Optimized XGBoost Architecture on FPGA for High-Performance and Seamless Deployment

Rodrigo Olmos
Centro de Electrónica Industrial
Universidad Politécnica de Madrid
Madrid, Spain
E-mail: rodrigo.olmos@upm.es

Andrés Otero
Centro de Electrónica Industrial
Universidad Politécnica de Madrid
Madrid, Spain
E-mail: joseandres.otero@upm.es

***Abstract*—This paper presents an accelerated implementation of the XGBoost algorithm optimized for FPGA platforms, designed to enhance performance in applications demanding real-time processing and high inference speeds. The proposed architecture is implemented on an FPGA and interfaces with a host CPU via PCIe. Performance is optimized through intermediate memories and a DMA subsystem for high-speed data transfer, enabling rapid access to model parameters and efficient inference execution while minimizing latency and maximizing hardware concurrency. To simplify usability, a set of automation scripts is developed to facilitate the seamless deployment of pre-trained XGBoost models, leveraging compatibility with the Python-based LightGBM library. This approach removes the need for extensive hardware expertise, enabling users to deploy FPGA-accelerated models directly from Python environments. Experimental results include a comparative analysis of inference speeds across FPGAs, GPUs, and CPUs, demonstrating significant performance gains achieved by the proposed system over traditional GPU- and CPU-based solutions. These gains result from speedups of 200 to 400 times compared to the same implementation running on a CPU or GPU. This efficient design establishes a robust framework for expanding parallel processing capabilities to more complex models and integrating advanced machine learning solutions into embedded computing environments.**

I. Introduction

High-speed deep learning (DL) inference is crucial for many applications driven by artificial intelligence (AI), such as large-scale data analysis, pattern recognition in security systems, speech processing with large language models (LLMs), and automated medical diagnosis. However, the increasing complexity of machine learning models—such as those based on large decision trees—poses significant performance and energy efficiency challenges when deployed on conventional hardware like CPUs (Central Processing Units) and GPUs (Graphics Processing Units). Field-Programmable Gate Arrays (FPGAs) have emerged as a promising alternative in this context.

FPGAs offer notable advantages, including high flexibility and the ability to provide custom acceleration that significantly improves system latency and throughput, all while maintaining excellent energy efficiency [4]. Traditionally favoured for edge applications, the availability of FPGA-based platforms designed for PCIe streaming (e.g., Alveo Platforms) [12] extends the benefits of FPGAs to the cloud domain, further broadening their potential use cases.

This work presents a software/hardware architecture for accelerating inference on XGBoost models [13]. XGBoost is a model based on Boosting techniques in which multiple decision trees are combined to improve the accuracy and robustness of the model in classification and regression tasks. Boosting is a machine learning (ML) strategy that sequentially adds models, with each model attempting to correct the errors of the previous one. XGBoost has gained significant popularity due to its efficiency and accuracy, making it widely used in applications that require fast, interpretable models.

One of the most notable features of decision trees is their interpretable structure, allowing for easy visualization and understanding of the model's decision-making process. However, decision trees often present significant computational complexity when implemented in high-demand, real-time applications. Combining multiple trees in ensemble methods, such as XGBoost, further increases this complexity, so parallelization becomes essential for enhancing performance. For this reason, decision trees and their ensemble variants benefit considerably from hardware acceleration, especially in FPGA implementations, which enable parallel and low-latency processing in inference tasks.

The architecture proposed in this paper has been designed and optimized for FPGA platforms with PCIe streaming capabilities. This design allows for real-time inference by providing fast access to model parameters, leveraging high-speed memory modules and a DMA transfer system that minimizes latency while maximizing concurrency [2]. Additionally, architecture-level optimization techniques are implemented to enable parallel processing of the trees, maximizing throughput and reducing bottlenecks typically encountered in inference with boosting models accelerated on hardware. This solution offers a remarkable advantage in real-time inference applications, handling large data volumes with low latency and high energy efficiency. It is a unique and competitive alternative in hardware-accelerated machine learning. The architecture has been described using Vitis HLS, demonstrating the possibilities enabled by high-level synthesis tools to deploy hardware-accelerated AI systems.

