Prepared for submission to JINST

Machine learning evaluation in the Global Event Processor FPGA for the ATLAS trigger upgrade

Zhixing Jiang[1,*] **Ben Carlson**[3] **Allison Deiana**[4] **Jeff Eastlack**[5] **Scott Hauck**[1] **Shih-Chieh Hsu**[1] **Rohin Narayan**[4] **Santosh Parajuli**[4] **Dennis Yin**[1] **Bowen Zuo**[1]

[1]*University of Washington*
[3]*Westmont College and University of Pittsburgh*
[4]*Southern Methodist University*
[5]*Michigan State University*

E-mail: zhixij@uw.edu*

Abstract: The Global Event Processor (GEP) FPGA is an area-constrained, performance-critical element of the Large Hadron Collider's (LHC) ATLAS experiment. It needs to very quickly determine which small fraction of detected events should be retained for further processing, and which other events will be discarded. This system involves a large number of individual processing tasks, brought together within the overall Algorithm Processing Platform (APP), to make filtering decisions at an overall latency of no more than 8ms. Currently, such filtering tasks are hand-coded implementations of standard deterministic signal processing tasks.

In this paper we present methods to automatically create machine learning based algorithms for use within the APP framework, and demonstrate several successful such deployments. We leverage existing machine learning to FPGA flows such as HLS4ML and fwX to significantly reduce the complexity of algorithm design. These have resulted in implementations of various machine learning algorithms with latencies of $1.2\mu s$ and less than 5% resource utilization on an Xilinx XCVU9P FPGA. Finally, we implement these algorithms into the GEP system and present their actual performance.

Our work shows the potential of using machine learning in the GEP for high-energy physics applications. This can significantly improve the performance of the trigger system and enable the ATLAS experiment to collect more data and make more discoveries. The architecture and approach presented in this paper can also be applied to other applications that require real-time processing of large volumes of data.

Keywords: Accelerator applications; Hardware and accelerator control systems; Trigger detectors; Data processing methods

Contents

1 Introduction

The ATLAS experiment at the Large Hadron Collider (LHC) [1] at CERN is undergoing continuous upgrades as part of the High-Luminosity LHC Upgrade [2] because of the need to handle an increased data output rate and refine data capture accuracy for the future High-Luminosity LHC (HL-LHC) upgrade [3]. The upgrades include a new decision-making module, Global Trigger subsystem, in the L0 Trigger [4], where L0 trigger is the first-level hardware-based decision system selecting relevant collision events for further analysis, which will require new and improved hardware and algorithms to increase its performance.

The upcoming Global Trigger subsystem is designed to run advanced algorithms, similar to those typically used for offline data analysis, on detailed data collected from various sub-detectors and processing units in real time. This approach will enhance the quality of detected events and observables, serving as inputs for the advanced decision-making processes handled by the Global Event Processor (GEP) [5]. As the GEP performs many tasks on the same FPGA, the feasible latency for typical individual algorithms is less than $1.2\mu s$, derived from the $25ns$ time for each bunch crossing (the time between collisions in the detector) and the number of parallel GEP units receiving data in a round-robin fashion (i.e., 25 ns x 48 GEP units). The FPGA resource utilization also must be small enough to incorporate many algorithms, placing practical constraints at the level of a few percent per resource type (LUT, FF, BRAM, DSP).

The GEP, which serves as an FPGA-based framework for an interconnected network of Algorithm Processing Units (APUs), orchestrates the data flow and the processing chain across multiple clock domains to execute the trigger algorithm. Data is pipelined through different APUs within the

GEP, with each APU handling individual sub-tasks of the overall trigger. Specialized algorithms are implemented in each APU for data analysis in a pipeline workflow.

The APU emerges as a paradigm of innovation within the ATLAS experiment's data processing systems, demonstrating superior performance over general-purpose processors. Its distinctive advantage lies in utilizing a single FPGA platform to host various algorithms, which streamlines efficiency by obviating the need for cross-platform conversion. With a specialized protocol, the APU facilitates ease of use for designers, enabling seamless integration of multiple APUs where each focuses on a distinct computational challenge. This modular approach, where individual APUs are dedicated to specific tasks and then unified, significantly amplifies the processing capacity of the Global Event Processor (GEP). Optimized for high-speed processing, the APU surpasses the latency limitations commonly associated with general-purpose processors. Its architecture is intricately designed to manage the complex data flow and algorithmic demands of particle physics experiments, ensuring the delivery of real-time analytics essential for prompt decision-making and dynamic experiment adaptation.

