Proceedings of the CTD 2023
PROC-CTD2023-XX
November 17, 2023

Track reconstruction for the ATLAS Phase-II High-Level Trigger using Graph Neural Networks on FPGA

Sachin Gupta

On behalf of the ATLAS TDAQ collaboration [1],
Physikalisches Institut
Heidelberg University, Germany

ABSTRACT

The High-Luminosity LHC (HL-LHC) will provide an order of magnitude increase in integrated luminosity and the amount of data produced per event. Foreseeing this data pile-up requires upgrades in 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 EF tracking, a heterogeneous computing farm consisting of CPUs and potentially GPUs and FPGAs is under study together with modern machine learning algorithms. Graph Neural Networks (GNN) are well suited for particle tracking and can provide fast inference on FPGAs. Optimizing the GNN-based pipeline specifically for FPGAs aims to reduce resources while retaining high track reconstruction efficiency and low fake rates required for the ATLAS Phase-II EF tracking system. Resources required for the machine learning step can be reduced using model compression techniques such as Qauntization Aware Training (QAT) and Pruning. These methods are applied in the first stage of the GNN-based pipeline and the first results are obtained for the TrackML dataset.

PRESENTED AT

Connecting the Dots Workshop (CTD 2023)
October 10-13, 2023

1 Introduction

The Large Hadron Collider (LHC) is set to reach unprecedented luminosity for the High-Luminosity LHC [2] phase. The average number of proton-proton collisions per bunch crossing will be increased by a factor of ten (Pileup $\langle\mu\rangle = 200$). Properly accounting for and mitigating the effects of pileup is crucial for distinguishing genuine signals of new physics from background events. Thus, the experiments at the LHC need to adapt to these new conditions, which come at much higher data rates. When it comes to the ATLAS experiment [3], upgrades will involve changing the detectors and the Trigger and Data Acquisition system (TDAQ). The Inner Detector will be replaced by a new all silicon detector, the Inner Tracker (ITk) [4]. The Phase II TDAQ system [5] will have a hardware-based Level-0 trigger and an Event Filter (EF) capable of performing online track reconstruction for the region of interest (pixel and strip detectors of ITk [4]) at 1 MHz and for full tracking at 150 kHz request rate. For the EF system, a heterogeneous computing farm consisting of CPUs and potentially GPUs and/or FPGAs is under study.

Tracking for the Inner Tracker (ITk) becomes challenging when particle density is high. Recently, graph neural networks have emerged as potential algorithms for particle tracking [7]. The detector hit information naturally represents the data structure of a graph, with hits corresponding to graph nodes and the connections between two hits corresponding to graph edges. The algorithm can be accelerated with parallelizable computing architectures such as GPUs and FPGAs. Keeping resource constraints in mind it becomes imperative to reduce the model size while keeping the model performance intact. The report summarises the effect of two model compression techniques: Quantization Aware Training (QAT) and Model Pruning on the first stage of the GNN-based pipeline [7].

2 Track Reconstruction using Graph Neural Networks

Particle tracking is a crucial and intricate task in any particle physics experiment. The complexity and computational cost are both expected to increase for HL-LHC. Graph neural networks are a non-conventional way of constructing particle tracks by modeling the dependencies among the particle hits. The application of a GNN for particle tracking was first introduced in Ref. [8]. Following this, the GNN-based pipeline has been applied for the TrackML dataset [6] and ATLAS ITk [9]. The algorithm has shown high track reconstruction efficiency and a low fake rate. The pipeline generates track candidates in three stages as shown in Figure 1.

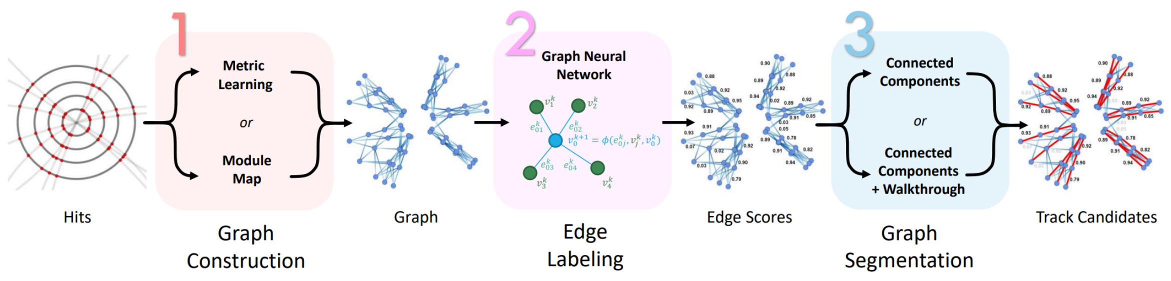


Figure 1: Schematic diagram of GNN based pipeline [9].