In addition to the performance improvements enabled by

hardware acceleration, this work focuses on enhancing accessibility to hardware resources. A Python-based scripting tool is provided with this aim, allowing users to deploy automatically previously trained with the LightGBM Python-based XGBoost library [9] onto hardware accelerators, hiding the technical details of FPGA architecture [8]. Furthermore, custom drivers have been developed to ensure seamless integration between the user's application and the hardware-accelerated models, making the implementation process almost transparent to the user. Overall, the proposed system empowers users with limited or no hardware expertise to leverage FPGA acceleration effectively. This approach simplifies the workflow and optimizes deployment for real-time applications.

Experimental results demonstrate that the proposed FPGA architecture significantly outperforms conventional GPU and CPU implementations manually described in OpenCL and C, respectively, for four open datasets from the Kaggle AI & ML community [10], particularly regarding inference speed. These benchmarks consist of executing different models and measuring the inference time. The comparison among these systems shows that using FPGAs improves the execution efficiency of XGBoost models, especially in applications where latency and real-time processing are critical [1].

The remainder of this paper is structured as follows. Section 2 provides an overview of decision trees and the XGBoost algorithm, including related work in hardware accelerators for these models. Section 3 details the proposed design methodology. Section 4 presents the FPGA architecture, focusing on the system's modules. Section 5 discusses the experimental setup and the results obtained, comparing the performance of the FPGA-based implementation with CPU and GPU alternatives. Finally, Section 6 concludes the paper with insights into potential future enhancements to expand the applicability and efficiency of the proposed system.

II. Background

A. Decision trees and XGBoost

Decision trees are machine learning models widely used in classification and regression tasks due to their interpretative simplicity and ability to handle numerical and categorical data [11]. A decision tree is a hierarchical structure consisting of decision and leaf nodes, as shown in Figure 1. At each decision node, a test is performed by comparing a specific dataset feature with a parameter discovered during the training procedure. A feature is an individual measurable property or attribute of the data. This process divides the data space into smaller, more homogeneous subspaces, progressively refining the classification or regression task by grouping similar data points. The process continues through multiple levels of the tree until reaching the leaf nodes where the model's final predictions are represented, either a class in classification or a numerical value in regression [11]. Training a decision tree involves iteratively selecting each node's optimal features and decision thresholds to minimize prediction error.

A common strategy to enhance the model's accuracy and robustness is to combine multiple trees using an ensemble approach. Algorithms like CART (Classification and Regression Trees) and XGBoost (eXtreme Gradient Boosting) are based on this technique. This work focuses on XGBoost due to its broader adoption in industry and academia.

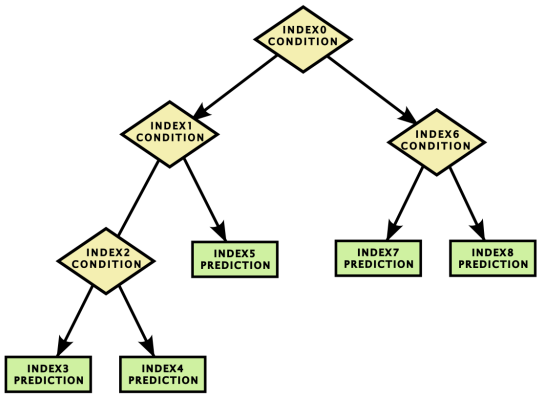

Fig. 1. Structure of a decision tree

XGBoost is a specialized decision tree algorithm that leverages the boosting method [13]. Boosting is an ensemble technique that combines multiple weak models—such as decision trees—to create a robust and accurate predictor [13], as shown in Figure 2. In XGBoost, each new tree is trained to correct the errors made by previous trees, resulting in a cumulative model that iteratively reduces prediction errors [9]. This approach enhances the accuracy and efficiency of predictive models, particularly for classification and regression tasks.

One main feature that positions XGBoost above other boosting methods is its focus on computational efficiency and its ability to handle large volumes of data. Additionally, XGBoost incorporates regularization into its learning process, which helps reduce overfitting and improves the model's generalization to unseen data. These qualities have made XGBoost popular in applications requiring high precision and performance, such as fraud detection [14], financial data analysis [16], and medical diagnosis [15].

However, due to its complex structure based on multiple decision trees, XGBoost demands substantial computational resources, especially to achieve real-time inference. Implementing XGBoost in hardware, such as on FPGA architectures, addresses these challenges by accelerating computations and enabling parallel processing. This makes the algorithm suitable for environments where speed and low latency are essential.

