2022 IEEE 30th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM) | 978-1-6654-8332-2/22/$31.00 ©2022 IEEE | DOI: 10.1109/FCCM53951.2022.9786085

FAXID: FPGA-Accelerated XGBoost Inference for Data Centers using HLS

Archit Gajjar*, Priyank Kashyap*, Aydin Aysu*, Paul Franzon*, Sumon Dey† and Chris Cheng‡
*Department of Electrical and Computer Engineering
North Carolina State University, Raleigh, North Carolina 27606
Email: {amgajjar, pkashya2, aaysu, paulf}@ncsu.edu
†Hewlett Packard Enterprise, San Jose, California, USA
‡Hewlett Packard Enterprise, Colorado Springs, Colorado, USA
Email: {sumon.dey, chris.cheng}@hpe.com

***Abstract*—Advanced ensemble trees have proven quite effective in providing real-time predictions against ransomware detection, medical diagnosis, recommendation engines, fraud detection, failure predictions, crime risk, to name a few. Especially, XGBoost, one of the most prominent and widely used decision trees, has gained popularity due to various optimizations on gradient boosting framework that provides increased accuracy for classification and regression problems. XGBoost's ability to train relatively faster, handling missing values, flexibility and parallel processing make it a better candidate to handle data center workload. Today's data centers with enormous Input/Output Operations per Second (IOPS) demand a real-time accelerated inference with low latency and high throughput because of significant data processing due to applications such as ransomware detection or fraud detection.**

This paper showcases an FPGA-based XGBoost accelerator designed with High-Level Synthesis (HLS) tools and design flow accelerating binary classification inference. We employ Alveo U50 and U200 to demonstrate the performance of the proposed design and compare it with existing state-of-the-art CPU (Intel Xeon E5-2686 v4) and GPU (Nvidia Tensor Core T4) implementations with relevant datasets. We show a latency speedup of our proposed design over state-of-art CPU and GPU implementations, including energy efficiency and cost-effectiveness. The proposed accelerator is up to 65.8x and 5.3x faster, in terms of latency than CPU and GPU, respectively. The Alveo U50 is a more cost-effective device, and the Alveo U200 stands out as more energy-efficient.

I. Introduction

Decision trees (DTs) are, basically, tree-like structures that model various outcomes of a particular problem depending on the inputs. These algorithms have proven quite productive in providing predictions in the following fields: 1) cyber-security, 2) recommendation engines, 3) credit card fraud detection, 4) hard-drive failure, 5) medical diagnosis, 6) weather predictions, among others [1]. The large DT-based model's complexities and memory requirements can be beyond the limited available resources on the CPUs and GPUs [2]. Additionally, due to the emerging Internet of things (IoT) field, the data center servers consistently observe more than a few million IOPS (MIOPS). It is preferred to address the problem of such enormous workloads with pure FPGAs or an FPGA-CPU co-design approach [3].

FPGAs can offer a high level of parallelism and acceleration, offering high throughput and low latency with careful design choices. Moreover, CPUs and GPUs tend to fail to provide sustainable performance because of the constant branch divergence while making model-based decisions [4]. The sequential comparisons of tree levels increase cache misses due to random memory accesses. Hence, the pipeline often observes stalls and fails to utilize the caches efficiently. Another reason for to inefficient use of caches is the unknown packets or samples. For example, when it comes to real-time ransomware detection, the samples are unknown and require immediate screening before they can be used in any way [5]. The malicious sample(s) immediately starts encrypting the valuable files as soon as it enters the system, putting the attacked machine and organization in a precarious situation. To, ideally, protect the data or keep the data loss minimum, classification demands imminent detection with latency as low as possible [6].

The proposed accelerator provides high performance while being energy efficient. In this work, we define the performance in terms of low latency, high throughput, and number of IOPS. We also measure the energy efficiency and cost-effectiveness of the accelerator. The accelerator can handle enterprise-level data workloads where CPUs and GPUs often fail [7]. We employ Xilinx Vitis HLS design flow to develop end-to-end acceleration applications and the Alveo series FPGA cards that are specially designed for data center acceleration [8]. We extensively benchmark latency, throughput, MIOPS, and power results against existing state-of-art CPU and GPU designs for a comparison. Moreover, we compare the proposed work with existing FPGA works. We also evaluate the accuracy, precision, confusion matrix, recall, and F1-score to verify the detection rates.

The main contributions of this work are the following:

- A scalable and customizable design architecture to tree depth, number of trees, number of samples, number of features, and datatype limited by available resource on a particular FPGA (subsection 4.2)
- A novel architecture allowing batch processing with the same loaded tree hiding main memory access latency and cautious design choices to map 12 compute units (CUs)

on the FPGAs (subsection 4.2 & 4.3)
- A high-performance and one of the leading accelerators with the ability to solve multiple problems in parallel to the best of authors' knowledge (subsection 4.3)
- A customized tree node structure to reduce the resource consumption on FPGAs without any restrictions on the number of trees limit (subsection 4.4)
- A highly optimized XGBoost inference application on FPGA that provides sustainable performance that is energy and cost-efficient (subsection 5.4)
- An exclusive benchmarking on CPU, GPU, and FPGAs comparing latency, throughput, and IOPS, including energy and cost analysis (subsection 5.4)

The remainder of the paper is organized in the following manner: Section 2 provides the theoretical background on XGBoost, while Section 3 offers related literature and prior work. Section 4 depicts and elucidates the proposed architecture. Section 5 presents experimental setup and benchmark validation, and Section 6 concludes the paper.

II. Background

This section gives a high-level description of XGBoost binary classification inference. In this paper, offline training is assumed, and the work focuses on accelerating the inference of the XGBoost. With this discussion, we aim to explain the working of XGBoost and how its prediction method is different from other tree-based algorithms.

