Track reconstruction for the ATLAS Phase-II Event Filter using GNNs on FPGAs

Sebastian Dittmeier[1,*] on behalf of the ATLAS TDAQ collaboration [1]

[1]Physikalisches Institut, Ruprecht-Karls-Universität Heidelberg,
Im Neuenheimer Feld 226, 69120 Heidelberg

Abstract. The High-Luminosity LHC (HL-LHC) will provide an order of magnitude increase in integrated luminosity and enhance the discovery reach for new phenomena. The increased pile-up necessitates major upgrades to the ATLAS detector and trigger. The Phase-II trigger will consist of two levels, a hardware-based Level-0 trigger and an Event Filter (EF) with tracking capabilities. Within the Trigger and Data Acquisition group, a heterogeneous computing farm consisting of CPUs and potentially GPUs and/or FPGAs is under study, together with the use of modern machine learning algorithms such as Graph Neural Networks (GNNs).
GNNs are a powerful class of geometric deep learning methods for modelling spatial dependencies via message passing over graphs. They are well-suited for track reconstruction tasks by learning on an expressive structured graph representation of hit data and considerable speedup over CPU-based execution is possible on FPGAs.
The focus of this publication is a study of track reconstruction for the Phase-II EF system using GNNs on FPGAs. We explore each of the steps in a GNN-based EF tracking pipeline: graph construction, edge classification using an interaction network, and track reconstruction. Several methods and hardware platforms are under evaluation, studying resource utilisation and minimization of model size using quantization aware training, while simultaneously retaining high track reconstruction efficiency and low fake rates required for the EF tracking system.

1 Introduction

The ATLAS experiment [2] will be upgraded for the operation at the High-Luminosity LHC [3, 4] to fully exploit its physics potential. This includes replacement of old and installation of new detectors [5–7], as well as changes to the trigger and data acquisition system (TDAQ) [8], such that trigger thresholds can possibly be set as low as they have been during LHC Run 1. To deal with the increased Level-0 trigger rates, the higher output data rates, and the larger event sizes compared to previous runs, all three major TDAQ components, the hardware-based Level-0 trigger system, the Event Filter (EF) system, and the data acquisition system, have to be upgraded [8]. For the EF system, a heterogeneous computing farm consisting of CPUs and potentially GPUs and/or FPGAs is under study [9], which could be

[*]e-mail: sebastian.dittmeier@cern.ch

more power efficient than a CPU-only solution. Since most of the computing resources are needed for the online track reconstruction within the Inner Tracker (ITk) [5, 6], the Event Filter Tracking (EF Tracking) is the main driver for the system requirements. Within the EF Tracking project the different hardware accelerators are being explored together with the application of novel track reconstruction algorithms and machine learning methods [9]. One of the approaches currently under study is the application of Graph Neural Networks (GNNs) and their execution on FPGAs.

2 Track reconstruction with Graph Neural Networks on FPGAs

Over the last years, the suitability of GNNs has been demonstrated for a variety of high-energy physics use cases, such as jet tagging [10], calorimeter energy measurements [11], and in particular charged particle track reconstruction [12–18]. Recently, excellent tracking performance has been demonstrated using ATLAS ITk simulation samples [19]. The work presented here builds on top of the developments from the Exa.TrkX [13] and GNN4ITK [19] projects, and makes use of the same track reconstruction pipeline, see figure 1.

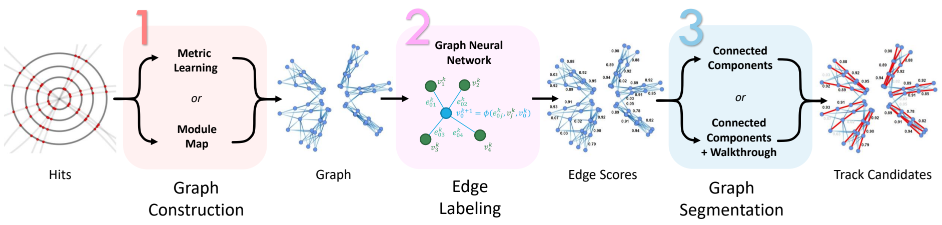

Figure 1. Schematic overview of the GNN-based track finding pipeline [20], which is divided into three individual steps: a graph construction step, a classification step (edge labeling) using a GNN, and the final graph segmentation step to isolate track candidates.

