# Online track reconstruction with graph neural networks on FPGAs for the ATLAS experiment

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

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

**Abstract.** For the HL-LHC upgrade of the ATLAS TDAQ system, a heterogeneous computing farm deploying GPUs and/or FPGAs is considered to be used for the Event Filter system, together with the use of modern machine learning algorithms such as Graph Neural Networks (GNNs) to solve computationally complex tasks within that system. In this study, the development of a GNN based track finding pipeline on FPGAs for the ATLAS Inner Tracker is presented as part of the Event Filter system. Each step in the GNN-based tracking pipeline is explored: graph construction, edge classification using an interaction network, and segmentation of the graph into track candidates. Optimizations of the GNN approach are investigated to minimize FPGA resource utilization and maximize throughput while maintaining high track reconstruction efficiency and low fake rates required for the ATLAS Event Filter tracking system. These optimizations include model hyperparameter tuning, model pruning, quantization-aware training, and sequential processing of sub-graphs across the detector.

ATL-DAQ-PROC-2025-003
04 February 2025

## 1 Introduction

For the operation at the High-Luminosity LHC [1], the ATLAS experiment will undergo a major upgrade [2]. This upgrade includes the installation of a new all-silicon tracking detector, the Inner Tracker (ITk) [3, 4], with extended forward coverage up to $|\eta| = 4$, see Figure 1, as well as a drastic re-design of the trigger and data acquisition (TDAQ) system [5]. Within the TDAQ system, track reconstruction for the ITk will be carried out in the Event Filter (EF) for those events that are accepted by the hardware-based Level-0 trigger at a rate of 1 MHz within regions of interest and 150 kHz for the full ITk.

For the EF system, the installation of a heterogeneous computing farm including GPUs and/or FPGAs is under consideration because of their potential to significantly reduce the EF's power needs compared to a CPU-only server farm [7]. Since track reconstruction for the ITk requires the largest fraction of the available online computing resources, studies on its acceleration using hardware accelerators are being carried out, alongside the exploration of using machine learning (ML) methods for this task [7]. One of the approaches currently under study is the implementation of a track finding pipeline based on Graph Neural Networks (GNNs) on FPGAs [8].

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

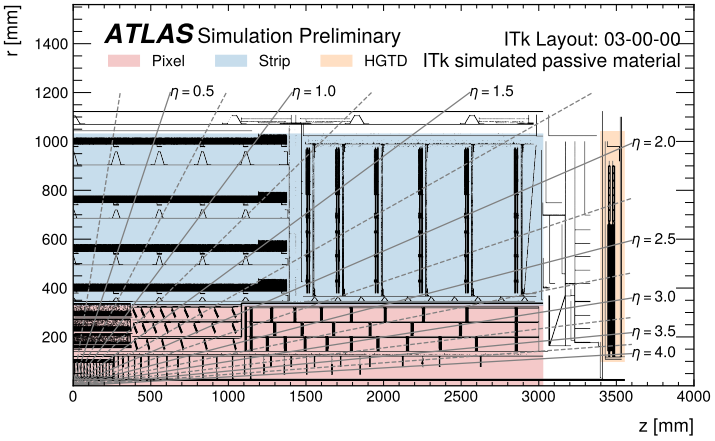


**Figure 1.** Layout of the ATLAS Inner Tracker [6], showing the locations of the pixel and strip detectors, alongside the High-Granularity Timing Detector (HGTD).

## 2 Tracking with Graph Neural Networks on FPGAs

The envisioned FPGA tracking pipeline consists of several steps that lead from raw pixel and strip hit data to tracks. First, pixel and strip hits are clustered, their local coordinates are transformed into global coordinates, and space-points are created from pairs of overlapping clusters within the two strip sensors per layer. Then, a pattern recognition algorithm is used to find track candidates. Multiple options for this task are under study, such as a GNN or a Hough Transform. Track candidates then get scored and their track parameters estimated, so that fake and duplicate tracks can be removed. For any pattern recognition algorithm that only uses a subset of the detector layers, like the currently studied implementation of a Hough Transform, the tracks are passed to an extrapolator algorithm to add hits from the remaining layers. Lastly, the tracks are sent back to the host CPU for a precision fit using the Kalman Filter.