B. Previous Works on Hardware Accelerators for XGBoost

Different contributions to the state-of-the-art address the acceleration of XGBoost in hardware, particularly on FPGAs. Some of the most relevant works are described next.

Kanani et al. proposed LightFPGA [1], a library that automates the conversion of pre-trained LightGBM models into Verilog descriptions. This solution enables hardware implementations that deliver improvements in latency, achieving speeds 100 to 400 times faster than CPU counterparts and

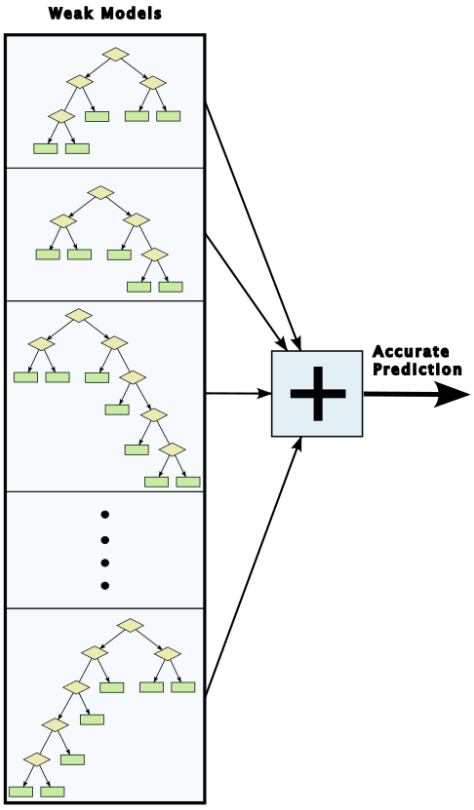

Fig. 2. Combination of XGBoost decisions trees

reducing energy consumption by 7 to 8 times. The tool enables quantization without loss of precision and automatically generates testbenches for validation, marking a significant milestone in simplifying the deployment of models on FPGAs.

Alternatively, works by Nane et al. [3] and Gajjar et al. [4] focus on accelerating XGBoost using High-Level Synthesis (HLS) tools. These works highlight the efficiency in data distribution and inter-FPGA communication, achieving superior performance compared to traditional software-based solutions. Moreover, they present a comprehensive comparative analysis of academic and commercial HLS tools, emphasizing the robustness and flexibility of commercial tools for this type of application.

Regarding specific applications, Krueger et al. [5] demonstrate how FPGAs can optimize data processing in positron emission tomography (PET) systems. Their implementation of gradient boosting trees on Kintex-7 FPGAs can process between 2.94 and 4.55 million gamma interactions per second, significantly reducing hardware resource usage while maintaining high positioning accuracy. Gamma interactions per second refers to the number of gamma-ray events a positron emission tomography system can detect and process every second. In turn, the paper by Owaida et al. [6] focuses on high data-rate applications such as particle classification in high-energy physics. Their architecture can handle up to half a million tree nodes within the FPGA's memory, providing performance improvements of up to 20 times over a 10-thread CPU and enabling hybrid CPU-FPGA processing for larger datasets.

Additionally, the works of Summers et al. [7] and Alcolea et al. [8] delve into methods for accelerating XGBoost on FPGAs. In particular, authors in [8] stand out for optimizing the execution of gradient-boosted decision trees in embedded systems, specifically for pixel classification in hyperspectral images. The authors demonstrate that their accelerator can process hyperspectral data at the same speed as the sensors generate it, achieving significant performance improvements over high-performance CPUs while consuming considerably less energy—averaging twice the speed and 72 times less energy compared to optimized software on a high-performance processor, and 30 times faster with 23 times less energy than an embedded processor. To achieve this, they employ a decision tree model encoding that is well-suited for execution on an FPGA.

Differently from the state-of-the-art, the proposal in this paper is to develop an automated system for deploying pre-trained XGBoost-based decision tree models directly on FPGA hardware. This approach includes seamless integration from model training in a software environment to its adaptation and deployment on specialized hardware, leveraging a structured process to maximize inference performance and minimize user involvement in hardware-specific tasks. The proposed system allows users to train, export, and deploy models in an FPGA-compatible format, using a pre-order depth-first search (DFS) traversal along with optimized parameter encoding for hardware execution, automating the model's configuration and loading process to enable real-time, high-speed inference on PCIe-based FPGA hardware accelerators optimized for the cloud.