The overall goal of the graph construction step is to maximize efficiency for creating as many true edges as possible, while obtaining a reasonably pure graph by limiting the amount of false edges. A graph is constructed from the point cloud of detector hits, which are represented as the nodes of the graph. Two methods for graph construction are currently available and investigated: metric learning, and module map. In metric learning [19], the node features are embedded into a latent space using a Multi Layer Perceptron (MLP). Nodes that are close to each other in the latent space are then connected with edges. In the module map approach [19], a map of modules is derived from detector simulations, which contains connections between modules following the trajectories of charged particles in the detector.

The graph is passed through the graph neural network. More specifically, an interaction network [21] is applied iteratively to the latent features of the graph, which are obtained from node and edge encoding MLPs. As a last step, the latent features of each edge are transformed into an edge classification score, which describes the probability that the edge originates from a true particle track. Applying a threshold to this score, false edges can be removed from the graph.

Track candidates are then formed by segmenting the graph in the third step of the pipeline. Sets of connected components are created, and a walkthrough algorithm is applied in case that multiple paths exist for a specific node.

Energy efficiency and data throughput are essential metrics for any tracking solution currently being developed for the ATLAS Event Filter system. Previous studies of implementing graph neural networks on FPGAs for high energy physics applications show a large potential for speed-up and energy savings compared to CPU and GPU implementations [22–24], though many of these studies target low latency applications with tighter constraints than the Event Filter system. However, this does not diminish the challenge of retaining the performance of e.g. the GNN4ITK GPU implementation using potentially smaller models on FPGAs.

3 Hardware developments: module map and walkthrough

A VHDL implementation of the module map approach is currently under development. The incoming ITk hit data is decoded into event headers, raw hit data, and module IDs. This information is stored and used within the Hit Buffers. When the event is fully loaded, the Output Ranager retrieves the hit data and edges are created according to the content stored in the Module Map RAM. The edges are then concatenated to create the full event graph, which can be passed to the graph neural network for processing. See figure 2 for a complete schematic.

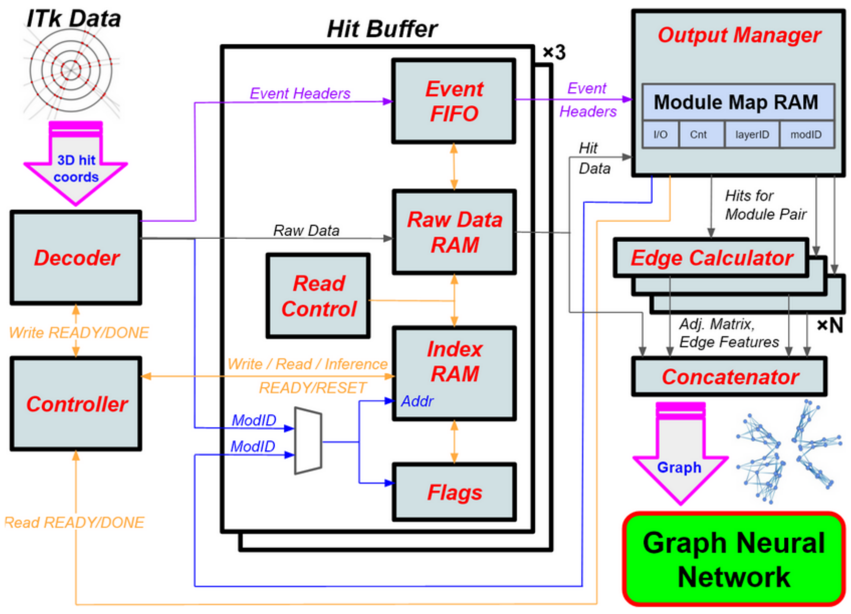

Figure 2. Block diagram of the module map approach implemented in VHDL.

