

FARM: Fast Acceleration of Random forests via in-Memory traversal

Aymen Ahmed

Computer Science and Engineering
University of Michigan
Ann Arbor, MI, USA
ahmedaj@umich.edu

Valeria Bertacco

Computer Science and Engineering
University of Michigan,
Ann Arbor, MI, USA
valeria@umich.edu

Abstract—Mainstream artificial intelligence (AI) solutions commonly rely on deep neural networks (DNNs) for their training and inference. Such AI models are often impenetrable to human interpretation, limiting their potential for adoption in sensitive domains, where an understanding of the root factors that led to a specific inference outcome is critical. In this context, tree-based ensemble models, such as random forests (RFs), XGBoost, and LightGBM, have recently risen as a key family of AI models that are “interpretable”. However, their limited performance is far from fulfilling the needs of time-critical applications, thus hindering their adoption.

This work identifies data retrieval inefficiencies in several tree-based inference models and proposes FARM, a novel hardware solution to accelerate inference on those models. FARM comprises two hardware innovations: a Processing-in-Memory (PIM) accelerator that performs the key computations of tree traversal directly within the HBM banks, and a Skipped Query Groups (SQG) design that bypasses unnecessary data movement and computation by coalescing burst memory activity. Our evaluation shows that FARM delivers up to 12× performance improvement for the three ensemble models studied (RFs, XGBoost, and LightGBM), compared to a GPU–HBM baseline.

I. INTRODUCTION

Many applications in critical domains demand AI models that are not only fast and accurate but also “interpretable”, that is, the classifications they offer can be justified and summarized in terms that their human users can understand. In parallel, recent regulations, such as the “*EU AI Act*”, label AI-based safety-critical systems as “high risk”, mandating that these AI systems be capable of providing a justification for their actions to facilitate auditing, accountability, and users’ trust [28].

Currently, the most popular, widely available, and widespread AI engines are all based on complex neural networks (NNs), delivering significant performance thanks to specialized hardware accelerators, but offering limited to no interpretability. While there has been some progress in enhancing NNs with interpretability features, the success has been very limited [30] [7] [37]. Due to their black-box nature, interpretability of NNs is usually limited to “global explanations,” that is, a high-level summary of the model’s embedded decision logic, disconnected from the specific classification produced for a given input.

Fortunately, other types of AI engines, while not as popular, are much more amenable to interpretability. Tree ensemble models, including random forests (RFs), XGBoost, and LightGBM, excel in offering a transparent internal structure, and

produce inference responses that are derived from a sequence of hierarchical, condition-based decisions, making them highly interpretable. Their appeal extends beyond interpretability due to their robust performance on tabular datasets (structured data organized in rows and columns), their need for less training data to achieve comparable accuracy, and their simpler tuning compared to NNs [9] [19] [1].

However, tree-based models are still computationally expensive during inference, particularly on GPU platforms. The use of the basic GPU-High Bandwidth Memory (HBM) setup for inference on these models leads to two major inefficiencies. First, there is substantial data movement between memory and host, resulting in heavy bandwidth demand. Second, there is an inefficient usage of the available bandwidth as GPU threads traverse different trees within the ensemble, triggering divergence and causing irregular and uncoalesced memory access patterns. To quantify this inefficiency, we studied RF inference using the NVIDIA cuML Forest Inference Library (FIL) and observed that GPU memory data utilization drops to as low as 45%, meaning that more than half of the data fetched from memory goes unused.