This work is significant because it marks the first time that machine learning tools such as hls4ml and fwX have been used for the ATLAS trigger system. Our paper describes how we deployed these tools into the APU development process, thus simplifying algorithm design and improving APU performance. With the integration of machine learning algorithms into the APU, we have striven towards the theoretical maximum latency of 1.2 microseconds.

The organization of this paper is as follows: Section 2 provides an introduction to the APU architecture and the communication protocols employed between the APUs. In Section 3, we present the APU development process using hls4ml and fwXmachina, and explain the implementation of machine learning algorithms into the APU. In Section 4, we present the results of our experiments and evaluate the performance of the GEP-defined algorithms implemented in the APU. We conclude our work in Section 5.

2 Infrastructure and Methods

The APU is a crucial component in the Global Event Processor (GEP) system, and the primary responsibility of the APU is to swiftly process and analyze the data generated by the particle detectors in real-time. Each APU performs a specific part of the overall computation. Given the high-speed data transmission from the detectors, the APU must match this pace, necessitating additional components within the GEP system. These components, which manage data transmission and synchronization, are critical to ensuring efficient, accurate, and rapid processing, minimizing data loss or corruption.

In the following subsections, we delve deeper into these aspects, discussing data transmission and synchronization and exploring how machine learning tools, specifically hls4ml and fwXmachina, integrate into the APU, enhancing its performance and data handling capabilities.

2.1 Integration of the Algorithm

Machine learning has recently been widely used in particle and energy research, as well as in LHC data analysis. In the APU, although not all algorithms can be achieved using machine learning, some of them can be solved using machine learning approaches, especially those related to particle

tagging or identification problems. For example, the B-tagging algorithm distinguishes between different jet types, including those originating from b-quarks (B-tagging), can be implemented using dense neural networks or convolution neural networks, and the Quark/Gluon jet tagging algorithm can be implemented using a CNN model. However, since the APU is a firmware-based FPGA design, neural network deployments in GPU code are not supported. Hence, hls4ml and fwX were applied to implement the neural networks on the FPGA. In this section, we will introduce how to integrate a machine learning model into an APU.

A key element in this integration is the consistent application of an Algorithmic State Machine (ASM), which serves as a bridge between the APU's firmware-based FPGA architecture and the ML models. Notably, both hls4ml and fwX, used for the generation of these ML models, employ Vivado HLS for creating Verilog code. This results in a similar structure and protocol across different ML models, allowing for a standardized approach in the ASM's application.

The ASM's primary function is to manage the protocol differences between the APU's FPGA design, which typically uses an addressable input memory buffer, and the streaming data model inherent to ML models. It ensures seamless data transmission, effectively converting the incoming data into a streaming format compatible with the ML models and formatting the output data for the APU's consumption. This process involves the ASM transitioning through various states – from an initial idle state to active data transfer, and finally to completion – ensuring efficient and accurate data handling.

The uniformity in the ASM design, dictated by the similar structure of the ML models generated by hls4ml and fwX, simplifies the integration process. It allows the APU to handle different types of ML algorithms without requiring significant alterations in the ASM structure or its operational methodology.

The detailed experimental results, which will be discussed in subsequent sections, highlight the effectiveness of integrating these diverse ML models into the APU. These results include comprehensive analyses of resource utilization, latency, and overall performance, demonstrating the practicality and efficiency of this integration approach.

In conclusion, the standardized ASM approach significantly enhances the APU's capability to manage a wide range of computational tasks, thereby bolstering the data processing prowess required for LHC experiments. This integration not only represents a technical achievement but also a crucial step forward in the field of high-energy physics research.

2.2 Data Transmission and Synchronization

In the GEP, raw input events arrive every $1.2\mu s$, with intervening inputs sent to additional GEP modules. Individual APUs perform portions of the overall computation, with data streaming in a fixed dataflow graph from APP to APP, where an APP is a container of an APU. Parallel paths in this dataflow graph represent different portions of the computation, while parallel execution units for a given step would be contained within an individual APU, as demonstrate in figure 1. BRAM-based buffers are placed in-between communicating APUs to store the input or output information from each APU, and allow parallel operation in the producer and the consumer. As illustrated in figure 2, BRAMs are stacked together to form a bank that stores data for multiple events. These data sources can be raw data from the detector or data from an upstream APU. An APU processes one event at a time, receiving data from the upstream BRAMs and storing the resultant data in a downstream

Algorithm 1 The ASM for streaming the data input/output to the DNN/BDT

```
Param_Delay ← n;
state ← IDLE;
while event_ready do
    if read_state = IDLE then
        if ready then
            counter ← data[0]          ▷ the first data contains the index of the last valid data
            read_state ← TRANSFER
        end if
    else if read_state = TRANSFER then
        enable_NN_in ← 1;
        for i ← 0 to counter − 1 do
            data ← read_upstream_BRAM(.addr(i));
            send_data_to_NN(data);
        end for
        read_state ← IDLE;
    end if
    if write_state = IDLE then
        if NN_output_valid then
            write_state ← TRANSFER;
        end if
    else if write_state = TRANSFER then
        enable_apu_out ← 1;
        for i ← 0 to counter − 1 do
            data ← read_Dense_output;
            send_data_out(data);
        end for
        write_state ← END;
    else if write_state = END then
        send_data_out(last_data_index);
        event_done ← 1;
    end if
end while
```

BRAM. Fanout in the dataflow graph is supported by parallel copies of the downstream memory buffers.