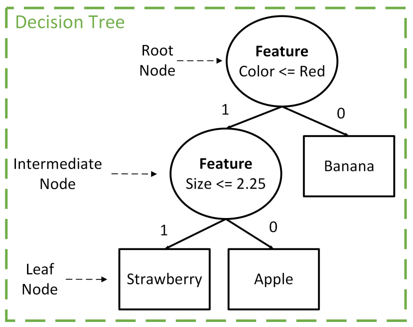
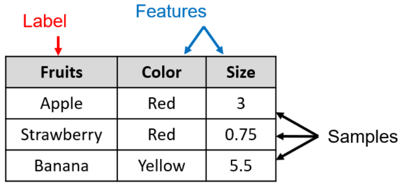


Fig. 1. Tree-Structure of any decision tree algorithm. Each tree consists of root, intermediate, and leaf nodes. Decisions in a form of comparisons are made to traverse from root to leaf node.


Label
Features
Samples

Fruits	Color	Size
Apple	Red	3
Strawberry	Red	0.75
Banana	Yellow	5.5

Fig. 2. A synthetic dataset for an explanation. Feature values are compared with the tree shown in Fig. 1 to identify a fruit-A simple tree traversal method.

The ML-based tree structures fundamentally work on decisions by comparing the input values with the trained model, whether it be Decision Trees (DTs), Random Forest (RF), Gradient Boosting Machine (GBM), or eXtreme Gradient Boosting (XGBoost) trees [9]–[11]. The algorithm compares the first value with the root node for any inputs. Based on the comparison values, the tree traversal is followed by the subsequent comparison until the algorithm identifies a leaf node. Once the algorithm collects the leaf node value, it performs a compute step that varies depending on the tree type. For example, in RF, each tree eventually votes 0 or 1 to contribute to an outcome. Afterward, the RF algorithm averages the votes from all the trees to make a prediction. However, in XGBoost, the final prediction is calculated with the sigmoid function instead of an average.

Fig. 1 and Fig. 2 explain the tree traversal using a simple demonstrative dataset. During actual inference, preprocessing the values in column *Color* to encode into numerical values is essential; in this example, 1 for red and 2 for yellow. As presented in Fig. 2, the DT uses the sample features to classify into a label 0 or 1. The root node depicted in Fig. 1 requests the feature *Color* from the sample and traverses the tree based on the outcome of the color being *red*. For the first sample from the dataset, that would be true, so the left branch is taken. The subsequent node requests feature *Size* and compares the value with *2.25*. Based on the outcome of the first sample, the predicted value would be *Apple*.

III. Related Work

This section focuses on an overview of the current literature on hardware-based solutions on the RF, GBM, and XGBboost highlighting the challenges involved in implementing these algorithms on FPGA and how our proposed work is distinct from the existing works.

Several implementations of DTs, RF, GBM, or XGBoost on FPGA realized hardware [12]–[15]. Prior works demonstrate the efficiency of XGBoost over any other tree-based algorithm [16]–[18]. More specifically, DT and RF tree algorithms focus on collecting votes of all trees and applying voting mechanisms to predict the outcome. These voting techniques consist of majority voting, averaged voting, or weighted voting [19]. On the other hand, GBM works on the principle of gradient boosting. XGBoost, a specific implementation of gradient boosting extensively uses regularized model formalization offering relatively higher performance. Additionally, its ability to handle missing values, regularization, flexibility, and parallel processing makes it an excellent candidate for acceleration [11].

Some works focus on the training aspect of the algorithm—other accelerate inference of the algorithms on FPGAs. In [20]–[23], training acceleration is the primary focus, which differs from our primary motivation. We assume the model trains on an offload machine, and, in this work, the model is one of the inputs to the FPGA for inference. The overall architecture of a design depends on the timing requirements [24]. Application-specific approaches that leverage specific optimizations are widely implemented and meet the best timing constraints [25]–[29]. In contrast to application-specific implementations, our proposed design is scalable and not restricted by a large number of trees, samples, features, models, and datatypes.

Moreover, an HLS approach is also used through Vivado HLS or Altera SDK kit to design FPGA accelerators [29], [30]. One of the state-of-the-art works capable of real-time inference for enterprise-level workload and their specifications [31], [32]. The proposed design employs Vitis tools from Xilinx. The Vitis platform is a specialized development environment for accelerating Xilinx embedded platforms or Alveo accelerator cards for on-premise or cloud deployment.

Compared to previous works, our proposed work is scalable, flexible, cost-effective, energy-efficient, and offers high-performance for large data center level workload. Additionally, the proposed accelerator can handle multiple problems that data centers can exploit to conserve resources. We tackle the I/O bottleneck at the server level by replicating the XGBoost inference CUs to run in parallel. Though the proposed work supports compact and shallow tree models, the architectural novelty can practically handle any number of trees to compensate for the restricted maximum depth of a single tree.

IV. Proposed Architecture

We propose a novel XGBoost binary classification architecture that exploits the FPGAs' reconfigurability and parallelism. The architecture can handle single-precision floating-point (float32) datatype. We deliberately choose float32 over other design choices to retain the maximum achievable accuracy. In the following subsections, we share complete architectural details of the design.

A. Overview of Accelerator Application

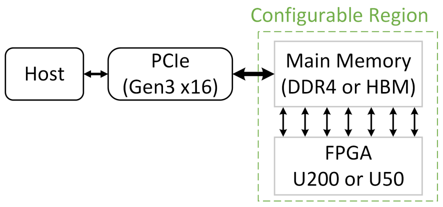


Fig. 3. Overview of the proposed architecture. The host communicates with an FPGA through the PCIe interface to send and receive computation data.

Fig. 3 depicts the high-level architecture of the FAXID. It consists of the host, main memory, FPGA, and PCIe switch to set up the communication between the host and FPGA. The host communicates with FPGA through PCIe Gen3 x16 slots offering a data transfer rate of 8.0 GT/s. The main memory can be either DRAMs or High Bandwidth Memory (HBM) depending on the FPGA card in use, i.e., Alveo U200 has DRAMs while Alveo U50 has HBM2 [33], [34]. Both FPGAs are built on a 16-nm Xilinx Ultrascale+ architecture. The Alveo U200 and U50 have 4x 8 GB of off-chip DDR4 memory and 8 GB of HBM2, respectively.