A first version of the walkthrough algorithm has been implemented in VHDL. Preliminary resource estimates have been obtained by synthesizing the design for an Intel Stratix 10 GX FPGA. In this version, a linear scaling between look up tables and graph edges is observed; 3000 graph edges lead to about 10 percent of look up tables being used. Due to the preliminary nature of the implementation, optimizations are possible and will be done once the rest of the pipeline is more mature. For the usage of the VHDL blocks in the full pipeline for EF Tracking, it is foreseen that they will be wrapped into kernels that can then be included in the full data flow architecture using the AMD Vitis [25] or Intel OneAPI [26] tools.

4 Model optimization studies: metric learning

A model resource optimization study has been carried out using the metric learning MLP for graph construction, derived from the Exa.TrkX [27] pipeline. This study follows the approach of using quantization, i. e. replacing floating-point numbers with fixed-point numbers of arbitrary bit-width, as used in [22], in addition with pruning, i. e. removing unnecessary weights from the network, as demonstrated in [28]. The model is implemented with PyTorch and features four 512 dimensional hidden layers. Each hidden layer consists of a linear layer, a batch normalization layer, and a ReLU activation function. A fifth linear layer is providing the final 12-dimensional output.

Quantization Aware Training (QAT) has been introduced into the model by replacing the linear with QuantLinear layers and the ReLU activation function with QuantReLU from the Brevitas [29] package. A heterogeneous quantization approach has been chosen, meaning that individual bit widths can be set for weights and activations in the first, last, and the internal layers. Weights in all linear and QuantLinear layers can be pruned during training using an iterative pruning procedure. The L1 loss has been included in the training loss, since it has been found to improve performance for the model being pruned iteratively. Unstructured pruning is applied after 180 epochs, or after 10 epochs if the validation loss is stable. A fine tuning approach has been chosen, meaning that neither weights nor learning rate have been rewound after each pruning step. Model size has been evaluated by computing Bit Operations (BOPs) [30], as described in [28], using the QONNX [31, 32] package. BOPs are sensitive to the bit widths of both weights (b_w) and activations (b_a), and also the model sparsity due to pruning (pruning factor f_p).

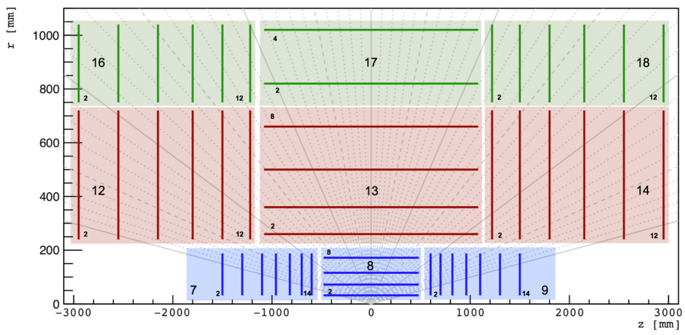

Figure 3. Schematic view of the TrackML detector geometry used for this resource optimization study [33]. It consists of a pixel detector (blue) close to the interaction point, complemented by strip detectors (green, red) on the outside.

A sample of 100 events of the TrackML [33] dataset ($t\bar{t}$ pair production, pile-up 200) has been used for this study. The data contains hit cluster position information as well as cluster energy and directional information derived from the cluster shape, as described in [34]. A hard p_T cut of 1 GeV has been applied to the input data, which reduces training time, but effectively also reduces the event size significantly. Because of this, and the fact that the TrackML detector geometry (figure 3) is simpler than the ATLAS ITk geometry, the results of this study are expected to be optimistic. The study is planned to be repeated with full ATLAS ITk simulation samples without applying any cut on the input data.