To address the significant challenge of data synchronization, given the arrival skew of raw data inputs and unsynchronizied clock speeds from the detectors, the Algorithm Processing Platform (APP) was developed. The APP serves as a wrapper for each APU and facilitates Clock Domain Crossing (CDC) through its sub-modules.

The APP comprises Synchronization Registers (SR), BRAMs, a Sync controller, and the APU itself. The BRAMs in the APP operate under two clocks: one that writes data from upstream

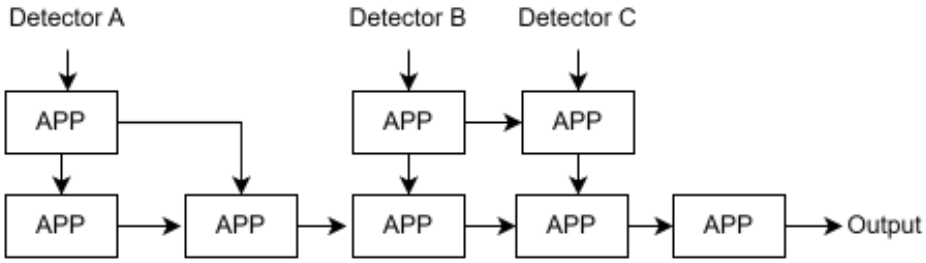

Figure 1. The dataflow of the APUs within the GEP

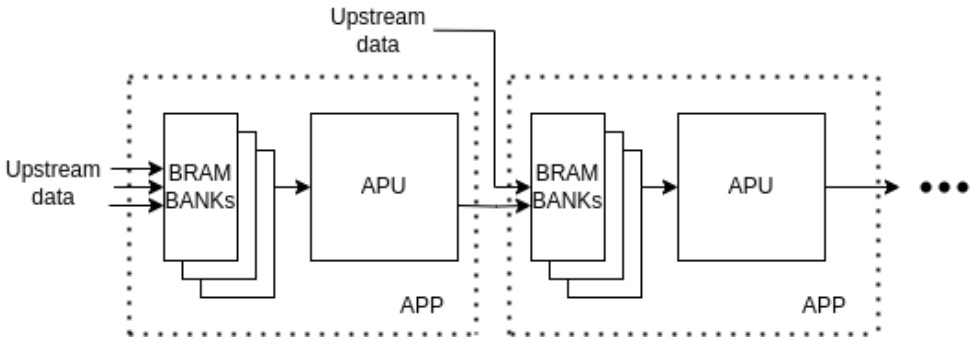

Figure 2. The communication between two APPs in a detailed view

and another that reads data for the APU within the APP. This dual-clock operation enables the transfer of data between different clock speeds. The SR, tasked with determining when data from a particular input source is ready, controls a stack of BRAMs in the APP and governs data storage and retrieval. The Sync controller, which contains a Finite State Machine (FSM), regulates the SRs for the selection of BRAMs, with the chosen BRAM sending or receiving data to or from the APU. The APP provides the solution to data synchronization through the BRAM banks. By managing the synchronization registers and the Sync controller, it ensures data consistency from different clock domains and guarantees that the APU processes data from the correct event, even with the presence of raw data input skew.

All trigger processing for a given Bunch Crossing (BC - an event in the detector) is handled in a single GEP. To process multiple events under significant throughput and latency constraints, the 48 GEP units operate in a round-robin fashion, where GEP1 processes data from BC1, followed by data from BC49, and so forth. Data processing within the APUs of GEP is pipelined, such that upstream APUs may be processing data for BC49, while while downstream APUs may still be processing data for BC1; in fact, we expect a plurality of BC's to be processed simultaneously within each GEP.