In this work, we propose a new solution, FARM, that tackles both these limitations through two complementary solutions. First, its Processing-in-Memory (PIM) design greatly reduces the amount of data transferred between memory and the host by carrying out core traversal steps directly in the HBM banks. Unlike prior PIM approaches [38] [35] [11] [27], FARM imposes no restrictions on ensemble size or tree depth, making it scalable to real-world models. The PIM accelerator also serves as a necessary foundation for our Skipped Query Groups (SQG) solution, which greatly boosts data utilization during tree traversal. SQG identifies groups of queries that need not to be evaluated at a given node, and blocks them from being fetched or processed, eliminating unnecessary accesses and computation. In summary, this paper makes the following contributions:

- A fully adaptable PIM accelerator for tree-based inference (generalizable to other binary tree traversal tasks), which slashes host–memory data movement by performing key computations directly in memory.
- A novel Skipped Query Groups (SQG) technology, which bypasses the analysis (fetching and computing) of groups of queries that do not need to be considered at a given node.

- We evaluate FARM on multiple tree ensemble models, demonstrating a performance improvement of up to 12× when compared to a GPU–HBM baseline, and up to 90% energy reduction.

II. MOTIVATION AND BACKGROUND

A. Tree Ensemble Models

Ensemble models rely on a collection of weak models to create one that achieves better performance than each weak single model. In tree-based ensemble methods, decision trees serve as the weak learners. A decision tree is a supervised machine learning model composed of interconnected nodes that form a tree-like structure. As shown in Figure 1, the topmost node of a decision tree is called the root node. Each node in a tree may have child nodes, which in turn can have their own children. Nodes that split further are known as decision nodes, while nodes that do not branch further represent the final classification for a specific path and are referred to as leaf nodes. Accordingly, a path through a decision tree starts at its root node and ends at a leaf node.

Decision trees are effective for classification tasks that must predict categorical labels. Consequently, classification ensembles like the Random Forest Classifier [25] and Gradient Boosting Classifier [3] leverage multiple decision trees to determine discrete class labels. While both random forest and gradient boosted trees rely on ensembles of decision trees, they differ primarily in how these trees are constructed during training.

RF tree structure. An RF model consists of a collection of disjoint trees, each trained using random subsets of the training data and a random subset of features. Each tree is composed of internal nodes that perform binary feature comparisons and leaf nodes that store the final inference values [4] [6].

Gradient Boosted tree structure. In contrast, gradient boosted trees are built incrementally to improve the accuracy of the model by reducing the overall classification error [5] [16]. The sequential and corrective structure of boosted trees often leads to higher accuracy with smaller models. However, they require much longer to train, with some studies reporting up to 50 times longer [3].

B. Tree Ensemble Inference

During inference, the trained ensemble classifies new queries by traversing each tree from root to leaf based on feature conditions. Each tree produces an output, and the results are aggregated across the ensemble to form the final prediction. The process of sifting a query through each tree is similar across ensemble models, with the main difference being in how individual trees’ classifications are combined to produce the final output.

RF inference. Figure 1 (left) shows the basic traversal technique for a single RF inference. Each tree filters the query through its nodes, from root to leaf, based on the feature-condition mapped to each node. When the query reaches a leaf node, the value associated with that leaf becomes the tree’s inference for the query. The final RF inference outcome is determined by taking a majority vote across all trees.

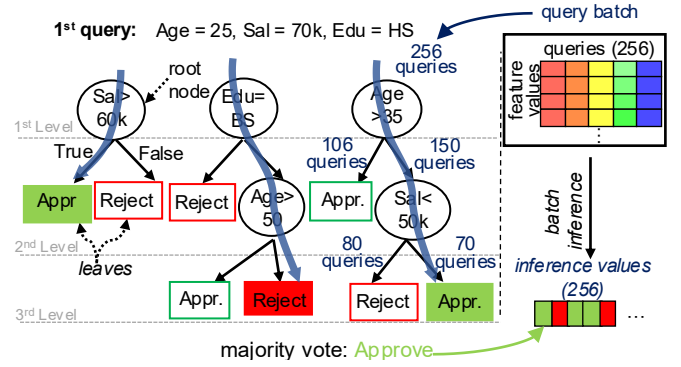


Fig. 1: A simple RF with three trees classifying loan approval as two of three votes approve the loan. Each node applies a test condition to select the next node to evaluate. The rightmost tree illustrates batch inference with 256 queries, arranged column-wise by query and row-wise by feature, leading to a final classification vector for the entire batch. As queries progress deeper, fewer remain active per node.