Model performance has been evaluated by computing the purity, defined as the number of true edges within the graph over the total number of edges within the graph, of the con-

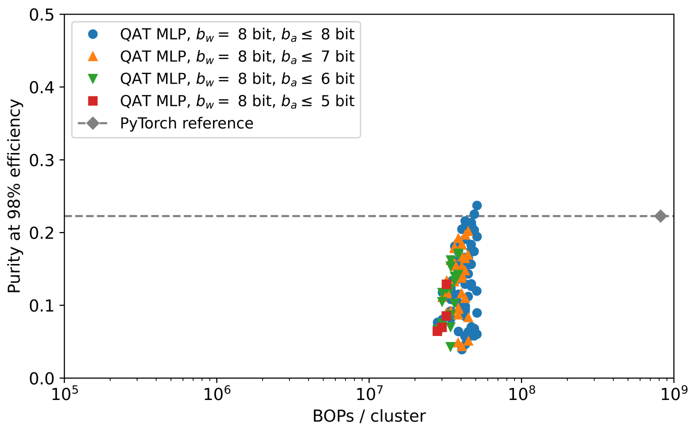

Figure 4. Purity at 98 % graph construction efficiency in the metric learning approach, versus model size given in Bit Operations (BOPs) per hit cluster to be processed. The results of a scan of the activations' bit widths b_a in the different layers are shown, with $b_a \leq N$ bit and N representing the largest bit width of all activations for a particular model. Weights have been fixed at a bit width $b_w = 8$ bit.

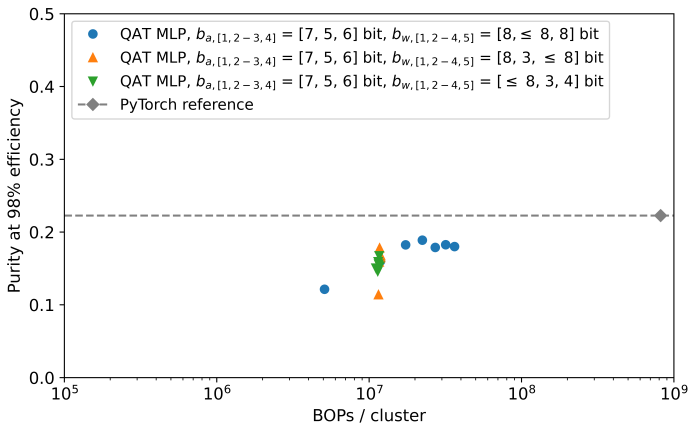

Figure 5. Purity at 98 % graph construction efficiency in the metric learning approach, versus model size given in Bit Operations (BOPs) per hit cluster to be processed. The results of a scan of the weights' bit widths b_w in the different layers are shown. Individual scans for the internal layers (blue dots), last layer (orange triangle upwards), and first layer (green triangle downwards) have been conducted. Activations have been fixed at a bit width $b_{a[1]} = 7$ bit in the first layer, $b_{a[2-3]} = 5$ bit in layers 2 and 3, and $b_{a[4]} = 6$ bit in the last layer.

structed graph at a fixed efficiency, defined as the number of true edges within the graph over the total number of true edges in the event, of $\varepsilon = 98\,\%$. This is achieved by varying the clustering radius in the edge construction step according to the model's performance. An optimization study has been carried out, optimizing activations and weight bit widths serially in independent steps. For the QAT models, the input data is rounded to a 13 bit fixed point representation (sign + 2 bit integer part + 10 bit fractional part).

Figure 4 shows the purity versus model size in BOPs for different models. In grey the reference value from the PyTorch model using 32 bit floating point precision is shown. The colored circles, triangles and squares correspond to a scan of activation bit widths b_a in the QAT model at a fixed bit width for all weights of $b_w = 8$ bit. It is found that the model is quite sensitive to the reduction of bits for activations. A scan of bit widths for the weights b_w has been performed by fixing the activation bit widths b_a, see figure 5. Due to the large hidden dimensions, largest resource savings are due to a reduction in bit widths in the internal layers. One can see that BOPs in this case can easily be reduced by a factor of about 100 by applying QAT without losing too much purity.