Both FPGAs are highly suitable for memory and compute-bound applications such as XGBoost. High-level architectural choices help overcome memory-bound limitations, and an optimized accelerator helps overcome compute-bound performance. In this organization, the host performs pre-computation tasks such as reading the model file and parsing the necessary information, followed by creating memory buffers, transferring data to the main memory. Once the setup is complete, the host invokes OpenCL calls to find an appropriate FPGA device, programs it with the FPGA's binary file, and transfers unknown samples for computation. When the predictions are received post-FPGA computation, we validate predictions with golden results concomitant by CPU and GPU results.

B. Architecture of Single CU

In this subsection, we detail the internal architecture of a single CU for XGBoost inference. Our proposed novel architecture is highly optimized with a primary focus on high performance. The CU, in total, performs six tasks.

As shown in Fig. 4, the first task is to initialize control registers from the given inputs such as number of trees, max nodes on a single tree, number of samples, and number of features per sample. The second task is to fetch input samples through the M-AXI interface exploiting burst access to gain maximum throughput. For the burst access, FIFOs are implemented as BRAMs. A single CU can load up to 125 samples including their features. The Alveo cards offer various SRAM options such as Ultra RAMs (URAM), Block RAMs (BRAM), LookUp Table RAMs (LUTRAM), and registers to synthesize a memory. We synthesize the input samples as URAMs as they offer the highest amount of storage on FPGAs. These URAMs for the proposed architecture are *ram_1wnr*, A RAM with 1 write port and N read ports using N banks internally. With *ram_1wnr*, the design can read multiple features allowing multiple comparisons in parallel without stalling the pipeline.

Once the input samples are stored, the first tree of the stored model in the main memory is loaded as a third task. The *ram_1wnr* type of LUTRAM is used to store a tree. We use LUTRAMs as they are closest to the compute logic blocks offering lower access time compared to URAMs or BRAMs. Similar to input samples, to read multiple values from the model, *ram_1wnr* type of memory is synthesized.

During the fourth task, the logic controller initiates comparisons between unknown samples and the model to collect the leaf values. The operation of tree traversal is similar to the process explained in Section 2. We store only one tree at a time and cache leaf values in the scratch memory. Once the comparisons are complete for all the samples with a first tree, the leaf value for each sample is stored in the scratch memory synthesized as LUTRAM with a type of *ram_2p* allowing writes to scratch memory and reads to sigmoid function simultaneously. In the second round, *XGBoost Tree* design block loads a second tree for comparisons, and leaf values get accumulated with the first round of values. The process keeps repeating until comparison with all trees is complete and the identified leaf values are added together for each sample. In addition, we restrict a maximum depth of 6 to save resources on FPGAs allowing us to map multiple CUs of the design. To expiate for the limited maximum depth, we offer no reservation on a number of trees. Additionally, we design six units of tree traversal logic in parallel to boost the

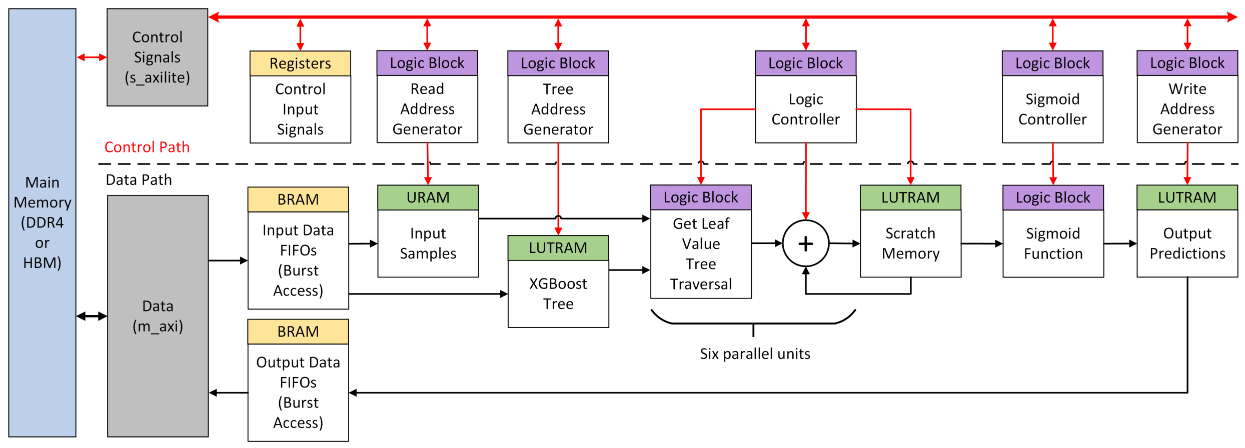


Fig. 4. The architecture of a single CU. The CU predicts the unknown samples in a total of six steps as soon as the host invokes a computation command.

performance of the proposed accelerator. Thus, the synthesis of the *ram_1wnr* for inputs and model complement performance of the six parallel units.

For large models with hundreds of trees, it is impractical to store the entire model on the local FPGA memories due to the meager available resources. When fetching data from main memory to local memory, it causes a significant clock cycle penalty. It takes around 70 clock cycles to transfer a single node of a tree, as communication with the main memory is quite costly in terms of clock cycles. To save resources on FPGA, our design can handle a maximum depth of 6 (128 nodes) per tree. Should only one sample be processed, it will cause a dramatic delay in predictions with shallow performance. To compensate for such high delays from main memory to local memories, we fetch 125 samples at a time and process them together to share the delay. After all 125 samples are processed, a new tree is loaded and these steps are repeated until all the trees are processed. To put them in a perspective, the cost, 70 clock cycles are averaged between 125 samples providing only 0.56 clock cycle delay per sample. Thus, although one tree is loaded at a time, a unique batch processing method provides high performance.

As a fifth task, the CU converts the accumulated leaf values into final probabilities with the sigmoid function. The final probabilities, post-sigmoid function, are floating-point values between 0 and 1. A comparator classifies final probabilities into 0s or 1s. Furthermore, if the final probability is greater than or equal to 0.5, the classification is 1, or 0 otherwise. The sixth task writes back the predicted binary classification values from *Output Predictions* design block to main memory through the M_AXI interface in burst access fashion. The *Output Predictions* memory is synthesized as LUTRAM with a type of *ram_1p* as design writes all sigmoid function values and then initiates the sixth task. Finally, the predicted values are sent back to the host through the PCIe interface.