Track finding with a GNN comprises three distinct sub-steps [9]. First, a graph is constructed from pixel and strip space-points acting as nodes and edges suggesting that two space-points in consecutive layers originate from the same particle. A GNN is used to classify these edges as true or false, and all edges classified as false are being removed from the graph. The remaining graph is then disconnected into individual track candidates.

This work relies on the ACORN [10] track finding pipeline, which includes two methods for graph construction: module map and metric learning. The module map is derived from simulation samples and contains most of the possible true connections between ITk detector modules, i. e. consecutive detector modules that are likely to be hit by a particle of interest. Doublets are created for all space-points of the corresponding module pairs. This is followed by geometrical cuts acting on space-point doublet and triplet level [11]. Metric learning is a machine learning approach, where space-points are embedded through a Multi-Layer Perceptron (MLP) into a higher dimensional space, which is followed by a radius clustering algorithm to create edges between space-points that are close in said embedded space. Metric learning is typically followed by a filter network to reduce graph sizes to similar orders of magnitude as the module map delivers after applying triplet cuts. The edge classifying GNN is built from encoding MLPs acting on node and edge features, several message passing

steps in the form of interaction network layers [12], and a decoding MLP transforming the resulting edge features into classification scores. A cut on this edge score allows to prune false connections from the graph. For disconnecting the pruned graph into track candidates, a connected component algorithm is used. Since these graph components may still contain more than one hit per layer, a second algorithm, referred to as walkthrough, can be used to split these components into individual track candidates [11].

For the implementation of GNN-based track finding within the FPGA tracking pipeline, the ML models have to be compressed to make efficient use of the FPGA fabric. The current strategy focuses on using quantization and pruning to reduce model size. Since the current demonstrators for the EF Tracking project are developed for AMD data center class FPGAs like the Alveo U250, the AMD Vitis [13] tool flow is used to implement the algorithms as kernels. In the following sections, the current development status of GNN track finding on FPGAs for ATLAS is described. Section 3 summarizes the FPGA deployment of metric learning, section 4 describes a compression study of the interaction network, section 5 presents the development of a connected components kernel, and section 6 discusses detector regionalization as an overall strategy to reduce graph sizes.

## 3 Graph Construction with Metric Learning on FPGA

In this study, the MLP used for embedding space-points into a latent space consists of 4 hidden layers, each made of a fully connected linear layer with 512 output dimensions, a batch normalization layer and ReLU activation function. Cylindrical coordinates of ITk space-points $(r, \phi, z)$ are used as input features. A fully connected linear layer with 12 output dimensions forms the output layer. The MLP is trained on 800 $t\bar{t}$-events overlaid with pileup $\langle\mu\rangle = 200$ derived from full detector simulation. The model is trained to embed pairs of hits of target particle tracks close-by, and to push false hit pairs away from each other, by using a hinge loss function [11]. Target particles are defined as any primary charged particle, except electrons, with $p_T > 1$ GeV, that leave at least 3 space-points in the ITk.

The model and its training are implemented in the ACORN [10] framework using PyTorch [14]. To compress the model for FPGA deployment, the same strategy as described in [8] is used. Quantization aware training is implemented using the Brevitas [15] package, and iterative magnitude based pruning with learning rate rewinding is added into the training process. To facilitate pruning, an L1 regularization term is added to the training loss. Through these measures, the number of non-zero weights could be reduced from 800 k to less than 30 k, and the bit precision of weights and activations could be reduced to 6 bit for the input and output layers, and 4 bit for the hidden layers. Biases are fixed at 8 bit precision, and the input data is quantized to 12 bit fixed-point representations.