III. Proposed Design Methodology

The system proposed in this work is based on decision tree models trained using the XGBoost algorithm, optimized for execution on FPGA hardware. The training and export workflow consists of several stages that ensure the proper adaptation of the model to the hardware architecture and its efficient operation in low-latency environments. The proposed design methodology is described next.

First, the model is trained with LightGBM in a software environment using Python. In this stage, the model is trained on the CPU. Afterward, the trained model's weights are sent to the FPGA, where the inference operations will be executed. LightGBM is in charge of hyperparameter tuning and model optimization through boosting techniques. During this stage, the model learns the dataset's features by constructing a set of decision trees. Each tree is trained sequentially to correct the errors of the previous trees, thus improving the model's accuracy with each iteration. The training phase also includes tree regularization to prevent overfitting, which is crucial to obtaining a model that generalizes well to unseen data.

Once training is completed with LightGBM, a custom script has been designed to export the model in a hardware-

compatible format that can be executed and interpreted by an FPGA. This is achieved using a pre-order DFS algorithm. This tree traversal method starts at the root node and explores all nodes from left to right and top to bottom. This arrangement is shown in Figure 1, where the index indicated in each node refers to the position where the node is stored in the exported file. During this process, essential parameters of the decision tree are extracted, including each node's decision thresholds, feature indices, and right-node indices. Left-node indices are not stored, as they are always one unit higher than the current node's index. These parameters are structured in a predefined format: a 64-bit vector for each node that contains all the necessary information to reconstruct and interpret the previously exported tree. The organization of this bit vector is shown in Table I.

This workflow automates the generation of indices and configures the trees into the required structure, allowing the model to be directly loaded into 64-bit width on-chip memory and ready for quick access during inference [7].

TABLE I
BIT MAP FOR EACH 64-BIT WORD DESCRIBING A TREE NODE.

BITS	FIELD
63 - 57	RESERVED
56	LEAF (1) / NODE (0)
55 - 48	FEATURE INDEX
47 - 40	NEXT NODE RIGHT INDEX
39 - 31	RESERVED
31 - 0	NODE THRESHOLD / LEAF VALUE

On the application's side, a library has been designed to enable the communication between the hardware generated on the FPGA and the host PC transparently, maximizing the performance of the hardware described in this paper. This is achieved thanks to the library's ability to distribute the workload automatically.

This fully automated workflow enables users without hardware design experience to train and export XGBoost models ready for FPGA acceleration. The efficient implementation includes the hierarchical structure of decision trees and all the information necessary to perform inference optimally. This training and export approach allows for smooth integration between software and hardware. It maximizes inference performance by eliminating the need for additional transformations when loading the model onto the FPGA. As a result, the proposed system can execute real-time inferences, leveraging the parallel processing capability and low latency of FPGA architectures.

IV. PROPOSED ACCELERATION ARCHITECTURE

The proposed architecture for accelerating XGBoost models is detailed in the following subsections. First, the design and implementation of the individual tree accelerator are presented. This is followed by describing the optimized infrastructure for efficient input data transmission to the FPGA card.

A. *Tree Architecture*

The architecture of the decision tree is responsible for making predictions by traversing a tree's nodes until reaching a leaf. Data corresponding to each node is loaded from an internal memory. This data includes the 64-bit vector shown in Table I, that include the following fields:

- **An indicator** determining whether the node is a leaf or an internal node "LEAF (1) / NODE (0)".
- **The associated prediction value** (if it is a leaf) "NODE THRESHOLD / LEAF VALUE".
- **The index of the attribute (feature)** used for comparison (if it is an internal node) "FEATURE INDEX".
- **The threshold value** for the comparison (if it is an internal node) "NODE THRESHOLD / LEAF VALUE".
- **The index of the right node** to which it should jump if the comparison condition is not met (if it is an internal node) "NEXT NODE RIGHT INDEX".

The prediction process, which is graphically represented in Figure 3, is described in Algorithm 1.