Gradient Boosted inference. In gradient boosted trees, each leaf node is associated with a numerical value, or “score,” which represents the model’s confidence in a given classification. In order to determine the likelihood of a given class, the scores from all trees are summed, scaled by their learning rates, and passed through a function that maps the result to a probability between 0 and 1. For multi-class tasks, a separate set of trees is trained for each class, with scores summed up within each set to select the class with the highest probability as the prediction.

During inference, queries can be evaluated individually to minimize latency or in batches to maximize throughput. In batch mode, queries can be arranged column-wise so that an entire row of feature values can be fetched and processed together, as shown in Figure 1 (right). The optimal batch size varies with the underlying hardware’s memory architecture [2]. To further improve efficiency, inference can exploit parallelism either by distributing queries across compute units, each holding a full copy of the model (data parallelism), or by splitting the model itself across units (model parallelism). In our evaluation, the latter proved more effective for large RF models, forming the baseline approach for this work (Section IV).

C. Tree Ensemble Inference Acceleration

The parallel nature of tree ensemble inference creates opportunities for GPU acceleration, where threads can process trees or nodes in parallel. However, GPU implementations introduce some new challenges: during batch inference, queries are split at each node, leaving progressively smaller subsets to be processed at each subsequent node. This trend underutilizes GPU threads, which must execute in lockstep by processing all queries in the batch and discarding the unnecessary results at the end. In addition, thread divergence arises when different queries follow different paths, causing scattered memory accesses.

We profiled RF inference execution for the Census dataset using the NVIDIA cuML FIL library with Nsight Compute

to evaluate memory utilization (the ratio of useful bytes to fetched bytes per transaction). We observed that GPUs achieve up to 90.9% utilization at shallow depths, where threads within a warp traverse similar paths and memory requests remain contiguous. However, as trees deepen and warp divergence scatters memory accesses, eliminating spatial locality, forcing entire 32-byte sectors to be fetched while only partially used, utilization gradually falls to 45% at depths of 10 and above. This problem is amplified by row-based accesses in HBM, where fetching a row requires 32 transactions of 32-byte packets [13], even though fewer queries remain active deeper in the tree. As illustrated in Figure 1, a batch of 256 queries may be fully utilized at the root, but subsequent nodes only process subsets of the batch as queries branch along different paths. For example, in the Higgs model, about 79% of nodes need data that occupies only a small fraction of a row, yet GPUs still retrieve whole sparsely utilized rows, wasting much bandwidth.

This underutilization and high bandwidth usage of a GPU-HBM setup have led to growing interest in Processing-in-Memory (PIM) approaches, which integrate lightweight compute units into the HBM logic layer to exploit locality and reduce unnecessary data movement. In the following section, we present our PIM-based solution, specifically designed to accelerate tree ensemble inference.

III. FARM

A. FARM Overview

FARM is a hardware accelerator designed to support tree ensemble inference tasks through two complementary components. The first is a Processing-in-Memory (PIM) accelerator, with one identical block deployed next to the row buffer of each memory bank, as illustrated in Figure 2. These units process all queries in a batch through every node of each tree, retaining key computations in memory and thus reducing congestion at the memory-host interface. The second component is the Skipped Query Groups (SQG) design (Section III-C), which further improves efficiency by bypassing computation and data fetching for query groups that are not relevant to the node under evaluation.

B. FARM PIM Architecture

System-level data organization. To best serve a model-parallelism implementation of a tree ensemble inference (see Section II-B), query data is organized by feature: for all queries in a batch, all values of one feature are adjacent in memory (Figure 1, right). For the HBM3 memory architecture we considered, memory rows are 1KB in size, thus, allowing 256 32-bit feature values to be stored per row. With thousands of rows per bank, each bank can accommodate thousands of features, which is sufficient for the models we target.