2.3 Hls4ml

The trigger upgrade project aims to develop a low latency data processing system for high-energy physics. To help achieve this, the project is utilizing a high-level synthesis tool [6] to convert ma-

chine learning models into FPGA firmware. High-Level Synthesis for Machine Learning (hls4ml) is an open-source software package that provides a user-friendly interface for converting high-level machine learning models into hardware implementations. The tool generates hardware designs in hardware description languages (HDLs) such as VHDL or Verilog, which can then be synthesized and implemented on FPGAs. The workflow of hls4ml is: 1) automatically converting a machine learning model from TensorFlow [7], Pytorch [8], or Keras [9] into an hls4ml project that is output in a hardware-oriented subset of C++; 2) using Vivado HLS to synthesize the C++ code into HDL; 3) Using Vivado to synthesize the HDL into an FPGA bitstream. Figure 3 shows the workflow of hls4ml. Hls4ml has been used in various high-energy physics experiments, including the Fermilab booster [10].

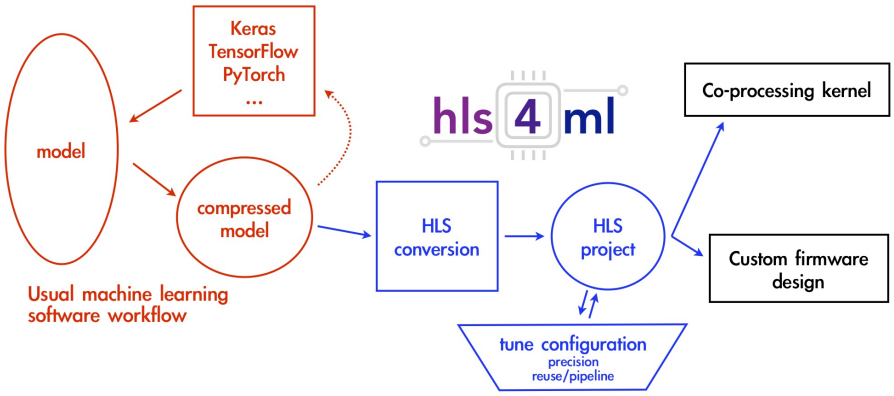

Figure 3. The workflow of hls4ml, hls4ml will first read the model from Pytorch, Tensorflow, or Keras, then convert to the hardware descriptive language using Vivado HLS, and eventually to the FPGA

Hls4ml is a promising tool for APU designs for several reasons. First, hls4ml is a convenient way to automatically convert a machine learning model into RTL, allowing for quick generation of different machine learning architectures. The user only needs to create the model using standard approaches in TensorFlow or Pytorch, and hls4ml can do the conversion to hardware. This saves designers significant amounts of time in implementing complex machine learning algorithms. Second, hls4ml can optimize hardware architectures for specific performance metrics, such as latency, throughput, or power consumption. This makes it a powerful tool for implementing real-time applications, such as those required by high-energy physics experiments. Third, hls4ml supports many different machine learning models, including dense neural network (DNN) [6], convolution neural network (CNN) [11], recurrent neural networks (RNN) [12], and graph neural networks (GNNs) [13, 14].

2.4 FwXmachina

The software package fwXmachina is used for implementing boosted decision tree-based machine learning algorithms onto FPGAs for high-energy physics applications [15–17]. Similar to hls4ml, it uses Vivado HLS to convert the model into RTL. It operates via a three-stage process: machine learning training with external software packages, optimization to fine-tune BDT structures and

parameters for physics performance and FPGA cost, and conversion to the firmware design through vendor tools.

The fwX software package has been used to implement nanosecond machine learning with deep decision trees that have been used for problems that include event classification, regression, and anomaly detection. These implementations have achieved high accuracy and low latency, making them suitable for real-time applications. The parallel decision paths architecture of fwX allows for efficient use of FPGA resources, resulting in high-performance implementations. Its ability to efficiently implement decision trees with large numbers of branches and leaves makes it a valuable tool for applications.

BDTs have been extensively utilized in high-energy physics applications, for instance in the discovery of the Higgs boson by the ATLAS and CMS collaborations [18, 19]. In this context, fwXmachina proves invaluable by efficiently implementing complex BDT models on FPGA, which has low latency (in nano second scale) and small resource usage.

The potential of fwXmachina is underscored by its remarkable performance metrics. In one study [15], for a complex BDT model with 100 training trees, a maximum depth of 4, and four input variables, it boasts a latency of only around 10 ns, or 3 clock ticks at 320 MHz. Notably, this level of performance is achieved with minimal resource utilization - less than 0.2% of look-up tables and block RAM usage, less than 0.01% of flip-flop usage, and no ultra RAM or digital signal processor (DSP) usage. This efficiency demonstrates fwXmachina's capacity to provide high-speed, low-resource implementations without compromising on the complexity or accuracy of the machine learning models.