Algorithm 1 Procedure to Make a Prediction

Require: Array of trees $trees[NTrees][NNodesAndLeafs]$, feature vector $features[NFeatures]$
Ensure: Prediction stored in $prediction$

$sum \leftarrow 0$
for $t \leftarrow 0$ **to** $NTrees - 1$ **do**
  $nodeIndex \leftarrow 0$
  while true do
    $treeNode \leftarrow trees[t][nodeIndex]$ ▷ Extract node data
    $featureIndex \leftarrow treeNode.featureIndex$
    $threshold \leftarrow treeNode.nodeValue$
    $nodeLeft \leftarrow nodeIndex + 1$
    $nodeRight \leftarrow treeNode.nextNodeRightIndex$
    if $features[featureIndex] < threshold$ **then**
      $nodeIndex \leftarrow nodeLeft$
    else
      $nodeIndex \leftarrow nodeRight$
    end if
    if not $(treeNode.leafOrNode \& 0x01)$ **then**
      break
    end if
  end while
  $sum \leftarrow sum + treeNode.nodeValue$
end for
$prediction \leftarrow sum$

B. *Trees Forest*

The architecture of the tree forest aggregates a large number of decision trees that execute in parallel. To achieve this level of parallelism, special care has been taken to determine which components can be parallelized—specifically, all the trees that run concurrently, as shown in Figure 4. Since the trees operate simultaneously, it is crucial to ensure they have the correct

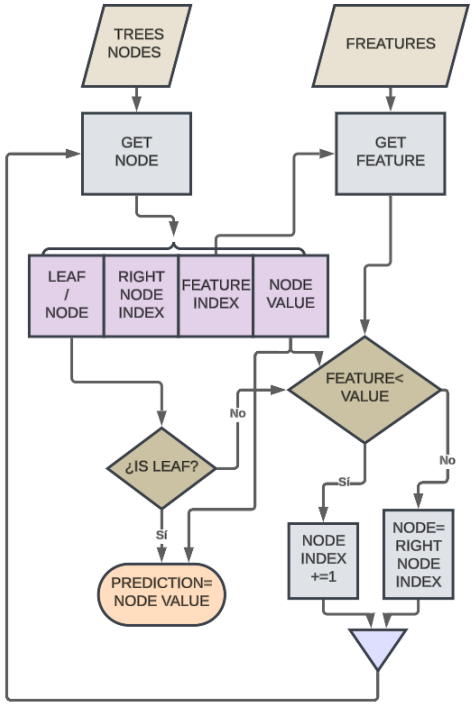

Fig. 3. Tree Architecture

access to the data they need, namely the features and the parameters of each node in each tree. To manage this, each tree has been assigned its dedicated memory to store its nodes, and the feature data are stored in registers to allow all trees to have parallel access.

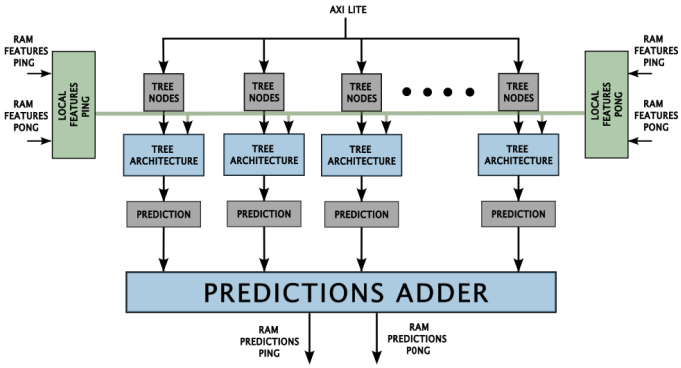

Fig. 4. Tree Forest Architecture (Tree Architecture from Figure 3)

Additionally, a ping-pong scheme has been implemented in both the memory and the registers. The memory stores all the features to be processed, while the registers provide fast access to the currently processed set of features. This strategy offers several advantages:

- **Reduced Latency**: Loading the next set of features to process from memory to the registers occurs concurrently with processing, eliminating waiting times between inference cycles at both the register and memory levels.
- **Minimized Memory Access**: Ping-pong registers ensure data availability in each cycle, reducing the need to access external memory and decreasing the total data access time.
- **Enhanced Parallelism**: With all trees executing concurrently, storing features in registers allows simultaneous processing. The ping-pong registers further improve parallelism by continuously providing feature data to the trees.

The system has been designed to perform feature processing in burst mode to increase throughput further. This reduces the number of control instructions that must be sent to the system, as a single set of instructions is sufficient for each burst to be processed. This approach involves sending a configurable number of features to process with a single instruction, and it works as follows:

1) A number N of features to be processed is sent.
2) The number of inferences to be performed is specified.
3) The signal to start processing is sent.
4) The system waits for the processing of the features to complete.