**Table 1.** Resource utilization of the Metric Learning MLP compiled with FINN for an Alveo U280.

| Resource | Utilization (% of U280) | |
|---|---|---|
| | version 1 | version 2 |
| LUT | 25 k (1.9 %) | 352 k (27 %) |
| FF | 63 k (2.4 %) | 107 k (4.1 %) |
| BRAM | 166 (8.2 %) | 0 (0 %) |
| DSP | 1547 (17 %) | 133 (1.5 %) |
| $f_{max}$ | > 300 MHz | 201 MHz |
| Throughput | 385 kHz | 20 MHz |

The model is exported to the QONNX [16] format, and compiled using FINN [17] into Vitis HLS code for FPGA deployment to test as standalone accelerator on an AMD Alveo U280 card. A first build of the accelerator at a target throughput of 300 kHz space-point embeddings, operated at a clock speed of 300 MHz, achieved a measured throughput of 385 kHz using little logic resources, but some more BRAM and DSPs, see version 1 in Table 1.

**Table 2.** Preliminary resource utilization of the radius clustering kernel implemented with Vitis HLS.

| Resource | Utilization (% of U280) | |
|---|---|---|
| LUT | 27 k | (2.1 %) |
| FF | 35 k | (1.3 %) |
| BRAM | 871 | (43 %) |
| DSP | 140 | (1.6 %) |

The radius clustering algorithm is implemented as a standalone kernel using Vitis HLS. The algorithm sorts the embedded hits according to their first 3 dimensions into separate buffers, which are one radius wide in each of the 3 dimensions, to reduce the number of hit comparisons that have to be done in the following step. Between all the hits within a buffer and within neighboring buffers, their full 12-dimensional distances are computed. An edge is created between two hits if their distance is smaller than the clustering radius, unless the node has already reached its maximum number of edges. A preliminary version of this algorithm is used for functional verification. It is not yet optimized for throughput or resource usage. It utilizes little logic resources and DSPs, but relies heavily on BRAM, see Table 2. This is because the algorithm stores two buffers in BRAM for processing, and each buffer is sized for up to 30 k embedded hits, since the output of the MLP is not yet optimized to form small buffers.

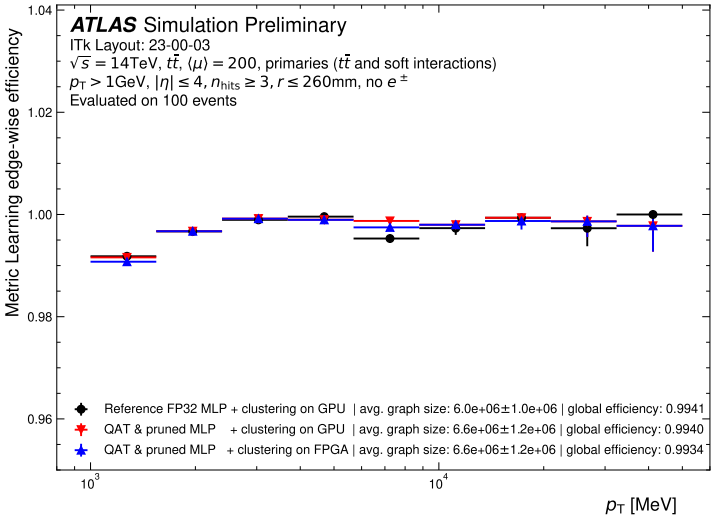


**Figure 2.** Comparison of the metric learning edge-wise efficiency of a PyTorch reference model using 32 bit floating point precision executed on GPU, a quantized and pruned model executed on GPU, and the same quantized and pruned model together with the radius clustering kernel running on the FPGA [18].