3 Experimental Result

3.1 Deep Neural Network for B-tagging

In the pursuit of refining particle identification within the ATLAS GEP, a Deep Neural Network (DNN) has been integrated into the APU, specifically focusing on a Jet tagging task. This task plays a crucial role in identifying the types of particles, particularly in distinguishing between different jet types, including those originating from b-quarks (B-tagging).

The employed DNN model for B-tagging is structured with four dense layers consisting of 16, 32, 32, and 5 neurons, respectively. The final layer employs softmax activation for classifying input data into five distinct categories, tailored to differentiate various particle types accurately. Figure 4 illustrates the DNN architecture, showcasing its layered structure and neuron configuration, which is pivotal for the B-tagging application.

The resource utilization of this DNN model is depicted in Table 1. The model demonstrates a balance between low latency and minimal resource usage, which is essential for real-time processing in the APU. With a latency of just 10 cycles, or 50ns at a 200MHz clock rate, this model exemplifies the feasibility of using hls4ml-generated machine learning models in APUs for high-energy physics experiments.

This B-tagging DNN model not only fulfills the real-time processing requirements but also highlights the effectiveness of implementing advanced machine learning techniques in the field of high-energy physics. The efficient use of FPGA resources, combined with the high-speed processing

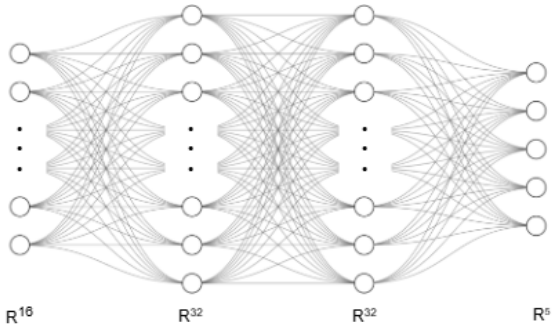

Figure 4. The architecture of the dense neural network

Table 1. The resource usage of the B-tagging DNN model

Resource	Utilization	Utilization %
DSP	625	9.1
FF	9646	0.41
LUT	54441	4.6
BRAM	18	0.83

capabilities, positions this approach as a valuable asset for current and future experiments in the ATLAS GEP.

3.2 VBF Classification in BDTs

Machine learning algorithms in the form of neural networks and boosted decision trees (BDT) are commonly used to separate signals and backgrounds in high-energy physics experiments. Examples include hadronic τ lepton identification [20] and identification of jets that contain a b-hadron [21].

As an example for BDT classification in the ATLAS GEP, we use the problem of separating vector boson fusion Higgs production from multijet background. We utilize the samples produced for the fwX classification paper [15]. Further details, as well as input distributions, are available in the fwX paper [15] and the corresponding public dataset [22]. As the VBF trigger is dominated by high transverse momentum (p_T) jets, we assume that the hardware studies performed will be a reasonable representation of the GEP performance.

The classifier is trained using kinematic variables corresponding to the two VBF jets. These include the transverse momentum of the sub-leading jet p_{T2}, and calculated quantities on the two VBF jets. These calculated quantities include the vector sum $p_T(jj)$, the scalar sum, H_T(jj), and the invariant mass of the two jets m_{jj}. To account for jets in opposite hemispheres of the detector, the product of the two jet pseudo-rapidity values are computed: $\eta_1 \cdot \eta_2$. The range and number of bits assigned to each input variable is summarized in Table 2.

The BDT model is trained using the TMVA [23] package, which implements the AdaBoost [24] method with 100 trees and a max depth of 4. During the simplification step performed by fwX, the number of trees was reduced to 10.

The performance of the model implemented in the APU is evaluated by examining the latency, as well as the FPGA resource costs using the Xilinx FPGA VU9P chip. The latency was evaluated to

Table 2. Input variables, range of each variable and number of bit assigned to each variable.

Variable	Range	bits
$\eta_1 \cdot \eta_2$	-20–20	12
p_{T2}	0 – 1000 GeV	12
p_T(jj)	0 – 1500 GeV	12
H_T(jj)	0 – 1500 GeV	12
m_{jj}	0 – 4500 GeV	7

Table 3. The resource usage of the classification BDT model

Resource	Utilization	Utilization %
DSP	2	0.029
FF	597	0.025
LUT	2756	0.23
BRAM	48	2.2

Table 4. The resource usage of the regression BDT model, post-synthesis. The utilization is given in the total number of available units utilized as well as the fraction available on the FPGA in %.

Resource	Utilization	Utilization (%)
DSP	0	0.0
FF	1987	0.084
LUT	3493	0.30
BRAM	12	0.56

be 7 clock cycles with the clock running at a rate of 320MHz, which means the latency is 21.875ns. The resource usage is shown in Table 3. These results underscore the extremely low resource consumption on the FPGA, showcasing its practicality and effectiveness.

