, in *4th International Workshop Connecting The Dots 2018 (CTD2018) Seattle, Washington, USA, March 20-22, 2018* (2018), 1810.06111

[15] Exa.TrkX, *HEP advanced tracking algorithms at the exascale* (2019), https://exatrkx.github.io/

[16] M.M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, P. Vandergheynst, IEEE Signal Processing Magazine **34**, 18 (2017)

[17] N. Choma, D. Murnane, X. Ju, P. Calafiura, S. Conlon, S. Farrell, Prabhat, G. Cerati, L. Gray, T. Klijnsma et al., *Track seeding and labelling with embedded-space graph neural networks* (2020), 2007.00149

[18] S. Farrell et al., *The HEP.TrkX Project: deep neural networks for HL-LHC online and offline tracking*, in *Proceedings, Connecting The Dots / Intelligent Tracker (CTD/WIT 2017): Orsay, France, March 6-9, 2017* (2017), Vol. 150, p. 00003

[19] V. Bertacchi (CMS), *DeepCore: Convolutional Neural Network for high p_T jet tracking*, in *Connecting the Dots and Workshop on Intelligent Trackers* (2019), 1910.08058

[20] S. Hochreiter, J. Schmidhuber, Neural Comput. **9**, 1735 (1997)

[21] A. Tsaris, D. Anderson, J. Bendavid, P. Calafiura, G. Cerati, J. Esseiva, S. Farrell, L. Gray, K. Kapoor, J. Kowalkowski et al., Journal of Physics: Conference Series **1085**, 042023 (2018)

[22] S. Amrouche et al., *The Tracking Machine Learning challenge : Accuracy phase* (2019), 1904.06778

[23] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, Y. Bengio, *Learning phrase representations using rnn encoder-decoder for statistical machine translation* (2014), 1406.1078

[24] C. Biscarat, S. Caillou, C. Rougier, J. Stark, J. Zahreddine, *Towards a realistic track reconstruction algorithm based on graph neural networks for the hl-lhc* (2021), 2103.00916

[25] X. Ju et al., *Graph Neural Networks for Particle Reconstruction in High Energy Physics detectors*, in *33rd Annual Conference on Neural Information Processing Systems* (2020), 2003.11603

[26] T.D. Le, H. Imai, Y. Negishi, K. Kawachiya, *Tflms: Large model support in tensorflow by graph rewriting* (2019), 1807.02037

[27] J. Pata, J. Duarte, J.R. Vlimant, M. Pierini, M. Spiropulu, *MLPF: Efficient machine-learned particle-flow reconstruction using graph neural networks* (2021), 2101.08578

[28] J. Hewes, A. Aurisano, G. Cerati, J. Kowalkowski, C. Lee, W. keng Liao, A. Day, A. Agrawal, M. Spiropulu, J.R. Vlimant et al., *Graph neural network for object reconstruction in liquid argon time projection chambers* (2021), 2103.06233

[29] A. Heintz et al., *Accelerated Charged Particle Tracking with Graph Neural Networks on FPGAs*, in *34th Conference on Neural Information Processing Systems* (2020), 2012.01563

[30] P.J. Fox, S. Huang, J. Isaacson, X. Ju, B. Nachman, *Beyond 4d tracking: Using cluster shapes for track seeding* (2020), 2012.04533

[31] S. Amrouche, M. Kiehn, T. Golling, A. Salzburger, *Hashing and metric learning for charged particle tracking* (2021), 2101.06428

[32] W. Guan, G. Perdue, A. Pesah, M. Schuld, K. Terashi, S. Vallecorsa, J.R. Vlimant (2020), 2005.08582

[33] C. Tüysüz, F. Carminati, B. Demirköz, D. Dobos, F. Fracas, K. Novotny, K. Potamianos, S. Vallecorsa, J.R. Vlimant, EPJ Web Conf. **245**, 09013 (2020), 2003.08126

[34] C. Tüysüz, F. Carminati, B. Demirköz, D. Dobos, F. Fracas, K. Novotny, K. Potamianos, S. Vallecorsa, J.R. Vlimant, *A Quantum Graph Neural Network Approach to Particle Track Reconstruction* (2020), 2007.06868

[35] C. Tüysüz, K. Novotny, C. Rieger, F. Carminati, B. Demirköz, D. Dobos, F. Fracas, K. Potamianos, S. Vallecorsa, J.R. Vlimant, *Performance of Particle Tracking Using a Quantum Graph Neural Network* (2020), 2012.01379

[36] P.W. Battaglia, R. Pascanu, M. Lai, D.J. Rezende, K. Kavukcuoglu, *Interaction networks for learning about objects, relations and physics* (2016), 1612.00222