This strategy significantly reduces control data traffic to the accelerator, thereby increasing system performance. This improvement arises because it eliminates the need to send the start signal and wait for each inference's completion. Instead, a single operation is performed for each N-batch of inferences.

C. *System Integration*

A PCIe interface communicates with the previously described IP core (Tree Forest). A PCIe/AXI bridge facilitates this communication by converting data from the PC host over PCIe into AXI transactions, as shown in Figure 5. Two types of AXI interfaces are employed:

- **AXI Lite**: Used for components requiring lower bandwidth or minimal data traffic, such as loading the tree node data and controlling the Tree Forest IP core.
- **AXI Full**: Used for operations that involve higher data traffic, such as loading feature data and reading predictions. The AXI Full interface transfers feature data to the ping-pong feature memory and reads prediction results from the ping-pong memory.

This arrangement ensures efficient data transfer and optimal performance by matching the interface type to the specific bandwidth requirements of each operation.

V. Architecture Evaluation

A. *Experimental setup*

Multiple models were trained to evaluate the state-of-the-art system for different problems. The execution time was measured on various FPGA platforms with the proposed architecture and different CPUs and GPUs.

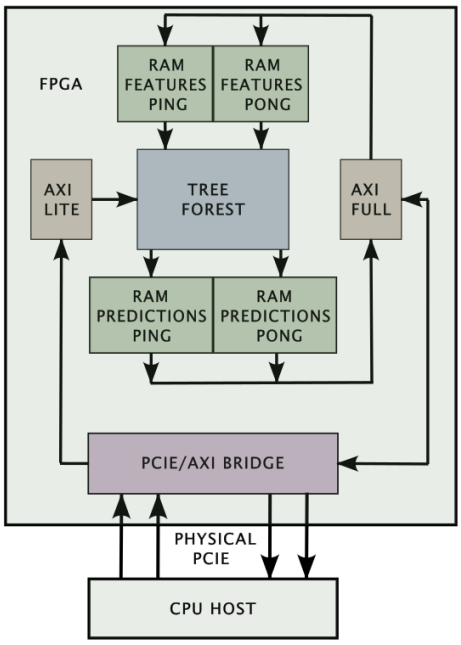

Fig. 5. System Integration (Tree Forest from Figure 4)

1) Models Description: The models are based on datasets sourced from Kaggle, each tailored for specific medical conditions. MODEL 1 uses the Diabetes dataset, which contains raw data, including records with missing values, for 768 patients and 8 features per patient. MODEL 2 utilizes the Heart Attack dataset, consisting of processed data for 303 patients, each with 13 features. MODEL 3 is built on the Lung Cancer dataset, which includes processed data for 3,000 patients and 15 features per patient. Lastly, MODEL 4 employs the Anemia dataset, containing processed data for 500 patients with six features per patient. The models' size varied depending on the number of trees being trained. In the case of using 512 trees, the models require 1048 MB, whereas with 128 trees, the storage of the models requires 262 MB. Due to this fact and size constraints, only 128 trees were used for the models deployed on the XC7K325T board.

2) Evaluated platforms: To evaluate the quality of the proposed hardware acceleration, the following devices were evaluated:

- Alveo U250: The Xilinx Alveo U250 accelerator card is an FPGA based on the UltraScale+ architecture, designed for data centre applications requiring high computational performance, such as machine learning and data analytics.
- XC7K325T: The Xilinx Kintex-7 XC7K325T is a medium-density FPGA with 326,080 logic cells and 840 DSP slices.
- E5-2696 v3: The Intel Xeon E5-2696 v3 processor features 18 cores and 36 threads, based on the Haswell-EP architecture, targeting servers and high-performance computing tasks.
- i7-8700K: The Intel Core i7-8700K is a processor with 6 cores and 12 threads, a base frequency of 3.7 GHz, and Turbo Boost up to 4.7 GHz, designed for high-performance desktop applications.
- RTX 4060: The NVIDIA GeForce RTX 4060 is a graphics card based on the Ada Lovelace architecture, aimed at 1080p gaming and content creation.
- RTX 3050: The NVIDIA GeForce RTX 3050 is an entry-level graphics card based on the Ampere architecture, intended for 1080p gaming.

3) Tree Parameters: The generated trees had the following characteristics:

- The number of trees: 512 (for all platforms except XC7K325T), 128 (for the XC7K325T platform).
- The depth of the trees: 8 (for all platforms).