3.3 Missing Transverse Momentum Regression BDT

Regression models are useful for a wide variety of physics applications, including reconstruction of missing transverse momentum, E_T^{miss} [25] and hadronic τ leptons [26]. To evaluate the hardware performance of a regression model in the APU, a regression model to evaluate E_T^{miss} is studied. The implementation in the fwX regression studies was originally performed using public Delphes samples [27] described in Ref [16].

In particular, this model is trained to identify the true E_T^{miss} based on a simulated sample of Higgs boson events that decay to neutrinos that do not interact with the detector. The eight input variables are described in Ref. [16]. The regression model is configured with 40 trees, a tree depth of 6.

The performance of the model implemented in the APU is evaluated by examining the latency, as well as the FPGA resource costs using the Xilinx FPGA VU9P chip. The latency was evaluated to be 11 clock cycles with the clock running at a rate of 320MHz, which makes the latency 34 nanoseconds. The resource usage is shown in the Table 4.

3.4 Quark-Gluon Jet Tagging Algorithm

The capacity to distinguish between quark-originated and gluon-originated jets is widely applicable to numerous physics investigations at the LHC[28–30]. This section introduces a technique for differentiating quark-based and gluon-based jets by employing a deep neural network classifier that analyzes the complete radiation pattern within a jet as an image. The energy deposits in the calorimeters serve as inputs for the jet reconstruction and classification algorithm. The energy deposit organization scheme makes use of topological calorimeter-cell clusters (topo-clusters)[31]. Topo-clusters are used as input for jet reconstruction with the anti-k_t jet algorithm[32] with distance parameter $R = 0.4$. Jets labeled as gluon or quark (excluding top quark) are considered. Jets with transverse momentum (p_T) between 50 and 75 GeV and $|\eta| < 2.5$ are selected where η is the pseudorapidity. Jets are required to satisfy generator-level matching criteria: the jet must be matched to a parton-level quark or gluon and all of its decay products within $\Delta R = 0.4$ where $\Delta R = \sqrt{(\Delta\eta)^2 + (\Delta\phi)^2}$ and ϕ is the azimuthal angle.

As a first step in constructing a jet image, the constituents inside a jet are translated in η and ϕ so that the jet's center is located at the center in η-ϕ space. Then, a fixed grid of size 15×15 in η and ϕ with pixel sizes 0.055×0.055 is centered on the origin. The intensity of each pixel is the total E_T within the pixel, using topocluster input. Pixel values are then normalized by dividing them by the value of the hottest (maximum) pixel in the image. This scaling ensures that the pixel values of the entire image are between 0 and 1. Then, the pixel values are scaled to a range between 0 and 255, this is done by multiplying each pixel value by 255.

In this study, we utilize images of jets as input for a deep neural network classifier, specifically a deep convolutional neural network (CNN). The CNN[33] architecture we employ involves a convolutional layer with ReLU activation, paired with a Max-pooling layer. The network outputs a softmax function that predicts the probability of a quark or gluon jet. The convolutional layer includes 4 filters with filter sizes of 2x2, while the Max-pooling layers perform a 2x2 down sampling. To avoid overfitting, we employ dropout on the convolutional and final fully connected layers at a rate of 0.1. Training is performed by minimizing the binary cross-entropy, using the Adam optimizer[34] implemented in Keras with a learning rate of 0.0001 over 100 iterations and a batch size of 256. The training dataset contains approximately 105K events, while the test dataset consists of around 26K events.

For this CNN algorithm, we convert it into an FPGA implementation via the hls4ml toolchain. The performance of the model implemented in the APU is evaluated by examining the latency, as well as the FPGA resource costs using the Xilinx FPGA VU9P chip. The latency was evaluated to be 233 clock cycles with the clock running at a rate of 200MHz, which makes the latency 1.2 microseconds. The resource usage is shown in the table 5.

4 Conclusion

In this paper, we developed mechanisms to easily implement machine learning based algorithms into the Algorithm Processing Unit for the ATLAS Global Event Processor. We tested Boosted Decision Tree and Neural Network models prepared using the fwX and hls4ml tools respectively.