C. Multiple CUs

In this subsection, we explain the creation and mapping layout of multiple CUs of the accelerator to improve the overall throughput of the proposed architecture.

An accelerator may not consume all the available resources of the target FPGA which opens doors to improve the throughput significantly by creating multiple copies of the same design explained in Section 4.2 to keep up with the data center computation requirements. We consider Xilinx's timing closure considerations guidelines to keep the resource usage under the recommended limits. The design performance requirements and timing closure can vary depending on the target. The tools spend extremely high time and effort when the design requirements are ambitious and/or more than recommended resources are consumed [8].

The tools map multiple CUs followed by place and route (P&R) either automatically or through manual assignments. We assign them manually and also make connections to the main memory using the linker configuration file in Vitis. Both FPGAs, Alveo U200 and U50 have 12 CUs. Each CU has its own private SRAMs with capabilities to communicate in parallel with the main memory. With private SRAMs on each CU, our design offers inference of multiple problems solving at the same time. Additionally, since each CUs receives individual inputs, assigning a number of CUs to a problem is flexible. The scheduler code on the host assigns equal CUs to each problem for simplicity in the case of multiple problem-solving. However, due to the flexibility, unequal CU assignment is also possible for latency constraint problems.

D. Customized Decision Node Format & Memory Layout

The structure of nodes becomes exceedingly important as it directly affects performance and resource usage. In our design, a single node takes 16 bytes to store all the entailed information. Moreover, with a maximum depth of 6, a tree has a maximum of 128 nodes requiring 128 $\times$ 16 bytes = 2 KB per tree.

A node is bifurcated into a root-or-intermediate node and a leaf node for categorization. The deliberate categorization is inevitable as there needs to be a way in hardware to identify whether the current node is a leaf node or not which is an essential step during tree traversal. Fig. 5 depicts the structure

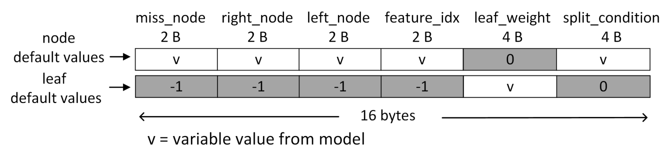


Fig. 5. A customized tree node structure is suitable for any classification problem. For any node, grey color boxes represent default values and the variable values are obtained from the model.

of both types of nodes. Moreover, on the top, names of the individual values and bytes required to store the values are displayed. The grey color boxes are the default values and the other values are variable values obtained from the trained model. A node typically needs six data types explained below:

- `split_condition`: Used to compare with input at any root or intermediate node in *float32*
- `leaf_weight`: Used to consider a predicted value by a particular tree in *float32*
- `feature_idx`: Used to identify if it is a leaf node or not. It also assists in determining which feature's split_condition is used to compare in *short int*
- `left_node`: Used to identify the left_node id of the current node in *short int*
- `right_node`: Used to identify the right_node id of the current node in *short int*
- `miss_node`: Used to travel to next node from the current node if the current input has a missing value in *short int*

V. Experimental Results

This section is divided into four subsections: 1) Datasets & Preparation, 2) Training, 3) Experimental Setup, and 4) Experimental Evaluation.

A. Datasets & Preparation

We collect a variety of datasets containing different problems. We, in total, gather four datasets from which one dataset is Hewlett Packard Enterprise (HPE) proprietary - the primary problem to be accelerated. The other three public datasets are picked to diversify the benchmarking. Table I showcases the employed datasets with a number of features and the datatype of each dataset. It also describes total samples' label distribution. Using these datasets, we build representative models that classify predicted values as 0s or 1s.

TABLE I
Dataset Collection

Dataset	0s	1s	Total	Features	Datatype
Fraud Detection [35]	15030	5438	20468	112	FP
Madeline [36]	1561	1579	3140	259	Integer
HPE	442	436	878	300	FP
Ransomware [37]	942	904	1846	234	Binary

Data pre-processing is an integral step before training and inference as the quality of data directly affects the model's ability to learn during training and the prediction scores post-inference. We pre-process dataset and apply standardization, one-hot encoding, and handling null values wherever applicable. We do not make any adjustments for unbalanced datasets.

B. Training

We train models on the CPU and GPU with different no. of trees. Our train-and-test split randomly bifurcates dataset into 70% and 30% for training and testing, respectively. The tuning parameters for the XGBoost models are: booster=gbtree, min_samples_split=2, n_jobs=12, gamma=0, colsample_bytree=1, colsample_bylevel=1, tree_method=exact, base_score=0.5, max_delta_step=0, learning_rate=0.3, and importance_type=gain. Furthermore, we train models with different n_estimators and max_depth.

With the grid search method, we get the adequate values for n_estimators and max_depth offering the optimum accuracy for a particular dataset. All the models are stored to avoid re-training every time inference is conducted. Additionally, in each inference iteration, input samples are randomly selected to avoid caching on CPUs and GPUs.

C. Experimental Setup

For the experimental setup and hardware benchmarking, we prepare three different hardware-based inferences: 1) CPU, 2) GPU, and 3) FPGA. We availed AWS EC2 instances for CPU and GPU-based inferences. The FPGA inference is our proposed accelerator.

The CPU instance on AWS EC2 has a 2.3 GHz (base) and 2.7 GHz (turbo) Intel Xeon E5-2686 v4 (Broadwell) processor utilizing 12 cores. On the software side, we borrowed the latest version of the open-source platform XGBoost (v1.4.2) to train and classify the predictions.

The G4 instance on AWS EC2 has Nvidia Tensor Core T4 GPU. We set up the latest version of the RAPIDS AI (v21.06) library-based environment to perform GPU-based XGBoost training and inference. The RAPIDS AI is a highly optimized open-source library developed by Nvidia for GPUs to train and test on ML-based algorithms [38]. RAPIDS AI offers a faster and efficient framework to efficiently use GPU architecture and run XGBoost on GPUs. Furthermore, we deploy CPU-trained models on FPGAs to run inference.