The performance of the metric learning based graph construction on FPGA is compared to the original PyTorch GPU implementation by evaluating the edge-wise graph construction efficiency on 100 $t\bar{t}$-events with pileup $\langle\mu\rangle = 200$, see Figure 2. Using the quantized and pruned MLP on GPU, an efficiency close to the one of the 32 bit floating point reference model is achieved, while creating graphs that are on average 10 % larger. Running the compressed MLP and the radius clustering kernel on the FPGA, graphs of the same size are constructed, while a global efficiency drop by less than 0.1 % is observed. The slight performance degradation is attributed to numerical inaccuracies caused by the QONNX export.

Towards a final implementation, several optimizations to reduce resource utilization and enhance throughput are considered. By training the metric learning MLP to produce wide, uniform and uncorrelated output feature distributions for the 3 dimensions used for buffer sorting, the BRAM utilization of the radius clustering kernel is expected to decrease by more than a factor of 10, and the number of needed computations will be reduced. By fully unrolling the metric learning MLP on the FPGA, throughput matching the clock speed can be achieved. This requires an even stronger degree of sparsity as used in the above model, so that the network can fit on the device. A throughput optimized version, targeting a clock speed of 300 MHz and an equivalent throughput of space-point embeddings, is implemented from a model containing less than 15 k non-zero weights for the same architecture. This implementation consumes less DSPs, however, it utilizes more logic resources, see version 2 in Table 1. At its maximum frequency of $f_{\text{max}} = 201$ MHz, the measured throughput of this implementation reaches 20 MHz of space-point embeddings. The discrepancy between clock speed and throughput is attributed to undersized FIFOs connecting the individual layers of the model. Adding larger FIFOs is expected to moderately increase the BRAM utilization.

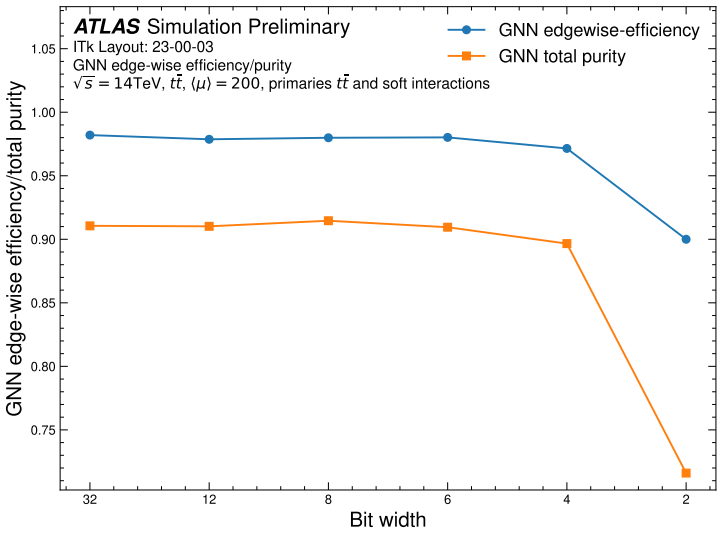


**Figure 3.** Edge-wise efficiency and purity of the GNN versus bit width used for weights and activations used within the network [18].

## 4 Compression of the Graph Neural Network

As a step towards deployment of the GNN on an FPGA, a model compression study is conducted using again the same strategy of applying quantization aware training and magnitude based iterative pruning [8]. For this study, a GNN is used which consists of node and edge encoder networks, 8 message passing steps including node and edge update networks operating on 48 hidden dimensions, and a final edge decoding network. The GNN edge-wise efficiency and purity are evaluated versus the bit width used for weights and activations, see Figure 3. It is found that down to 4 bits for weights and activations, a performance comparable to the 32 bit floating point model can be achieved. For a fixed bit width of 6 bit for weights and activations, a training with iterative magnitude based pruning is conducted. It is observed that more than 90 % of weights can be removed without significantly affecting model performance. These findings show great potential for reducing the model footprint when targeting model deployment on an FPGA.