To ensure efficient parallel processing, the same batch of queries is processed through all trees in the model concurrently. Note that tree data is stored in other memory banks (whose banks' FARM units are inactive during the inference process). Trees are stored in vertex order, breadth-first (root node first, then all nodes at depth 1, then all nodes at depth 2, etc.)

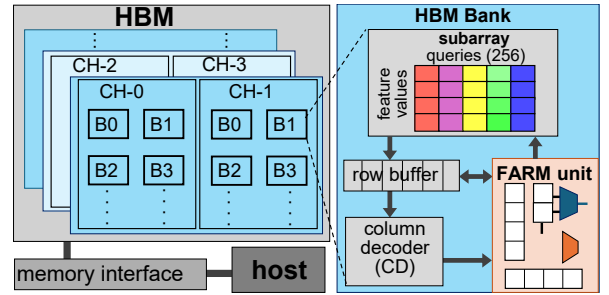


Fig. 2: FARM units co-located within each bank of an HBM

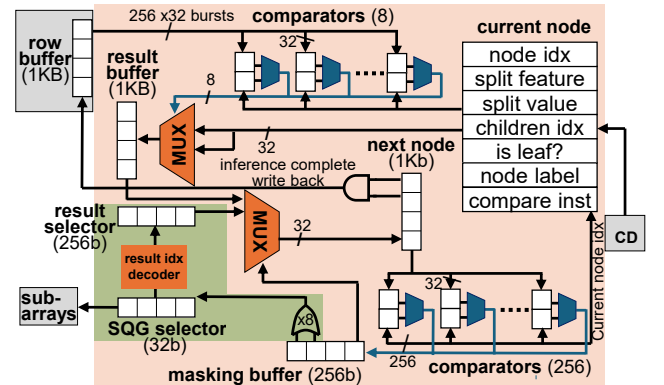


Fig. 3: FARM hardware architecture: the top set of comparators determines which child node to select for each query. The *next node* buffer stores the index of the next node that should process the query batch, while the bottom set of comparators filters which *next node* to update based on the current node's index.

One computation step of a FARM unit entails processing a batch of queries (256 in our implementation) through one whole tree. At the end of the computation, the inference result of each query through that tree is stored in the *next node* buffer (1KB to store 256 32-bit values), where a negative value indicates that the inference for the corresponding query is complete. Inference results are then transferred to the host, which coordinates the processing of all queries through all trees, and then computes final majority votes.

FARM functionality. To launch the processing of a batch of queries through one tree, the host sends a PIM command with the pointer to the root node of a tree. FARM fetches the data of the first node and updates its *current node* buffer with the corresponding information about the node, as shown in Figure 3. Based on the current node's selection feature, FARM identifies the index of the row with the corresponding feature in the memory bank that stores the query batch. The *current node* buffer also includes pointers to its children: one of them will be assigned to each query, as it is evaluated against the node's split condition. After all queries in a batch are processed through a node, the next node is loaded to the *current node* buffer, selecting among the children of the node just completed, then its corresponding queries are processed. Specifically, when FARM loads a new current node, it compares its index with all the entries in the *next node* buffer, and carries out its comparison process if at least one entry matches the *current node's* index. Once all queries have reached leaf nodes, all the

values in the *next node* buffer will be inference results (negative values). At this point, FARM sends an interrupt to initiate the transfer of the inference results back to the host.