Graph Construction: This is the first step that generates the initial combinatorics also called *graphs*. This can be done in two ways: *metric learning* and *module map*. *Metric learning* is a machine learning-based method that implements a Multi Layer Perceptron (MLP) to generate an initial graph. Hits are mapped to some latent space with the MLP. In the latent space, for every hit, a circle of radius r is drawn and the edges are constructed among all the hits lying within the circle. The network learns to put all the hits belonging to the same particle within the circle as shown in Figure 2. The goal of this step is to keep as many true edges as possible without producing too many fake edges. *Module map* constructs the graph by connecting hits from the different modules of the detector and then applying the geometric cut values. The modules are linked with the help of simulated data.

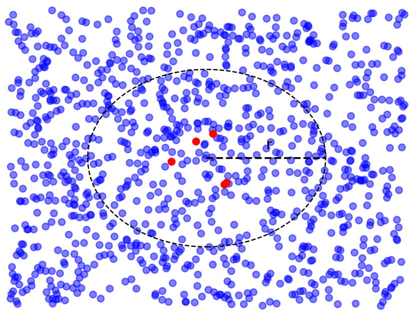


Figure 2: Toy example of the latent space. Red-colored hits belong to the same particle.

Edge Labeling: Once the graph is generated, the GNN assigns an edge score to each edge. The value of the edge score is between 0 and 1. The GNN is trained such that an edge belonging to the actual particle track is assigned a score value closer to 1. Physics-motivated Interaction Network (IN) GNN [10] is trained for the classification of edges.

Graph segmentation: After edge labeling, a cut value on the edge score is applied and final edges are selected. Graph segmentation is a task-specific post processing [11]. For track formation, it gives “track candidates” that specifically denote a discrete and simply connected line within the graph. These track candidates can be further supplied to a track fitting algorithm.

3 Performance Metrics for Graph Construction

We applied the aforementioned model compression techniques to the first stage of the pipeline i.e. *graph construction* with metric learning. Before discussing their effect on the performance of MLP it is important to give the performance metric for this stage. As mentioned in Section 2, the goal of the graph construction is to keep as many true edges without producing too many fake edges. The performance is quantified by two metrics: *Efficiency* and *Purity*

$$\text{Efficiency} = \frac{\text{True edges in the graph}}{\text{Total true edges}} \tag{1}$$

$$\text{purity} = \frac{\text{True edges in the graph}}{\text{Total edges in the graph}} \tag{2}$$

The graph’s efficiency assesses its ability to encompass all genuine edges through the initial combinatorial process. In contrast, purity gauges the proportion of generated edges that are indeed true, with purity being directly correlated to the graph’s size. Constructing the graph by including all conceivable edges results in 100 percent efficiency, but purity tends to approach zero in such cases. Consequently, as purity decreases, the graph size tends to increase. Thus our aim is to keep the model as pure as possible while fixing its efficiency at 98 %.

4 Resource Estimation

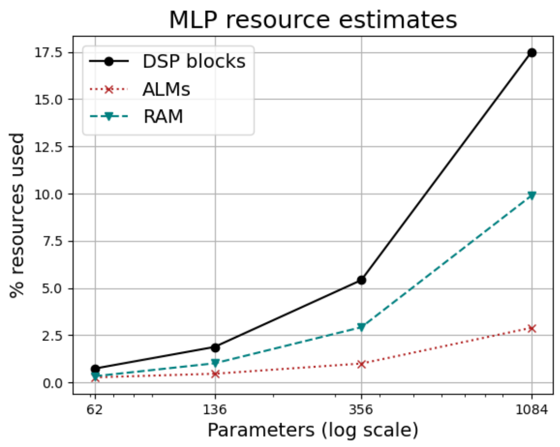
The effectiveness of GNN based tracking has been well-established in previous studies [7, 9, 11]. For EF tracking, its implementation on FPGAs is explored. FPGAs can simultaneously provide parallelization with low energy consumption [12]. For FPGA deployment, the algorithm must be converted to a Hardware Description Language (HDL) [13]. However, technologies like High Level Synthesis (HLS) can also be used which generates code similar to C languages. Integrated Development Environments (IDEs) like Intel Quartus [14], showcase resource estimates generated by the underlying High-Level Synthesis (HLS) compiler.