Our study underscores the efficacy of machine learning tools when integrated into the APU framework, as demonstrated by the performance evaluation presented in Table 6. The various

Table 5. The resource usage of the qg tagger CNN model

Resource	Utilization	Utilization %
DSP	305	4.5
FF	4812	0.20
LUT	7504	0.63
BRAM	9	0.42

machine learning models, ranging from the VBF classifier to the more complex q/g CNN, are implemented with impressive efficiency, maintaining latency values from as low as 22ns up to $1.2\mu s$. Notably, the resource utilization for these models remains commendably low, with less than 10% of the total resources of the FPGA VCU118 being employed, even for the more resource-intensive B-tagging DNN. This data indicates not only the high efficiency of our integrated ML models but also showcases the scalable complexity of the models that the APU can support. The proportional increase in resource usage, such as the LUT and DSP consumption, aligns with the enhanced capabilities and complexities of the respective algorithms, thereby validating the APU's capability to execute advanced computational tasks within the stringent requirements set by the GEP.

As we look to the future, this work lays the groundwork for the integration of increasingly complex machine learning models, which could further enhance the performance of APU. The methodologies presented in this paper have potential applications in various experimental setups, thereby contributing to the continuous improvement and evolution of real-time data processing systems. With ongoing advancements in machine learning and FPGA technologies, the application of tools such as hls4ml and fwX may become even more critical at the nexus of high-energy physics and real-time data processing. For instance, the deployment of recurrent neural network (RNN) implementations on FPGAs, as discussed in [12], or the advancements in real-time data processing illustrated in [35, 36], exemplify the expanding scope of these technologies.

Overall, this work emphasizes the ability to easily deploy the hls4ml and fwX tools, demonstrating their successful application in meeting the needs of the next generation of the LHC's high-speed data processing systems.

Table 6. Comparison of Model Complexities

	VBF classifier	**MET regression**	**B-tagging DNN**	**q/g CNN**
Tool	fwX (Depth = 4)	fwX (Depth = 6)	HLS4ML	HLS4ML
Clock	320 MHz	320 MHz	200 MHz	200 MHz
Latency	22 ns	34 ns	50 ns	1.2 us
LUT	0.23%	0.30%	4.6%	0.63%
DSP	0.029%	0.0%	9.1%	4.5%
FF	0.025%	0.084%	0.41%	0.20%
BRAM	2.2%	0.56%	0.83%	0.42%

Acknowledgments

We acknowledge the ATLAS Global Even Processor group as a supportive community of experts and collaborators. This group was important for the development of this project. We particularly thanks Wade Fisher offers clear guidance to conduct this project.

Jiang, Hauck and Hsu are supported by National Science Foundation (NSF) grants No. 2117997. Carlson is supported by NSF grant No. 2209370 and No. 2117997 and would like to thank Steve Roche for technical support with fwX.

References

[1] L. Evans and P. Bryant, eds., *LHC Machine*, *JINST* **3** (2008) S08001.

[2] G. Apollinari, O. Brüning, T. Nakamoto and L. Rossi, *High Luminosity Large Hadron Collider HL-LHC*, *CERN Yellow Rep.* (2015) 1 [1705.08830].

[3] ATLAS Collaboration collaboration, "System Specification for the Global Trigger." ATL-COM-DAQ-2021-093, Nov, 2021.

[4] ATLAS Collaboration collaboration, "Technical Design Report for the Phase-II Upgrade of the ATLAS TDAQ System." ATLAS-TDR-029, Sep, 2017. 10.17181/CERN.2LBB.4IAL.

[5] G.T. Community, *Atlas tdaq phase-ii upgrade: Firmware specifications for the global trigger*, *CERN* (2021) .

[6] J. Duarte, S. Han, P. Harris, S. Jindariani, E. Kreinar, B. Kreis et al., *Fast inference of deep neural networks in FPGAs for particle physics*, *Journal of Instrumentation* **13** (2018) P07027.

[7] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro et al., *TensorFlow: Large-scale machine learning on heterogeneous systems*, 2015.

[8] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan et al., *Pytorch: An imperative style, high-performance deep learning library*, in *Advances in Neural Information Processing Systems 32*, pp. 8024–8035, Curran Associates, Inc. (2019), http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[9] F. Chollet et al., "Keras." https://keras.io, 2015.

[10] J. St. John et al., *Real-time artificial intelligence for accelerator control: A study at the Fermilab Booster*, *Phys. Rev. Accel. Beams* **24** (2021) 104601 [2011.07371].

[11] T. Aarrestad et al., *Fast convolutional neural networks on FPGAs with hls4ml*, *Mach. Learn. Sci. Tech.* **2** (2021) 045015 [2101.05108].

[12] E.E. Khoda, D. Rankin, R.T. de Lima, P. Harris, S. Hauck, S.-C. Hsu et al., *Ultra-low latency recurrent neural network inference on fpgas for physics applications with hls4ml*, 2022.

[13] Y. Iiyama et al., *Distance-Weighted Graph Neural Networks on FPGAs for Real-Time Particle Reconstruction in High Energy Physics*, *Front. Big Data* **3** (2020) 598927 [2008.03601].