FARM internal architecture and operations. Figure 3 outlines the detailed structure of the FARM architecture. At the start of an iteration, FARM loads the memory row corresponding to the *split feature* index, transferring to the row buffer all the values for that feature for the entire query batch. This process takes multiple distinct bursts depending on bank geometry and feature representation (32 in our case). As soon as the first burst completes, FARM begins to process all feature data through a battery of comparators, which are set up with the current node’s *compare instruction* ($=$, $<$, $>$), and *split value* (top of Figure 3). The output of these comparisons is combined with the children’s index field of the *current node* to update the *result* buffer: a 1KB vector with 256 32-bit index values for each query, specifying which next node the query should be processed at, based on the outcome of the current node’s analysis. Note that the *result* buffer may include spurious information, as it is possible that not all the queries need analyzing at the current node. Thus, the information stored in the *result* buffer must be filtered through the *masking* buffer to update only the appropriate entries in the *next node* buffer.

The *next node* buffer maintains accurate next-node information for all queries, specifying which node should analyze that query next. The *masking* buffer (256b to store 1 bit results from the comparators) is generated by comparing the *current node*’s index against each of the indices stored in the *next node* buffer, so that only updates relevant to the current node’s analysis are modified in the *next node* buffer.

When the current node is a leaf, the corresponding *result* buffer and *next node* buffer entries are updated with classification values, which we encode with distinct negative integers. For instance, in the case of RFs and with reference to the example in Figure 1, we encode *Accept* as -1 and *Reject* as -2. For boosted trees, leaf nodes instead store a score. In this case, the host applies an offset to encode the score as a negative value, and later reverses the offset when decoding.

C. FARM SQG Design

As mentioned above, data is loaded into the row buffer in multiple bursts of queries. In our design, FARM processes eight 32-bit queries at a time, as they all can fit in a single burst access at a time. As the analysis moves deeper into the tree, queries to be analyzed at a given node become sparser and sparser, possibly leading to entire bursts going completely unused. To optimize for this time-consuming and wasteful computation, we equipped FARM with a novel functionality, called Skipped-Query Groups (SQG). This feature employs a 32-bit *SQG selector* buffer. The buffer has one bit per row buffer burst, which tracks whether there is any query within each burst that needs processing at the current node. If no query must be analyzed, the entire burst can be skipped from loading (into the row buffer) and processing. The *SQG selector* is also used to update the *result selector* buffer using the *result idx decoder*. This decoder identifies the indices in the *result* buffer that should update the corresponding entries in the *next*

node buffer. The *result selector* buffer is then used to transfer values from the *result* buffer to the corresponding entries in the *next node* buffer, and skip entries when a skipped query group occurs. For example, if the second query group is skipped, then the *result selector* makes sure that the second group of results in the *result* buffer does not update the second group of entries in the *next node* buffer.

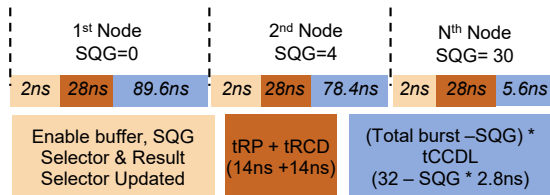


Fig. 4: FARM operation timing and row buffer fill latency.

D. FARM Timing

The performance of FARM is primarily constrained by the row buffer fill latency, which includes the row precharge time ($t_{RP} = 14\text{ns}$), row-to-column delay ($t_{RCD} = 14\text{ns}$ for HBM3 [13]), and column-to-column delay per burst ($t_{CCDL} = 2.8\text{ns}$). Assuming a 1 GHz clock, the maximum latency of 119.6ns occurs when all 32 bursts are loaded ($t_{RP} + t_{RCD} + 32 \times t_{CCDL}$, shown in blue in Figure 4). With SQG, this reduces to $t_{RP} + t_{RCD} + (32 - \text{SQG}) \times t_{CCDL}$, since skipped groups lower the number of bursts. Additional operations of comparing the *next node* and current node buffers, and updating the *SQG selector* each take one cycle (shown in beige in Figure 4). By contrast, the GPU-HBM baseline requires the same 119.6ns to fill the row buffer plus another 40ns to transfer data between memory and host (819 GB/s bandwidth, 16 channels per stack).