The proposed accelerator is designed with Vitis HLS tools (v2020.2). It offers a variety of tools underneath for application development, synthesis, optimization, P&R, and performance analysis. The CUs are designed purely in C++ and the host application consists of C++ code and OpenCL calls. The code performs pre-computing tasks, post-computing validation, and performance measurement. The OpenCL calls manage communication between the host and FPGA to find and program the device, create memory buffers, transfer data, task enqueuing, and receive output predictions back.

To keep the benchmarks reasonable, we choose the best model accuracy by SK-Learn's grid search method [39], [40]. We let the framework find the best model with the highest accuracy by using hyperparameter input values of maximum depth and number of trees. The max_depth values are 4 and 5, and the numbers of trees are 100, 200, 300, 400, and 500. We randomly create batches of input samples to be predicted. The sizes of batches are 1000 and 10000. Depending on the problem requirements, it may be practicable to exploit CUs to solve multiple problems at the same time. We also collect multiplex-binary classification benchmark results provided in the next subsection.

D. Experimental Evaluation

In this subsection, we collect heuristic and cumulative benchmark results. We compare the proposed accelerator against state-of-art CPU and GPU implementations as well as FPGA-based existing works where feasible. Additionally, we share prediction scores, power consumption, and resource utilization of both FPGAs.

Performance Measurement: Vitis Analyzer reports operating frequency of the two designs are 341 MHz and 344 MHz for U200 and U50, respectively. Ref. [41] suggests that HBMs provide lower latency and higher throughput compared to DDRs due to the novel memory architecture and the fact that HBMs are closer to the CUs. In addition, the HBMs tend to be more helpful for memory-bound applications. To study the effects of the employed main memory, we deploy the same design on both FPGAs and, inescapably, only change the main memory connections. We use standard datatype *float* for the input samples to handle any types of inputs as long as they are within the range of *float32*. The usage of the *float* makes architecture robust against cyber threats such as ransomware. During the designing of the proposed accelerator, we first design *float16*-based CUs. However, we notice significant accuracy drop down to approx. 55%. Furthermore, it may be arduous to find input values' range, especially, when adversarial sophisticated attacks are evolving periodically.

Table II demonstrates the FPGA accelerator's speedup over CPU and GPU. For 1000 samples in a batch, the FPGA accelerator can outperform CPU within the range of 16x to 65.8x times and GPU within the range of 2.5x to 4.1x times for the presented datasets in terms of latency. Similarly, when the batch size is 10000, the speedup-spans between 28.2x to 52.1x and 2.5x to 5.25x over CPU and GPU, respectively.

With the publicly available market price of the hardware, we calculate IOPS/Cost to delineate the cost-effectiveness of predicting a number of samples per dollar. Hence, for (Kio/s)/US$ in Table II, the higher value represents cost efficiency. Against CPU and GPUs, both FPGAs tend to be quite cost-efficient. In particular, Alveo U50 is 21.3x cost-effective over the CPU and 4x economical over GPU. Compared to Alveo U200, the Alveo U50 is 1.8x better for cost-effectiveness.

Energy Consumption: We measure the power of the proposed design and compare it with the GPU followed by energy consumption ($power \times latency$) with performance analysis where lower is better in this case. The CPU has a TDP of 145W. The maximum power envelop of the Nvidia Tensor core T4 GPU is listed as 70W. The maximum total power of U50 and U200 is 75W and 225W, respectively.

The average power consumption of XGBoost inference on GPU is found to be 27.7W. The design on U50 consumes 14.3W and the U200 consumes 10W. Table II presents the calculated energy consumption from the reported power and latency. The proposed design on U200 is 13.7x times efficient than GPU. For energy-efficiency demands, the U200 is 1.4x efficient compared to Alveo U50.

Prediction Scores: In standard ML practices relying only on accuracy is not adequate. It is important to dive into finer details to observe what the classifier is analytically imparting. For example, ransomware predictions require extra attention to false-negative cases. Should they unknowingly cross the data center security systems, can cause perilous events. Thus, to holistically analyze classifiers' behavior, we measure the confusion matrix (true positive, true negative, false positive, and false negative) of the datasets. We quantify precision, recall, F1, and cohen score from the confusion matrix in addition to the reported accuracy. Our design provides a significant performance boost over CPU as well as GPU without sacrificing prediction scores as shown in Table III.

Resource Consumption: Table IV indicates the corresponding resource consumption for the implemented accelerators on U50 and U200 with 12 CUs. The reported data also includes resource utilization of AXI (m_axi and s_axilite) and memory controllers. With quite the area remaining on the FPGAs, more CUs can be mapped on the device if necessary. Increasing the CUs implicates a tad higher latency, power, throughput, and IOPS. It is another way to provide more resources to multiple problems. Similarly, reducing the CUs would help achieve low latency and power requirements problems affecting throughput and IOPS negatively. Thus, we leave such critical decisions to designers' discretion adhering to design requirements.

Multiplex Problem Solving: The proposed design is the first design, the authors are aware of, to address multiple problems classification in parallel. A *Problem 1* with 1000 samples in a batch requires a total of 8 CUs with each CU processing 125 samples at a time. Our proposed design on both FPGAs consists of 12 CUs. To effectively utilize all CUs, the design can attempt to solve multiple problems. Hence, the other 4 ideal CUs can be adroitly utilized to classify *Problem 2*. Moreover, a scheduler can assign a problem to ideal CUs while others are still classifying the designated problem. Depending on the project requirements, designers may write a scheduling algorithm.