4) Training Parameters: The training parameters used in LightGBM for model training were as follows:

- Learning rate: 0.1 (applied to all models and platforms).
- Training size: 80% and test size: 20% (applied to all models and platforms).

B. Experimental Results

These results show the average inference time of various models and are shown in Table II.

TABLE II
AVERAGE EXECUTION TIME FOR VARIOUS MODELS

	MODEL 1	MODEL 2	MODEL 3	MODEL 4
Alveo 250	**200ns**	**390ns**	**110ns**	**240ns**
XC7K325T	**205ns**	**394ns**	**109ns**	**242ns**
E5 2696V3	27us	18us	60us	60us
i7 8700k	23us	14us	48us	47us
RTX 4060	25us	27us	25us	25us
RTX 3050	49us	63us	42us	55us

Regarding the slight differences in latency between the Alveo U250 and the XC7K325T, it must be noticed that all trees on both boards are executed in parallel. This results in no overhead, even when more trees are added. In this context, observed latency differences arise from the Linux driver that manages communication with PCIe, the Linux scheduler, the possible state of the PCIe at that moment or the system load. Since PCIe is not used in the bare-metal mode, we do not have complete control over the timings.

The system operates at a clock frequency of 125 MHz. Table III provides a detailed overview of the resource utilization for different system configurations, specifically varying the number of decision trees processed in parallel. The metrics include the utilization of Block RAMs, Flip-Flops, and Look-Up Tables. This analysis helps evaluate the scalability of the design as the number of trees increases.

TABLE III
RESOURCES UTILIZATION (XC7K325T)

	16 trees	32 trees	64 trees	128 trees	256 trees
BRAMs	64	128	256	512	1024
Flip-Flops	16481	17347	29253	53128	100818
lookup table	13648	22927	44262	86912	172247

The memory resource utilization is calculated as follows:

$$M_{\text{total}} = T \cdot N \cdot M_n$$

Where:
T: the number of parallel trees.
N: the number of nodes per tree (256 in this case).
M_n: the memory required per node (64 in this case).

The number of trees that are executed in parallel determines the throughput of the proposed system, while an increase in the number of trees executed in parallel results in a proportional increase in the memory used.

The system's accuracy was equivalent to that achieved when running the models in software, with minor differences due to slight variations in the implementation of the floating-point operators. Additionally, the slightly higher accuracy of the FPGA implementations (Alveo U250 and XC7K325T) compared to other platforms can be attributed to the use of custom floating-point operators in the FPGA, which minimize numerical errors and result in more precise outcomes. Detailed results are shown in Table IV.

TABLE IV
ACCURACY FOR VARIOUS MODELS

	MODEL 1	MODEL 2	MODEL 3	MODEL 4
Alveo 250	**95.42%**	**96.42%**	**97.42%**	**94.42%**
XC7K325T	**95.42%**	**96.42%**	**97.42%**	**94.42%**
E5 2696V3	95.22%	96.22%	97.22%	94.42%
i7 8700k	95.35%	96.35%	97.39%	94.35%
RTX 4060	95.37%	96.39%	97.37%	94.37%
RTX 3050	95.35%	96.35%	97.35%	94.35%

The FPGA-based implementation demonstrated a speed advantage, achieving execution times between 200 and 400 times faster than the same implementation running on a CPU or GPU.

VI. CONCLUSIONS AND FUTURE WORK

This work presents an FPGA-accelerated architecture for decision tree model inference, implemented explicitly for the XGBoost algorithm. The proposed solution significantly improves inference speed compared to conventional CPU- and GPU-based systems, positioning itself as an efficient option for real-time applications. The hardware implementation reduces latency and optimizes data flow through a memory system and PCIe/AXI control, achieving concurrent processing that maximizes performance without compromising model accuracy. The accessibility of the architecture is also highlighted as a significant contribution, allowing users with limited hardware expertise to train and deploy models seamlessly via a Python script. Several opportunities for expansion and optimization are proposed for future work. First, the architecture could be adapted to handle more complex models, such as neural networks or ensemble models with heterogeneous structures, broadening its application to advanced classification and prediction tasks. Additionally, implementing model compression techniques, such as tree pruning or parameter quantization, could enhance memory efficiency on the FPGA, allowing the system to manage larger models without increasing resource consumption. Finally, integrating parameter self-tuning capabilities into the FPGA could make the system even more adaptable, dynamically adjusting resources and performance based on workload demands and application requirements.