Three QAT models have then been chosen and pruned iteratively to further reduce model size, see figure 6. In all three cases, pruning factors f_p above 92 % have been reached before purity is reduced, which gives a total model size reduction factor of 1000 and more. For comparison, the PyTorch reference model has been pruned, and a similar pruning factor of 90 % is obtained.

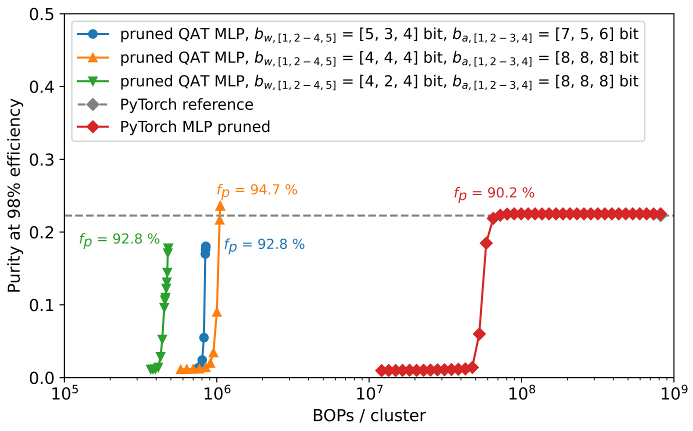

Figure 6. Purity at 98 % graph construction efficiency in the metric learning approach, versus model size given in Bit Operations (BOPs) per hit cluster that is being processed. Three QAT models have been chosen and pruned iteratively including L1 loss in the training loss. All three models achieved pruning factors f_p above 92 % without drops in performance. Also the PyTorch reference model has been pruned, dropping only at pruning factors larger than 90 % .

5 Future work

More work is going on towards the realization of tracking with graph neural networks on FPGAs for the ATLAS Event Filter system. While the study presented above is using TrackML data samples, it will be repeated and extended to the full pipeline using realistic ATLAS ITk simulation samples. In parallel, reduction of the size of the interaction network simply by reducing the number of parameters is also under investigation. A comparison between the two graph construction methods, metric learning and module map, is planned once both algorithms are available for hardware deployment. Segmentation of the graph into detector regions is being studied as a mitigation for potential excessive block memory utilization.

FPGA dataflow translation of the MLPs within the pipeline as well as of the interaction network itself have been started with the HLS4ML [35, 36] framework, and are also planned to be done with FINN [37, 38]. Finally, the goal is to have a fully running GNN based track reconstruction pipeline on an FPGA. Devices from both vendors, AMD and Intel, are under consideration for this task.

6 Conclusion

For online track reconstruction within the ATLAS Event Filter system at the HL-LHC, the application of graph neural networks on FPGAs is currently under investigation. The focus of the study presented here is set on the graph construction step. Model resource optimization studies and dedicated hardware implementations are underway. As demonstrated with the TrackML data sample, significant resource savings are expected to be achieved by exploiting quantization and pruning. These studies will be followed up by using realistic ATLAS ITk simulation samples, and by investigating and optimizing the interaction network.