For multiplex problems solving, a problem can consume any number of assigned CUs based on the scheduler ranging between 1 to 12. In this paper, for simplicity, all the problems share equal CUs. However, since each CU individually receives the inputs, the proposed design can assign unequal CUs to multiple problems if latency is a primary concern. The CPU and GPU do not have that support and predictions for all problems run sequentially. As denoted in Table V, for four problems with 10000 samples each is the only case where GPU outperforms FPGAs in terms of latency, however, throughput is approx. 7x lower than FPGAs. Also, in Table V,

TABLE II
PERFORMANCE METRICS OF A SINGLE PROBLEM ON CPU, GPU, AND FPGAS. THE CALCULATED CPU VALUES FOR IOPS/POWER AND ENERGY USE 145W AS THE POWER CONSUMPTION VALUES COULD NOT BE ESTIMATED DUE TO THE AWS SECURITY RESTRICTIONS ON THE HARDWARE INSTANCE.

Dataset	No. of Samples	Hardware Inference	Latency (ms)	Throughput (TB/s)	IOPS (Mio/s)	Cost (US$)	IOPS/Cost (Kio/s)/US$	IOPS/Power (Kio/s)/W	Energy(mJ) ($W \times ms$)
FD 200 Trees	1000	CPU	8.64	0.19	0.23	1999	0.12	2	1252.8
		GPU	1.97	0.83	1.02	2199	0.46	37	54.6
		U50	0.54	3.01	3.7	2879	1.29	258	7.7
		U200	0.6	2.7	3.31	4825	0.69	331	6
FD 200 Trees	10000	CPU	64.2	2.54	0.31	1999	0.16	2	9309
		GPU	5.71	28.53	3.5	2199	1.59	126	158.17
		U50	2.27	71.72	8.8	2879	3.06	615	32.5
		U200	2.34	69.67	8.55	4825	1.77	855	23.4
Madeline 300 Trees	1000	CPU	14.99	0.25	0.13	1999	0.07	1	2173.5
		GPU	2.19	2	0.92	2199	0.42	33	60.7
		U50	0.75	5.05	2.68	2879	0.93	187	10.7
		U200	0.78	4.85	2.57	4825	0.53	257	7.8
Madeline 300 Trees	10000	CPU	115.37	3.27	0.17	1999	0.09	1	16729
		GPU	12.42	35.14	1.61	2199	0.73	58	344
		U50	3.61	104.3	5.54	2879	1.92	387	51.6
		U200	3.99	94.61	5.02	4825	1.04	502	39.9
Ransomware 100 Trees	1000	CPU	33.57	0.1	0.06	1999	0.03	0.4	4868
		GPU	2.08	1.63	0.96	2199	0.44	35	57.6
		U50	0.51	6.63	3.91	2879	1.36	273	7.3
		U200	0.51	6.71	3.96	4825	0.82	396	5.1
Ransomware 100 Trees	10000	CPU	104.67	3.24	0.19	1999	0.10	1	15177
		GPU	10.57	32.09	1.89	2199	0.86	68	292.8
		U50	2.01	168.6	9.94	2879	3.45	695	28.7
		U200	2.14	158.3	9.34	4825	1.94	934	21.4
HPE 400 Trees	1000	CPU	15.97	0.27	0.13	1999	0.07	1	2315.6
		GPU	2.19	1.99	0.91	2199	0.41	33	60.7
		U50	0.87	5	2.29	2879	0.80	160	12.4
		U200	0.94	4.67	2.14	4825	0.44	214	9.4
HPE 400 Trees	10000	CPU	156.49	2.79	0.13	1999	0.07	1	22691
		GPU	12.32	35.45	1.62	2199	0.74	58	341.3
		U50	4.49	97.24	4.46	2879	1.55	312	64.2
		U200	4.7	92.82	4.25	4825	0.88	425	47

TABLE III
PREDICTION SCORES COMPARISONS. THE ACCELERATOR BOOSTS PERFORMANCE WITHOUT SACRIFICING PREDICTION SCORES.

Dataset	Number of Trees	Test Samples(30%)	Hardware	Accuracy	Precision	Recall	F1	Cohen
Fraud Detection	200	4914	CPU	93.94	0.90	0.87	0.88	0.84
			GPU	93.94	0.90	0.87	0.88	0.84
			U50	93.51	0.87	0.89	0.88	0.83
			U200	93.51	0.87	0.89	0.88	0.83
Madeline	300	787	CPU	81.70	0.82	0.83	0.82	0.63
			GPU	81.70	0.82	0.83	0.82	0.63
			U50	81.83	0.80	0.86	0.83	0.64
			U200	81.83	0.80	0.86	0.83	0.64
Ransomware	100	554	CPU	99.55	1	1	1	0.99
			GPU	99.55	1	1	1	0.99
			U50	99.52	1	0.99	1	0.99
			U200	99.52	1	0.99	1	0.99
HPE	400	878	CPU	78.36	0.77	0.81	0.79	0.57
			GPU	78.36	0.77	0.81	0.79	0.57
			U50	78.31	0.76	0.83	0.79	0.56
			U200	78.31	0.76	0.83	0.79	0.56

TABLE IV
RESOURCE UTILIZATION. DUE TO THE HBM AND DDR, BOTH FPGAS HAVE DIFFERENT RESOURCE CONSUMPTION AS IT ALSO INCLUDES AXI AND MEMORY CONTROLLERS DESPITE HAVING THE SAME NUMBER OF CUS ON EACH.

Device	LUT(Logic)	LUT(Memory)	Registers	BRAM	URAM	DSP	PLL	MMCM
U50	196006(22.53%)	47905(11.92%)	362294(20.82%)	984(73.21%)	480(75.00%)	136(2.29%)	1(6.25%)	3(37.50%)
U200	391212(33.09%)	102574(17.33%)	755479(31.95%)	1740.5(80.58%)	480(50.00%)	148(2.16%)	16(26.67%)	6(20.00%)

for two problems, HPE and Madeline both problems would be assigned six CUs each. Both classifying 1000 or 10000 samples each makes a total of 2000 or 20000. The rest of the multiplex problems also follow the same principle.