The machine learning parts (graph construction and edge labelling) are converted to HLS with the open-source Python library HLS4ML ("High-Level Synthesis for Machine Learning") [15]. Non-machine learning parts i.e. graph segmentation, can be implemented with HLS, or directly in an HDL. The report only discusses the machine learning-based part as they are expected to predominantly occupy FPGA resources.

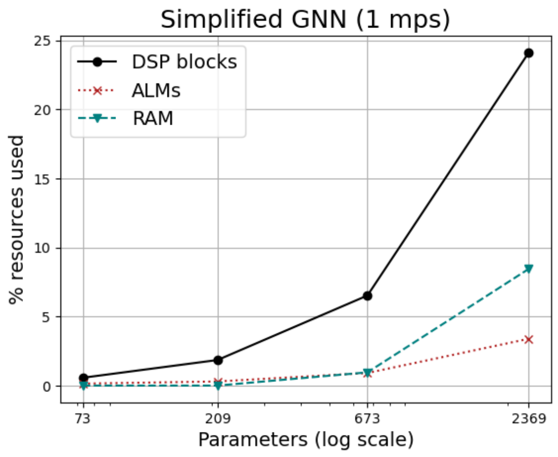
Resource	**DSP blocks**	**ALMs**	**M20Ks**
Availability	5,760	933,120	11,721

Table 1: Resource specifications of the Intel Stratix 10 GX 2800 FPGA [18]

The machine learning part of the algorithm was trained using the Python library PyTorch [16]. The PyTorch model was first converted to ONNX [17] before HLS4ML generated the HLS code which was then compiled using Intel Quartus, and resources were estimated for an Intel Stratix 10 GX 2800 [18]. The resources available on the target device are specified in Table 1. Digital Signal Processing blocks (DSP blocks) perform operations such as matrix multiplications. Adaptive Logic Modules (ALMs) are logic elements typically consisting of Look-Up Tables (LUTs) and Flip Flops (FFs), which are used to implement and store logic operations. Random Access Memory (RAM) is used for storage and retrieval of larger amounts of data. On Intel FPGAs, this is referred to as "M20K" [19].



(a)



(b)

Figure 3: Resource usage of the metric learning (a) and Simplified GNN (single message passing step) (b) for different number of parameters [19].

Resources were computed by varying the number of hidden dimensions for the MLPs in stages one and two [19]. A simplified GNN model was considered because some of the functions used in the PyTorch model, i.e. the aggregation at nodes and the index matrix, were not yet supported by HLS4ML. For simplified GNN resource estimation is shown for one message passing step (Figure 3.b). In both cases, Digital Signal Processing (DSP) blocks are predominantly used due to the presence of matrix multiplications in neural networks.

5 Model Compression Techniques

As seen above, an FPGA implementation of the offline-like GNN-based pipeline would be predominantly occupying the FPGAs DSP blocks. For the ITk, the number of parameters is expected to be in the order of a few hundred thousand. The challenge is to decrease resource usage while maintaining performance (efficiency and purity). Several compression techniques for machine learning models have been developed [20]. Below, we outline the techniques that we have applied to the pipeline.

5.1 Quantization Aware Training

QAT [22] allows the training of a neural network for representation different from Floating-Point32 [21] while keeping the performance intact. The bit width of the neural network parameters: weights, activation, and bias, are defined first and the training of the network is done afterwards. Hence, not only does the network learn the value of the parameters, but also learns their representation for different bit representations. QAT implementation can be achieved by the Python library: **Brevitas** [23] which can be integrated with PyTorch as shown in Figure 4

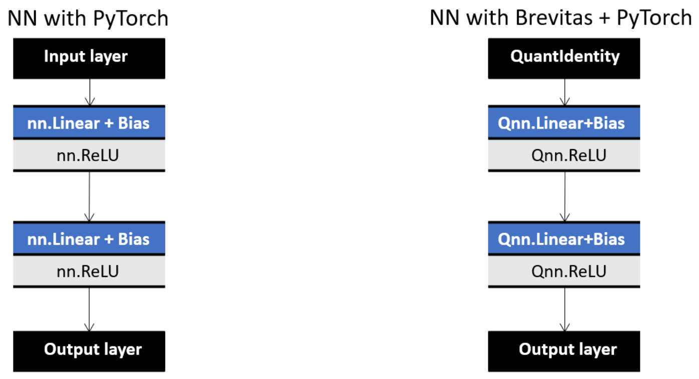


Figure 4: Schematic diagram of Neural network with Pytorch and Pytorch+Brevitas. Linear layers (nn.Linear) are replaced by Quant Linear (QNN.Linear) and usual ReLU activation (nn.ReLU) is replaced by Qnn.ReLU. Input data can also be quantized by QuantIdentity.