## 5 Graph Segmentation with Connected Components on FPGA

The interaction network provides a score per edge, which is used to prune the graph by removing fake edges. A connected component algorithm [19] is used to segment the pruned graph into individual track candidates. The remaining graph components may still contain nodes with multiple edges which branch off into multiple directions. Track scoring algorithms, developed as part of the full FPGA tracking pipeline [7], are expected to recover the good individual track candidates out of these components. Hence, the focus is set on the development of a graph segmentation kernel to create the connected components.

The graph segmentation kernel is implemented with Vitis HLS. It implements the graph pruning step by applying a programmable threshold. The surviving edges are written into a more memory efficient graph table, where nodes without any remaining edges are completely dropped. From this graph structure, the nodes forming the connected components are grouped together by following the edges. A preliminary implementation shows that the algorithm can be implemented with little logic resource utilization, see Table 3. BRAM usage is expected to be decreased with an optimized design. A first timing study indicates a processing time below 200 ms for a full ITk event. This processing time is currently dominated by the graph pruning step, and is expected to decrease with an implementation optimized for throughput.

**Table 3.** Preliminary resource utilization of the graph segmentation implemented with Vitis HLS.

| Resource | Utilization (% of U280) | |
|---|---|---|
| LUT | 13 k | (1.0 %) |
| FF | 17 k | (0.7 %) |
| BRAM | 181 | (9.0 %) |
| DSP | 0 | (0 %) |

## 6 Detector Regionalization

Considering the application of GNN track finding for events of the full ITk detector, not only the models are large and require compression, but also the graphs are large, with hundreds of thousands of nodes and a few million edges. Storing these large graph structures can only be realized by utilizing the external DDR memory or HBM of the accelerator cards. Since in most cases only a small region of the event has to be processed, see section 1, detector

regionalization is also considered as a strategy for full detector events to remove the need for extensive external memory access. In this scenario, a full detector event is split into smaller detector regions that can be processed independently. Each region covers a detector volume of $\eta \times \phi = 0.2 \times 0.2$ with significant overlap between regions. Having one graph per region, this approach results in more than 1000 graphs for a full event. The individual graphs are on average a factor 100 smaller compared to full detector graphs.

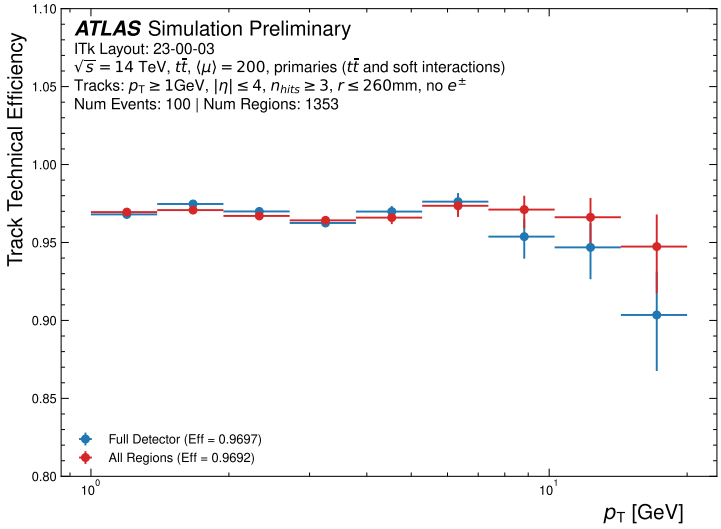


**Figure 4.** Comparison of the track finding efficiency for a GNN trained on full detector events between its inference on full detector graphs and regionalized detector graphs [18].

To validate that this approach works with the GNN based track finding, regionialized graphs have been constructed with the module map for $t\bar{t}$-events with $\langle\mu\rangle = 200$. A GNN that has been trained on full detector graphs is used for inference with these smaller regional graphs. The track finding performance is evaluated by measuring the efficiency to find reconstructible, primary non-electron tracks with $p_{\mathrm{T}} > 1$ GeV by applying the GNN and graph segmentation. An overall efficiency comparable to inference on full detector graphs is achieved for the regional tracking case, as can be seen in Figure 4. Due to the large overlaps, the regionalization approach produces many more duplicate tracks compared to inference on full detector graphs. A duplicate removal algorithm, part of the overall FPGA tracking pipeline, is expected to filter these duplicates before sending the track candidates back to the host.