ACKNOWLEDGMENT

This work was conducted as part of the TED2021-132768B-I00 project, which received funding from MICIU/AEI /10.13039/501100011033 and the European Union's NextGenerationEU/PRTR.

REFERENCES

[1] A. Kanani, S. Vaidya, and H. Agarwal, "LightFPGA: Scalable and Automated FPGA Acceleration of LightGBM for Machine Learning Applications," in *2021 25th International Symposium on VLSI Design and Test (VDAT)*, Sept. 2021, doi: 10.1109/VDAT53777.2021.9600900.

[2] M. Owaida, A. Kulkarni, and G. Alonso, "Distributed Inference over Decision Tree Ensembles on Clusters of FPGAs," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 12, no. 4, art. no. 17, pp. 1–27, Sept. 2019, doi: 10.1145/3340263.

[3] R. Nane *et al.*, "A Survey and Evaluation of FPGA High-Level Synthesis Tools," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 35, no. 10, pp. 1591–1604, Oct. 2016, doi: 10.1109/TCAD.2016.2593660.

[4] A. Gajjar, P. Kashyap, A. Aysu, P. Franzon, S. Dey, and C. Cheng, "FAXID: FPGA-Accelerated XGBoost Inference for Data Centers using HLS," in *2022 IEEE 30th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 15-18 May 2022, doi: 10.1109/FCCM53951.2022.9786085.

[5] K. Krueger *et al.*, "High-Throughput FPGA-Based Inference of Gradient Tree Boosting Models for Position Estimation in PET Detectors," *IEEE Trans. Radiat. Plasma Med. Sci.*, vol. 7, no. 3, pp. 253-264, Mar. 2023, doi: 10.1109/TRPMS.2023.3238904.

[6] M. Owaida, H. Zhang, C. Zhang, and G. Alonso, "Scalable inference of decision tree ensembles: Flexible design for CPU-FPGA platforms," in *2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 30 April - 2 May 2017, pp. 38-45, doi: 10.1109/FCCM.2017.12.

[7] S. Summers *et al.*, "Fast inference of Boosted Decision Trees in FPGAs for particle physics," *J. Inst.*, vol. 15, no. 5, pp. P05026, May 2020, doi: 10.1088/1748-0221/15/05/P05026.

[8] A. Alcolea and J. Resano, "FPGA Accelerator for Gradient Boosting Decision Trees," *Electronics*, vol. 10, no. 3, art. no. 314, Jan. 2021, doi: 10.3390/electronics10030314.

[9] LightGBM, "LightGBM's Documentation," Available: https://lightgbm.readthedocs.io. [Accessed: Sept. 2024].

[10] "Datasets," Kaggle, Available: https://www.kaggle.com/datasets. [Accessed: Sept. 2024].

[11] Scikit-learn, "Tree-Based Models," Available: https://scikit-learn.org/stable/modules/tree.html. [Accessed: Sept. 2024].

[12] AMD, "Alveo U250 Accelerator Card," Available: https://www.amd.com/es/products/accelerators/alveo/u250/a-u250-a64g-pq-g.html. [Accessed: Nov. 2024].

[13] XGBoost Documentation, "XGBoost: Scalable, Portable and Distributed Gradient Boosting (GBM, GBRT, GBDT) Library," Available: https://xgboost.readthedocs.io/en/stable/. [Accessed: Nov. 2024].

[14] Y. Zhang, J. Tong, Z. Wang, and F. Gao, "Customer Transaction Fraud Detection Using Xgboost Model," *2020 International Conference on Computer Engineering and Application (ICCEA)*, Guangzhou, China, 2020, pp. 554–558, doi: 10.1109/ICCEA50009.2020.00122.

[15] A. Ogunleye and Q.-G. Wang, "XGBoost Model for Chronic Kidney Disease Diagnosis," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 17, no. 6, pp. 2131–2140, Nov.–Dec. 2020, doi: 10.1109/TCBB.2019.2911071.

[16] K. Garg, K. S. Gill, S. Malhotra, S. Devliyal, and G. Sunil, "Implementing the XGBOOST Classifier for Bankruptcy Detection and Smote Analysis for Balancing Its Data," *2024 2nd International Conference on Computer, Communication and Control (IC4)*, Indore, India, 2024, pp. 1–5, doi: 10.1109/IC457434.2024.10486274.