IV. EXPERIMENTAL EVALUATION

A. Experimental Setup

We evaluated FARM on three ensemble models, RFs, XGBoost, and LightGBM implemented with `scikit-learn` [26]. We determined the optimal training configurations via grid search, and the trained models with encoded negative leaf values were then exported for inference. We implemented the inference algorithm in C++ and compared it against the NVIDIA cuML library. For models with comparable classification accuracy, our baseline achieves similar execution time as the NVIDIA library implementation on some datasets and up to 2× faster performance on others, making it the preferred choice for integration with our hardware evaluation framework. We also evaluated two tree ensemble inference implementations, using both model and data parallelism (see Section II). We found that model parallelism is the best approach when taking into consideration our specific batch size and the different model sizes used.

Simulators. To simulate FARM, we leveraged and modified two existing simulators: ZSim [31], a multi-core system simulator, and Ramulator 2.0 [17], a cycle-accurate DRAM simulator. We modified the ZSim simulator to embed Ramulator and then modeled a multicore GPU processor with 16K cores connected to HBM3 memory (similar to NVIDIA H100). For

TABLE I: Properties of experimental datasets and models

Dataset	Random Forest		Boosted Trees		
	#Trees /Depth	Accu- racy	#Trees /Depth	XG boost Acc.	LG- BM Acc.
Census	200/45	91.9	128/10	87.1	88.8
Cover type	75/30	94.5	100/15	95.8	91.2
Higgs	150/20	73.6	110/15	75.3	74.9
MNIST	200/45	96.8	100/10	97.2	97.5
SUSY	150/30	80.1	128/8	80.3	80.2

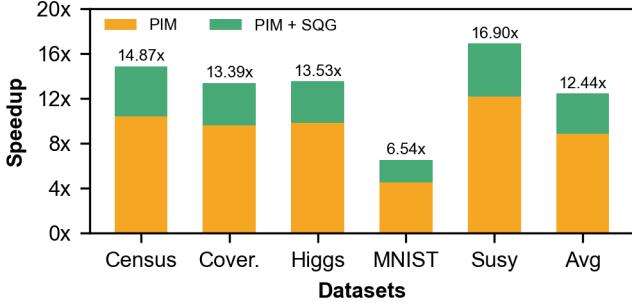


Fig. 5: Relative speedup of FARM over GPU-HBM memory baseline (PIM and SQG breakdown). We also show the baseline cycles under the dataset’s name.

the simulation of FARM units, we extended the GPU-HBM model in Ramulator, leveraging the approach from prior work [33]. Each FARM unit executes inference tasks in parallel with batches of 256 queries. We also assume a 1KB HBM row buffer (256 feature values), thus loading a row requires 32 bursts of 256 bits, with each burst transferring eight 32-bit feature values.

Datasets. Table I summarizes the datasets used in our evaluation, along with the number of trees and maximum depth of the trained random forest and gradient boosted models. Our selection includes three binary classification tasks (Census, Higgs, and SUSY) and two multiclass datasets (Covertypes with seven classes and MNIST with ten classes).

Performance results are presented in two parts. The first section highlights the performance benefits of FARM for RF models, while the second section demonstrates the advantages of our design for gradient boosted models, specifically XG-Boost and LightGBM.

B. RF Performance

Figure 5 represents the relative speedup of the FARM architecture compared to the GPU-HBM baseline. FARM provides an average speedup of 12.44x, of which approximately 30% is due to SQG data-access optimization, while the rest is attributed to the acceleration from performing computations in memory. The figure also shows that the performance improvements vary from 16.9x for the largest model, SUSY, to MNIST experiencing a 6.5x speedup. The reason for this difference is mainly due to differences in model sizes, depth, and distribution of nodes over the range of depths.