TABLE V
PERFORMANCE MEASURES OF MULTIPLE PROBLEMS

No. of Problems	Dataset	No. of Trees	Total Samples per Batch	Hardware Inference	Latency (ms)	Throughput (TB/s)	IOPS (Mio/s)	Cost (US$)	IOPS/Cost (Kio/s)/US$
2	HPE	400	2000	CPU	27.67	0.27	0.14	1999	0.07
	Madeline	300		GPU	2.83	2.66	1.41	2199	0.64
				U50	1.43	11.38	2.80	2879	0.97
				U200	1.44	11.25	2.77	4825	0.57
2	HPE	400	20000	CPU	237.19	3.18	0.17	1999	0.09
	Madeline	300		GPU	17.26	4.66	2.32	2199	1.06
				U50	8.44	192.8	4.74	2879	1.65
				U200	8.79	185.2	4.55	4825	0.94
3	HPE	400	3000	CPU	29.22	0.35	0.21	1999	0.11
	Madeline	300		GPU	3.23	3.15	1.86	2199	0.86
	Ransomware	100		U50	1.36	25.51	4.43	2879	1.54
				U200	1.38	25.04	4.35	4825	0.90
3	HPE	400	30000	CPU	221.33	4.60	0.27	1999	0.14
	Madeline	300		GPU	20.62	49.34	2.91	2199	1.32
	Ransomware	100		U50	11.96	289	5.02	2879	1.74
				U200	12.32	280.6	4.87	4825	1.01
4	HPE	400	4000	CPU	40.35	0.16	0.20	1999	0.10
	Madeline	300		GPU	3.68	1.77	2.17	2199	0.99
	Ransomware	100		U50	1.944	27.07	4.12	2879	1.43
	Fraud Detection	200		U200	1.95	27.04	4.11	4825	0.85
4	HPE	400	40000	CPU	273.17	2.39	0.29	1999	0.15
	Madeline	300		**GPU**	**13.86**	**47.05**	5.77	2199	2.62
	Ransomware	100		U50	15.98	329.2	5.01	2879	1.74
	Fraud Detection	200		U200	16.36	321.8	4.98	4825	1.03

Comparison with other FPGA Designs: Many of the existing works do not have DDRs or HBMs and they synthesize the design based on the available resources on FPGA. The operating frequency and power consumption is directly related to silicon technology. For example, Alveo cards use 16-nm technology while FPGA in Ref. [4] is built on 28-nm technology. Moreover, the application-specific work may have optimizations that may not be feasible to generalized accelerators such as FAXID. In pixel classification, input values' range can be determined but against threat detection, it may not be a feasible approach [13]. Moreover, for small problems, the number of trees can be synthesized directly on SRAMs which the data center-level complex models may not have the luxury of.

To make fair a comparison, we compare the clock cycle latency of predicting one sample. As the proposed architecture is customized and designed to handle multiple sample processing, we take an average of total clock cycles. As single CU processes 125 samples, we take an average clock cycle latency of 125 samples excluding main memory access clock cycles and compare them with other works wherever possible. Our design takes an average of 7.1 clock cycles to predict one sample. In Ref. [14], maximum reported clock cycle latency is 13. A total of 12 clock cycles are report for similar scenario in Ref. [5], [25], [29]. In Ref. [4], a datapath pipeline is 8 clock cycles deep. Hence, a minimum of 8 clock cycles are required in addition to the clock cycles consumed by the final compute step *Reducer*.

VI. CONCLUSION

This paper presents a novel XGBoost accelerator for enterprise-level data workload offering high performance. The proposed design is energy-efficient as well as cost-effective, predominantly offering multiplex problem classification in parallel. With scalable architecture and available resources on FPGA, different timing and requirement constraint designs can be rapidly generated. We extensively benchmark a variety of measures with diverse real-life datasets. We exhibit speedup of up to x65.8 times over CPU and x4.1 over GPU with 1000 samples. With 10000 samples, FPGA is x52.1 faster than CPU and x5.3 faster than GPU. With almost the same performances, we showcase a cost-effective approach on U50 and an energy-efficient approach on U200.

ACKNOWLEDGMENT

We are thankful to the anonymous reviewers for their valuable suggestions to help improve the manuscript. This material is based upon work supported by the National Science Foundation under Grant No. CNS #16-244770 - Center for Advanced Electronics through Machine Learning (CAEML) and its industry members. We appreciate the generous donation of the hardware and/or AWS resources from Xilinx and HPE.

REFERENCES

[1] O. M. Alhawi, J. Baldwin, and A. Dehghantanha, "Leveraging machine learning techniques for windows ransomware network traffic detection," in *Cyber threat intelligence*. Springer, 2018, pp. 93–106.
[2] Y. Mishina, R. Murata, Y. Yamauchi, T. Yamashita, and H. Fujiyoshi, "Boosted random forest," *IEICE TRANSACTIONS on Information and Systems*, vol. 98, no. 9, pp. 1630–1636, 2015.
[3] M. Owaida and G. Alonso, "Application partitioning on fpga clusters: inference over decision tree ensembles," in *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2018, pp. 295–2955.
[4] M. Owaida, H. Zhang, C. Zhang, and G. Alonso, "Scalable inference of decision tree ensembles: Flexible design for cpu-fpga platforms," in *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2017, pp. 1–8.

[5] M. Elnawawy, A. Sagahyroon, and T. Shanableh, “Fpga-based network traffic classification using machine learning,” *IEEE Access*, vol. 8, pp. 175 637–175 650, 2020.

[6] N. Scaife, H. Carter, P. Traynor, and K. R. Butler, “Cryptolock (and drop it): stopping ransomware attacks on user data,” in *2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2016, pp. 303–312.

[7] M. Owaida, A. Kulkarni, and G. Alonso, “Distributed inference over decision tree ensembles on clusters of fpgas,” *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 12, no. 4, pp. 1–27, 2019.

[8] *Vitis Unified Software Development Platform 2020.2 Documentation*, Xilinx, March 2021, https://www.xilinx.com/html_docs/xilinx2020_2/vitis_doc/index.html.

[9] L. Breiman, “Random forests,” *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[10] J. H. Friedman, “Greedy function approximation: a gradient boosting machine,” *Annals of statistics*, pp. 1189–1232, 2001.

[11] T. Chen and C. Guestrin, “Xgboost: A scalable tree boosting system,” in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 2016, pp. 785–794.