7 Acknowledgements

S. D. acknowledges support by the Federal Ministry of Research and Education (BMBF).

References

[1] ATLAS TDAQ Collaboration, *The ATLAS Trigger/DAQ Authorlist*, version 14, ATL-COM-DAQ-2022-127, CERN, Geneva, 2022, `https://cds.cern.ch/record/2842310`

[2] ATLAS Collaboration, Journal of Instrumentation **3**, S08003 (2008)

[3] I. Zurbano Fernandez et al., **CERN-2020-010** (2020)

[4] ATLAS Collaboration, **CERN-LHCC-2015-020** (2015)

[5] ATLAS Collaboration, **CERN-LHCC-2017-021** (2017)

[6] ATLAS Collaboration, **CERN-LHCC-2017-005** (2017)

[7] ATLAS Collaboration, **CERN-LHCC-2020-007** (2020)

[8] ATLAS Collaboration, **CERN-LHCC-2017-020** (2017)

[9] ATLAS Collaboration, **CERN-LHCC-2022-004** (2022)

[10] E.A. Moreno et al., Eur. Phys. J. C **80**, 58 (2020)

[11] S.R. Qasim et al., Eur. Phys. J. C **79**, 608 (2019), `1902.07987`

[12] S. Farrell et al., *Novel deep learning methods for track reconstruction*, in *4th International Workshop Connecting The Dots 2018* (2018), `1810.06111`

[13] 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`

[14] G. DeZoort et al., Comput. Softw. Big Sci. **5**, 26 (2021), `2103.16701`

[15] X. Ju et al., Eur. Phys. J. C **81**, 876 (2021), `2103.06995`

[16] C. Biscarat et al., EPJ Web Conf. **251**, 03047 (2021), `2103.00916`

[17] S. Thais et al., *Graph neural networks in particle physics: Implementations, innovations, and challenges* (2022), `2203.12852`

[18] R. Liu et al., *Hierarchical Graph Neural Networks for Particle Track Reconstruction*, in *21th International Workshop on Advanced Computing and Analysis Techniques in Physics Research: AI meets Reality* (2023), `2303.01640`

[19] S. Caillou et al. (ATLAS), *ATLAS ITk Track Reconstruction with a GNN-based pipeline* (2022), `https://cds.cern.ch/record/2815578`

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

[21] P.W. Battaglia et al. (2016), `1612.00222`

[22] A. Elabd et al., Front. Big Data **5**, 828666 (2022), `2112.02048`

[23] Z. Que et al. (2022), `2209.14065`

[24] S.Y. Huang et al. (2023), `2306.11330`

[25] AMD, *Vitis unified software platform documentation: Embedded software development (ug1400)* (2023), `https://docs.xilinx.com/r/en-US/ug1400-vitis-embedded`

[26] Intel, *Intel® oneAPI Toolkits* (2023), `https://www.intel.com/content/www/us/en/developer/tools/oneapi/toolkits.html`

[27] D. Murnane et al., *Tracking-ml-exa.trkx*, `https://github.com/HSF-reco-and-software-triggers/Tracking-ML-Exa.TrkX`

[28] B. Hawks et al., Frontiers in Artificial Intelligence **4** (2021)

[29] A. Pappalardo, *Xilinx/brevitas* (2023), `https://doi.org/10.5281/zenodo.3333552`

[30] C. Baskin et al., ACM Trans. Comput. Syst. **37** (2021)

[31] A. Pappalardo et al., *QONNX: Representing Arbitrary-Precision Quantized Neural Networks*, in *4th Workshop on Accelerated Machine Learning (AccML) at HiPEAC 2022 Conference* (2022), `2206.07527`

[32] Y. Umuroglu et al., *fastmachinelearning/qonnx* (2022), `https://github.com/fastmachinelearning/qonnx`

[33] M. Kiehn et al., EPJ Web Conf. **214**, 06037 (2019)

[34] P.J. Fox et al., JINST **16**, P05001 (2021), `2012.04533`

[35] J. Duarte et al., JINST **13**, P07027 (2018), `1804.06913`

[36] FastML Team, *fastmachinelearning/hls4ml* (2023), `https://github.com/fastmachinelearning/hls4ml`

[37] M. Blott et al., ACM Transactions on Reconfigurable Technology and Systems (TRETS) **11**, 1 (2018)

[38] Y. Umuroglu et al., *FINN: A Framework for Fast, Scalable Binarized Neural Network Inference*, in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays* (ACM, 2017), FPGA ’17, pp. 65–74