Sensitivity to RF node distribution. Figure 6 illustrates the distribution of nodes at various depth levels for the RF models used in our experiments. The figure shows that Higgs, Covertypes, and SUSY are wider compared to other models

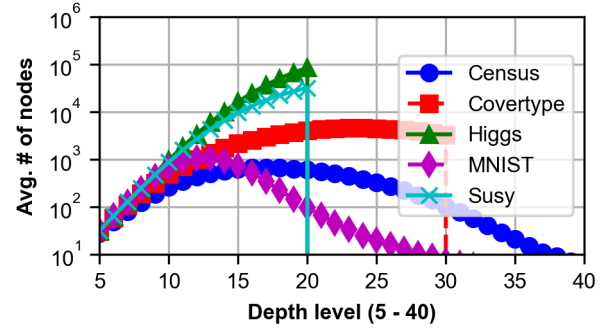


Fig. 6: RF model topology analysis: average number of nodes for each depth level in the trees comprising the model.

(as indicated by the area under each curve). Correspondingly, these models secure greater benefits from our FARM solution. Conversely, as can be noted in Figure 5, the MNIST dataset, despite its significant depth, exhibits limited improvement because many tree paths do not reach maximum depth, leading to a decrease in node count early on, a consequence of sparser trees not benefiting as much from our PIM solution. On the other hand, the Census dataset, while not as wide as other models, experiences significant speedup due to its deeper trees and the fact that queries are sifted to the deepest levels.

Sensitivity to tree depth. We also analyzed the effect of varying tree depths using the Higgs dataset. We observed that at depth 1, SQG provides no benefit since all queries must be sifted through the root node. At depth 5, the speed up due to SQG is 1.1x, as queries become sparser, allowing some groups to be skipped. However, improvements plateau at deeper levels, as reduced tree widths near the leaves limit again the number of query groups that can be skipped. For example, increasing tree depth from 15 to 20 improves performance only slightly (from 3.4x to 3.6x).

Data Transfer Reduction. We analyzed the amount of data transferred from memory to the host during inference to evaluate the impact of FARM. We observed that FARM significantly reduces memory data transfers, which directly contributes to the observed performance gains. Notably, datasets with higher reductions in data transfer tend to exhibit greater performance improvements. For example, the Higgs, SUSY, Covertypes, and Census datasets show the highest reductions—94.7%, 94.0%, 93.5%, and 91.5%, respectively, which aligns with greater performance improvements when compared to the MNIST dataset.

C. Gradient Boosted Performance

We evaluated two different gradient boosted tree models in our experiments: XGBoost and LightGBM. These boosted models also benefit significantly from FARM, achieving average speedups of approximately 5.4x for XGBoost and 5.3x for LightGBM over the GPU-HBM baseline, as shown in Figure 7. While these improvements are substantial, they are lower than the 12x speedup observed for random forests. This difference is due to the structure of gradient boosted trees, as they typically achieve comparable classification performance using

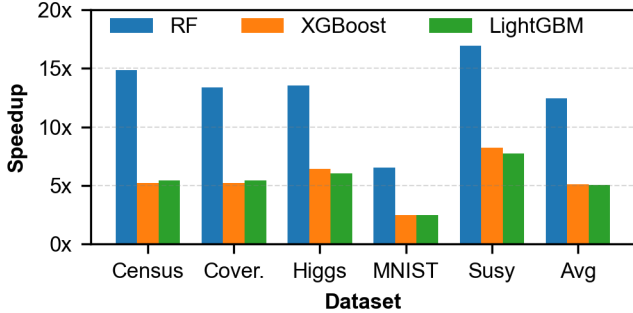


Fig. 7: Relative speedup of FARM over a GPU-HBM baseline for RFs and gradient boosted models (XGBoost and LightGBM). The last bar represents the average over all datasets across the different models.

shallower trees and fewer overall nodes. The smaller size and depth of gradient boosted models also reduce the effectiveness of FARM’s SQG mechanism, as queries are more likely to concentrate in a smaller number of active nodes, limiting the ability to skip memory bursts during traversal.