[14] A. Heintz et al., *Accelerated Charged Particle Tracking with Graph Neural Networks on FPGAs*, in *34th Conference on Neural Information Processing Systems*, 11, 2020 [2012.01563].

[15] T.M. Hong, B.T. Carlson, B. Eubanks, S. Racz, S. Roche, J. Stelzer, D. Stumpp, *Nanosecond machine learning event classification with boosted decision trees in FPGA for high energy physics*, *JINST* **16** (2021) P08016 [2104.03408].

[16] B. Carlson, Q. Bayer, T.M. Hong and S. Roche, *Nanosecond machine learning regression with deep boosted decision trees in FPGA for high energy physics*, *JINST* **17** (2022) P09039 [2207.05602].

[17] S. Roche, Q. Bayer, B. Carlson, W. Ouligian, P. Serhiayenka, J. Stelzer et al., *Nanosecond anomaly detection with decision trees for high energy physics and real-time application to exotic Higgs decays*, 2304.03836.

[18] ATLAS Collaboration, *Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC*, *Phys. Lett. B* **716** (2012) 1 [1207.7214].

[19] CMS Collaboration, *Observation of a New Boson at a Mass of 125 GeV with the CMS Experiment at the LHC*, *Phys. Lett. B* **716** (2012) 30 [1207.7235].

[20] CMS collaboration, *Identification of hadronic tau lepton decays using a deep neural network*, *JINST* **17** (2022) P07023 [2201.08458].

[21] ATLAS collaboration, *Measurements of b-jet tagging efficiency with the ATLAS detector using $t\bar{t}$ events at $\sqrt{s}$ = 13 TeV*, *JHEP* **08** (2018) 089 [1805.01845].

[22] S. Roche, B. Carlson and T.M. Hong, "fwXmachina example: VBF Higgs vs multijet." Mendeley Data, 2021. 10.17632/kp3myh3v89.1.

[23] A. Hoecker et al., *TMVA - Toolkit for Multivariate Data Analysis*, 2007.

[24] Y. Freund and R.E. Schapire, *A desicion-theoretic generalization of on-line learning and an application to boosting*, in *Computational Learning Theory*, P. Vitányi, ed., (Berlin, Heidelberg), pp. 23–37, Springer Berlin Heidelberg, 1995.

[25] CMS collaboration, *Performance of missing transverse momentum reconstruction in proton-proton collisions at $\sqrt{s}$ = 13 TeV using the CMS detector*, *JINST* **14** (2019) P07004 [1903.06078].

[26] ATLAS collaboration, *Reconstruction of hadronic decay products of tau leptons with the ATLAS experiment*, *Eur. Phys. J. C* **76** (2016) 295 [1512.05955].

[27] S. Roche, B. Carlson and T.M. Hong, "fwXmachina example: Missing transverse energy regression." https://data.mendeley.com/datasets/d4c94r9254/1, 2022. 10.17632/d4c94r9254.1.

[28] ATLAS collaboration, *Quark versus Gluon Jet Tagging Using Jet Images with the ATLAS Detector*, Tech. Rep. ATL-PHYS-PUB-2017-017, CERN, Geneva (2017).

[29] J.S.H. Lee, I. Park, I.J. Watson and S. Yang, *Quark-gluon jet discrimination using convolutional neural networks*, *Journal of the Korean Physical Society* **74** (2019) 219.

[30] P.T. Komiske, E.M. Metodiev and M.D. Schwartz, *Deep learning in color: towards automated quark/gluon jet discrimination*, *Journal of High Energy Physics* **2017** (2017) .

[31] M. Aaboud, G. Aad, B. Abbott, J. Abdallah, O. Abdinov, B. Abeloos et al., *Jet reconstruction and performance using particle flow with the atlas detector*, *The European Physical Journal C* **77** (2017) 1.

[32] M. Cacciari, G.P. Salam and G. Soyez, *The anti-kt jet clustering algorithm*, *Journal of High Energy Physics* **2008** (2008) 063.

[33] I. Goodfellow, Y. Bengio and A. Courville, *Deep Learning*, MIT Press (2016).

[34] D.P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, 2017.

[35] N.M. Michels, A.J. Jinia, S.D. Clarke, H.-S. Kim, S.A. Pozzi and D.D. Wentzloff, *Real-time classification of radiation pulses with piled-up recovery using an fpga-based artificial neural network*, *IEEE Access* **11** (2023) 78074.

[36] N. Ghielmetti, V. Loncar, M. Pierini, M. Roed, S. Summers, T. Aarrestad et al., *Real-time semantic segmentation on fpgas for autonomous vehicles with hls4ml*, 2022.