5.2 Model Pruning

The second method we investigated for resource reduction is model pruning, founded on the principle that sparse matrices can significantly enhance computational speed compared to regular matrices. It is the technique of discarding the weights that do not contribute to the model's performance. The magnitude of weights is penalized by L1-Loss, and the smallest weights are removed after certain epochs.

6 Results

QAT and model pruning were applied to the MLP in the graph construction stage that was trained and tested with the TrackML dataset [6]. A heterogeneous structure of bit widths was used for weights and activations as shown in Figure 5. No biases were involved after the linear layers. For FPGA-specific requirements, Batchnorm was used. Model pruning was done with the frequency of 180 epochs with 10 % weight pruning.

Each parameter's bit width is characterized by three variables i.e. for weights we have $b_{w[1,2-4,5]} = [B_I, B_H, B_O]$ and for activation, we have $b_{a[1,2-3,4]} = [A_I, A_H, A_O]$. Various configurations are used to evaluate the model performance i.e. purity at 98 % efficiency with respect to model size. The efficiency is

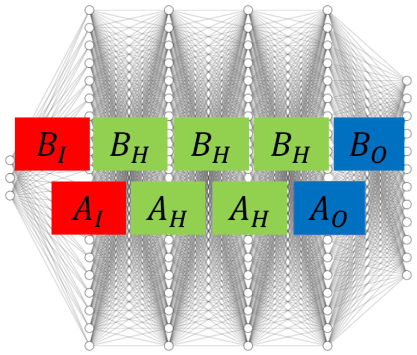


Figure 5: MLP with input dimension 3 and output dimension 12. Each hidden layer has 512 nodes. MLP has five regions with different bit widths for parameters shown by three different colors. $[B_I, B_H, B_o]$ and $[A_I, A_H, A_o]$ are the bit width variables for weights and activations respectively.

fixed by varying the clustering radius in the graph construction stage as mentioned in Section 2. The model size is quantified by calculating Bit Operations (BOPs) [24] using the QONNX [25] package. The Floating-Point32 model is used as a reference model. The effect of iterative pruning is clearly seen in the reduction of BOPs and purity. The performance is retained by order of three reductions in BOPs by applying QAT and pruning together. More details of this study can be found in Ref. [26].

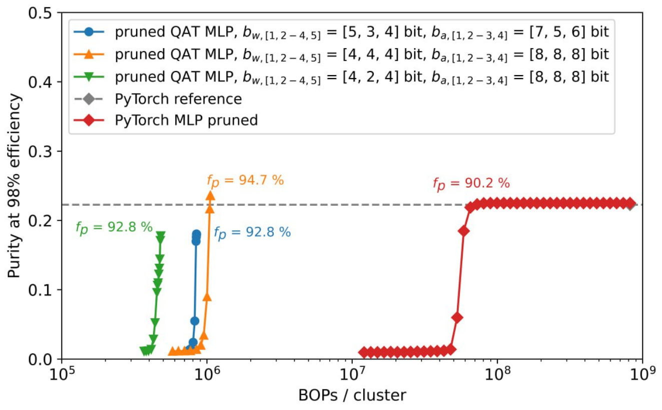


Figure 6: Purity at fixed 98 % efficiency for various models vs BOPs per hit cluster [26]. QAT models are pruned iteratively with L1 loss and retain performance for pruning factors f_p above 92 %. The PyTorch reference model is also pruned and retains performance at f_p =90 % but with larger BOPs compared to any QAT+pruned model.