[12] R. Narayanan, D. Honbo, G. Memik, A. Choudhary, and J. Zambreno, “An fpga implementation of decision tree classification,” in *2007 Design, Automation & Test in Europe Conference & Exhibition*. IEEE, 2007, pp. 1–6.

[13] A. Alcolea and J. Resano, “Fpga accelerator for gradient boosting decision trees,” *Electronics*, vol. 10, no. 3, p. 314, 2021.

[14] F. Saqib, A. Dutta, J. Plusquellic, P. Ortiz, and M. S. Pattichis, “Pipelined decision tree classification accelerator implementation in fpga (dt-caif),” *IEEE Transactions on Computers*, vol. 64, no. 1, pp. 280–285, 2013.

[15] B. Van Essen, C. Macaraeg, M. Gokhale, and R. Prenger, “Accelerating a random forest classifier: Multi-core, gp-gpu, or fpga?” in *2012 IEEE 20th International Symposium on Field-Programmable Custom Computing Machines*. IEEE, 2012, pp. 232–239.

[16] H. Jiang, Z. He, G. Ye, and H. Zhang, “Network intrusion detection based on pso-xgboost model,” *IEEE Access*, vol. 8, pp. 58 392–58 401, 2020.

[17] C. Wang, C. Deng, and S. Wang, “Imbalance-xgboost: leveraging weighted and focal losses for binary label-imbalanced classification with xgboost,” *Pattern Recognition Letters*, vol. 136, pp. 190–197, 2020.

[18] S. S. Dhaliwal, A.-A. Nahid, and R. Abbas, “Effective intrusion detection system using xgboost,” *Information*, vol. 9, no. 7, p. 149, 2018.

[19] M. Barbareschi, S. Del Prete, F. Gargiulo, A. Mazzeo, and C. Sansone, “Decision tree-based multiple classifier systems: An fpga perspective,” in *International Workshop on Multiple Classifier Systems*. Springer, 2015, pp. 194–205.

[20] T. Sadasue, T. Tanaka, R. Kasahara, A. Darmawan, and T. Isshiki, “Scalable hardware architecture for fast gradient boosted tree training,” *IPSJ Transactions on System LSI Design Methodology*, vol. 14, pp. 11–20, 2021.

[21] T. Tanaka, R. Kasahara, and D. Kobayashi, “Efficient logic architecture in training gradient boosting decision tree for high-performance and edge computing,” *arXiv preprint arXiv:1812.08295*, 2018.

[22] C. Cheng and C.-S. Bouganis, “Accelerating random forest training process using fpga,” in *2013 23rd International Conference on Field programmable Logic and Applications*. IEEE, 2013, pp. 1–7.

[23] *XGBoost Exact Updater IP core*, InAccel, July 2019, https://github.com/InAccel/xgboost.

[24] X. Lin, R. S. Blanton, and D. E. Thomas, “Random forest architectures on fpga for multiple applications,” in *Proceedings of the on Great Lakes Symposium on VLSI 2017*, 2017, pp. 415–418.

[25] S. Buschjäger and K. Morik, “Decision tree and random forest implementations for fast filtering of sensor data,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 65, no. 1, pp. 209–222, 2017.

[26] W. Song, Q. Han, Z. Lin, N. Yan, D. Luo, Y. Liao, M. Zhang, Z. Wang, X. Xie, A. Wang *et al.*, “Design of a flexible wearable smart semg recorder integrated gradient boosting decision tree based hand gesture recognition,” *IEEE transactions on biomedical circuits and systems*, vol. 13, no. 6, pp. 1563–1574, 2019.

[27] R. Kulaga and M. Gorgon, “Fpga implementation of decision trees and tree ensembles for character recognition in vivado hls,” *Image Processing & Communications*, vol. 19, no. 2-3, p. 71, 2014.

[28] D. Tong, Y. R. Qu, and V. K. Prasanna, “Accelerating decision tree based traffic classification on fpga and multicore platforms,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 11, pp. 3046–3059, 2017.

[29] S. Summers, G. Di Guglielmo, J. Duarte, P. Harris, D. Hoang, S. Jindariani, E. Kreinar, V. Loncar, J. Ngadiuba, M. Pierini *et al.*, “Fast inference of boosted decision trees in fpgas for particle physics,” *Journal of Instrumentation*, vol. 15, no. 05, p. P05026, 2020.

[30] H. Nakahara, A. Jinguji, T. Fujii, and S. Sato, “An acceleration of a random forest classification using altera sdk for opencl,” in *2016 International Conference on Field-Programmable Technology (FPT)*. IEEE, 2016, pp. 289–292.

[31] *Xelera Suite*, Xelera, May 2020, https://xelera.io/.

[32] *Xelera Decision Tree Inference*, Xelera, June 2021, https://github.com/xelera-technologies/Tree-Inference.

[33] *Alveo U200 Data Sheet*, Xilinx, May 2020, https://www.xilinx.com/support/documentation/data_sheets/ds962-u200-u250.pdf.

[34] *Alveo U50 Data Sheet*, Xilinx, August 2020, https://www.xilinx.com/support/documentation/data_sheets/ds965-u50.pdf.

[35] (2021) Fraud detection bank dataset. [Online]. Available: https://www.kaggle.com/volodymyrgavrysh/fraud-detection-bank-dataset-20k-records-binary

[36] (2018) Madeline dataset. [Online]. Available: https://www.openml.org/d/41144

[37] S. Kok, A. Abdullah, and N. Jhanjhi, “Early detection of crypto-ransomware using pre-encryption detection algorithm,” *Journal of King Saud University-Computer and Information Sciences*, 2020.

[38] *RAPIDS: Collection of Libraries for End to End GPU Data Science*, Open Source, 2018, https://rapids.ai.

[39] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, “Scikit-learn: Machine learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[40] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux, “API design for machine learning software: experiences from the scikit-learn project,” in *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, 2013, pp. 108–122.

[41] *Virtex UltraScale+ HBM FPGA: A Revolutionary Increase in Memory Performance*, Xilinx, July 2019, https://www.xilinx.com/support/documentation/white_papers/wp485-hbm.pdf.