[37] ATLAS Collaboration, Tech. Rep. CERN-LHCC-2017-021. ATLAS-TDR-030, CERN, Geneva (2017), https://cds.cern.ch/record/2285585

[38] T. Ben-Nun, T. Hoefler, *Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis* (2018), 1802.09941

[39] A. Sergeev, M.D. Balso, *Horovod: fast and easy distributed deep learning in TensorFlow* (2018), 1802.05799

[40] M. Abadi et al., *TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems* (2016), 1603.04467

[41] R. Okuta, Y. Unno, D. Nishino, S. Hido, Crissman, *CuPy : A NumPy-Compatible Library for NVIDIA GPU Calculations*, in *31st Conference on Neural Information Processing Systems (NIPS 2017)* (2017), http://learningsys.org/nips17/assets/papers/paper_16.pdf

[42] M. Fey, J.E. Lenssen, *Fast Graph Representation Learning with PyTorch Geometric*, in *ICLR Workshop on Representation Learning on Graphs and Manifolds* (2019)

[43] J. Johnson, M. Douze, H. Jégou, *Billion-scale similarity search with GPUs* (2017), 1702.08734

[44] *CuGraph*, https://github.com/rapidsai/cugraph (2020), accessed 2021-03-01

[45] D. Chicco, *Siamese Neural Networks: An Overview* (Springer US, New York, NY, 2021), pp. 73–94, ISBN 978-1-0716-0826-5, https://doi.org/10.1007/978-1-0716-0826-5_3

[46] B. Harwood, V.K. B G, G. Carneiro, I. Reid, T. Drummond, *Smart Mining for Deep Metric Learning*, in *ICCV 2017 : International Conference on Computer Vision* (2017), pp. 2840–2848

[47] N. Ravi, J. Reizenstein, D. Novotny, T. Gordon, W.Y. Lo, J. Johnson, G. Gkioxari, *Accelerating 3D Deep Learning with PyTorch3D* (2020), 2007.08501

[48] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, Y. Bengio, *Graph attention networks* (2017), 1710.10903

[49] X. Glorot, A. Bordes, Y. Bengio, *Deep Sparse Rectifier Neural Networks*, in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, edited by G. Gordon, D. Dunson, M. Dudík (JMLR Workshop and Conference Proceedings, Fort Lauderdale, FL, USA, 2011), Vol. 15 of *Proceedings of Machine Learning Research*, pp. 315–323, http://proceedings.mlr.press/v15/glorot11a.html

[50] K. He, X. Zhang, S. Ren, J. Sun, *Deep residual learning for image recognition* (2015), 1512.03385

[51] L. Perez, J. Wang, CoRR **abs/1712.04621** (2017), 1712.04621

[52] ATLAS Collaboration (ATLAS), JINST **3**, S08003 (2008)

[53] S. Chatrchyan et al. (CMS), JINST **3**, S08004 (2008)

[54] *Deep underground neutrino experiment*, http://www.dunescience.org/

[55] L. Bagby, B. Baibussinov, B. Behera, V. Bellini, R. Benocci, M. Betancourt, M. Bettini, M. Bonesini, T. Boone, A. Braggiotti et al., Journal of Instrumentation **16**, P01037 (2021)

[56] G. Abbiendi, C.M.C. Calame, U. Marconi, C. Matteuzzi, G. Montagna, O. Nicrosini, M. Passera, F. Piccinini, R. Tenchini, L. Trentadue et al., The European Physical Journal C **77**, 139 (2017)

[57] S. Scardapane, I. Spinelli, P.D. Lorenzo, IEEE Transactions on Signal and Information Processing over Networks **7**, 87–100 (2021)

[58] P. Indyk, R. Motwani, *Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality*, in *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing* (Association for Computing Machinery, New York, NY, USA, 1998), STOC '98, p. 604–613, ISBN 0897919629, https://doi.org/10.1145/276698.276876

[59] *NVIDIA TensorRT*, https://docs.nvidia.com/deeplearning/tensorrt/index.html (2020), accessed 2021-03-01

[60] S.R. Qasim, J. Kieseler, Y. Iiyama, M. Pierini, Eur. Phys. J. C **79**, 608 (2019), 1902.07987

[61] N.P. Jouppi et al., SIGARCH Comput. Archit. News **45**, 1–12 (2017), 1704.04760

[62] Z. Jia, B. Tillman, M. Maggioni, D.P. Scarpazza (2019), 1912.03413

[63] J. Duarte, J.R. Vlimant, *Graph neural networks for particle tracking and reconstruction* (2020), 2012.01249