## 7 Summary and Outlook

For the HL-LHC upgrade of the ATLAS Event Filter system, a heterogeneous online computing farm is considered to be deployed. As part of a full tracking pipeline on FPGAs, the application of GNN-based track finding is under study. Preliminary resource utilizations of components of this track finding approach, such as graph construction with metric learning, and graph segmentation with connected components, are well within the budget of the targeted FPGAs. Functional verification of metric learning is achieved by running the MLP and the radius clustering kernel on FPGA. Using quantization aware training and pruning shows to be a viable approach to compress the interaction network, as well. Detector regionalization

has been validated as an approach to reduce graph sizes, potentially avoiding the use of external memory for the graph structures. Furthermore, the study shows that regional tracking and full detector tracking can be done with the same GNN. Future work will focus on the translation of the interaction network for FPGAs, and on the integration of the GNN-based track finding into the full FPGA tracking pipeline.

## 8 Acknowledgements

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

## References

[1] I. Zurbano Fernandez et al., **CERN-2020-010** (2020). 10.23731/CYRM-2020-0010
[2] ATLAS Collaboration, **CERN-LHCC-2015-020** (2015).
[3] ATLAS Collaboration, **CERN-LHCC-2017-021** (2017).
[4] ATLAS Collaboration, **CERN-LHCC-2017-005** (2017).
[5] ATLAS Collaboration, **CERN-LHCC-2017-020** (2017).
[6] ATLAS Collaboration, ATLAS Inner Tracker Layout 03-00-00 (2023), https://atlas.web.cern.ch/Atlas/GROUPS/PHYSICS/PLOTS/ITK-2023-001/
[7] ATLAS Collaboration, **CERN-LHCC-2022-004** (2022).
[8] S. Dittmeier (ATLAS TDAQ), Track reconstruction for the ATLAS Phase-II Event Filter using GNNs on FPGAs, EPJ Web Conf. **295**, 02032 (2024). 10.1051/epjconf/202429502032
[9] S. Caillou et al. (ATLAS), ATLAS ITk Track Reconstruction with a GNN-based pipeline (2022), `https://cds.cern.ch/record/2815578`
[10] M.J. Atkinson et al., ACORN, https://github.com/GNN4ITkTeam/CommonFramework
[11] ATLAS Collaboration, Computational Performance of the ATLAS ITk GNN Track Reconstruction Pipeline, **ATL-PHYS-PUB-2024-018** (2024).
[12] P.W. Battaglia et al., Interaction Networks for Learning about Objects, Relations and Physics (2016), `1612.00222`.
[13] AMD, Vitis unified software platform documentation: Embedded software development (ug1400) (2023), https://docs.xilinx.com/r/en-US/ug1400-vitis-embedded
[14] A. Paszke et al., Pytorch: An imperative style, high-performance deep learning library, Advances in neural information processing systems **32** (2019).
[15] A. Pappalardo, Xilinx/brevitas (2023), `https://doi.org/10.5281/zenodo.3333552`
[16] A. Pappalardo et al., QONNX: Representing Arbitrary-Precision Quantized Neural Networks, in *4th AccML at HiPEAC 2022 Conference* (2022), `2206.07527`
[17] M. Blott et al., Finn-r: An end-to-end deep-learning framework for fast exploration of quantized neural networks, ACM TRETS **11**, 1 (2018).
[18] ATLAS Collaboration, Approved plots for the EF Tracking project (2024), https://twiki.cern.ch/twiki/bin/view/AtlasPublic/EFTrackingPublicResults
[19] L. He et al., The connected-component labeling problem: A review of state-of-the-art algorithms, Pattern Recognition **70**, 25 (2017). https://doi.org/10.1016/j.patcog.2017.04.018