7 Conclusion

Track reconstruction using GNN on FPGA is under investigation for the ATLAS Event Filter system at the HL-LHC. FPGA resources are mostly occupied by machine learning parts of the pipeline and thus resource reduction is a must. Model compression techniques, like quantization and pruning, were studied in the initial phase, yielding promising resource consumption reduction while maintaining performance. The results presented here were obtained with the TrackML dataset and will be followed up by using realistic

ATLAS ITk simulation samples. Future work will require an estimation of resource consumption for QAT and the pruned model as well. The effect of model compression is yet to be investigated for GNN. Exploring various frameworks for translating models to FPGAs, such as FINN [27], in conjunction with AMD FPGAs [28], is another avenue worth investigating.

ACKNOWLEDGEMENTS

Sachin Gupta 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] I. Zurbano Fernandez et al., High-Luminosity Large Hadron Collider (HL-LHC): Technical design report, https://cds.cern.ch/record/2749422

[3] ATLAS collaboration, Journal Journal of Instrumentation, s08003(2008)

[4] ATLAS Inner Tracker , https://atlas.cern/Discover/Detector/Inner-Detector

[5] ATLAS collaboration, Technical Design Report for the Phase-II Upgrade of the ATLAS TDAQ System, https://cds.cern.ch/record/2285584

[6] https://www.kaggle.com/c/trackml-particle-identification

[7] Xingyang Ju et al., Graph Neural Networks for Particle Reconstruction in High Energy Physics detectors, 2003.11603(2020)

[8] Steven Farrell et al., Novel deep learning methods for track reconstruction, arXiv 1810.06111 2018

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

[10] Gage DeZoort et al., Charged Particle Tracking via Edge-Classifying Interaction Networks, https://arxiv.org/pdf/2103.16701

[11] Xingyang Ju et al., Performance of a geometric deep learning pipeline for HL-LHC particle tracking, Eur. Phys. J. C (2021) 81: 876, https://doi.org/10.1140/epjc/s10052-021-09675-8,2021

[12] Elabd A, Razavimaleki V et al., Graph Neural Networks for Charged Particle Tracking on FPGAs. Front. Big Data 5:828666. doi: 10.3389/fdata.2022.828666

[13] Chinedu et al., Hardware description language (HDL): An efficient approach to device independent designs for VLSI market segments, 10.1109/ICASTech.2011.6145181, 2011

[14] Mahesh A. Iyer, Junaid Khan et al., Advanced Physical Synthesis in the Intel® Quartus® Prime Pro Edition Software,

[15] Javier Duarte et al., Fast inference of deep neural networks in FPGAs for particle physics, JINST 13 P07027, 2018

[16] Paszke, Adam Gross et al., PyTorch: An open source deep learning platform, https://pytorch.org/

[17] Open Neural Network Exchange, https://github.com/onnx/onnx

[18] Intel® Stratix® 10 FPGA and SoC FPGA, https://www.intel.com/content/www/us/en/products/details/fpga/stratix/

[19] Sara Schjødt Kjær, Implementation and optimisation of a Graph Neural Network-based track reconstruction pipeline on Intel FPGAs for the ATLAS TDAQ system for HL-LHC, M. Sc. Thesis, Copenhagen, 2023, https://nbi.ku.dk/english/theses/masters-theses/sara-schjoedt-kjaer/

[20] James O' Neill An Overview of Neural Network Compression, arXiv 2006.03669, 2002.

[21] IEEE Standard for Floating-Point Arithmetic, in IEEE Std 754-2019 (Revision of IEEE 754-2008), vol. no. pp.1-84, 22 July 2019, doi: 10.1109/IEEESTD.2019.8766229.

[22] Markus Nagel and Marios Fournarakis et al., A White Paper on Neural Network Quantization, arXiV 2106.08295, 2021.

[23] Alessandro Pappalardo, Xilinx/brevitas, 10.5281/zenodo.3333552, https://doi.org/10.5281/zenodo.3333552, 2023.

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

[25] Alessandro Pappalardo, Yaman Umuroglu et al., QONNX: Representing Arbitrary-Precision Quantized Neural Networks, arXiv 2206.07527 (2022)

[26] Sebastian Dittmeier et al. "Track reconstruction for the ATLAS Phase-II Event Filter using GNNs on FPGAs", Proceedings of the CHEP 2023 conference (to be published), ATL-DAQ-PROC-2023-006, https://cds.cern.ch/record/2870183

[27] FINN documentation, https://finn-hlslib.readthedocs.io/en/latest/

[28] AMD Vitis Software Platform 2023.2 Release: Accelerate Development of High-Performance Designs, https://community.amd.com/t5/adaptive-computing/amd-vitis-software-platform-2023-2-release-accelerate/ba-p/639533