V. AREA AND ENERGY

A. Area

We implemented FARM in Verilog and synthesized it with Synopsys Design Compiler using the SkyWater SKY130 PDK to estimate the area required for each FARM unit. The results were scaled to a 20nm process node following [34], resulting in an estimated area of $81,974 \mu\text{m}^2$. We compared this estimate to Samsung’s general-purpose PIM unit at 20nm [20], which occupies $712,000 \mu\text{m}^2$: FARM uses only about 11% of the area, making it practical for HBM integration. Even under conservative assumptions of increased area of up to 8x [18] when implementing our design on a DRAM process, the footprint would remain well within commercial constraints.

B. Energy

FARM reduces system energy in three ways: minimizing data accessed in HBM through SQG, reducing transfers from HBM to the host, and offloading computation to in-memory components that consume less energy than the GPU host.

We model FARM’s energy use as the sum of subarray-to-row-buffer transfers ($E_{\text{SA-to-RB}}$, 1.2 pJ/bit), FARM component activity ($E_{\text{FARM-components}}$), and data transfers to the host ($E_{\text{data-transfer-to-host}}$, 3.97 pJ/bit [24]). Additionally, we modeled the GPU host energy consumption using data from Zsim output in combination with the NVIDIA H100 datasheet [23].

We observed that FARM cuts energy consumption by an average of 91% for RF models, with the highest reduction of 94% for SUSY, and the lowest reduction of 84% for MNIST. We also observed an average energy reduction of 77% and 78% for XGBoost and LightGBM, respectively. The slightly reduced energy saving is due to the smaller size and depth of the gradient boosted models. Table II reports the energy breakdown for Census: we can observe that most of the benefit comes from

TABLE II: Energy consumption for inference on Census dataset

Energy Consumption	RF	XGBoost	LGBM
Baseline Host	201.82 J	144.5 J	119.4 J
Data Transfer to baseline Host	1.15 J	0.9 J	0.6 J
Baseline Total	202.9 J	145.5 J	120.11 J
Host	13.57 J	27.87 J	22.0 J
Data Transfer to Host	0.43 mJ	0.01 mJ	0.02 mJ
Data Transfer to FARM	13.2 mJ	3.8 μJ	1.6 μJ
FARM Computation	0.85 μJ	0.009 μJ	0.01 μJ
FARM Total	13.5 J	27.87 J	22.06 J
Improvement over HBM	93.31%	80.85%	81.63%

TABLE III: Summary of related works

	Model	Max #Trees	Max #Depth	Solution
[36]	RF	-	6	GPU & FPGA
[11]	XGBoost	500	6	SRAM
[27]	XGBoost	4096	8	CAM
[8]	XGBoost-Binary	-	6	FPGA
[14]	RF	-	6	IC
FARM	RF, Boosted Trees	-	-	PIM

computing directly within FARM units, with additional savings from reduced data transfers.

VI. RELATED WORKS

In-memory computing accelerators have been developed for a variety of ML models [10], [12], [29]. While much attention has focused on accelerating tree ensemble training [15], [21], [35], some works have targeted inference on GPUs and FPGAs. However, these works either often constrain models to fewer shallow trees to fit within GPU texture memory and on-chip FPGA memory [32], [36] or focus on exploring memory layouts tailored to software/kernel-level optimizations [22].

Table III summarizes tree ensemble inference accelerators. Some of these works accelerate XGBoost using on-chip memory structures [11], [27], while [8] proposes an FPGA accelerator limited to binary classification. A common limitation across these designs is support for only shallow ensembles, typically capped at depths of 6–8 due to architectural constraints. By contrast, our target applications (Table I) entail depths of 20–45 for RF and 8–15 for boosted trees. Hence, FARM addresses the key limitation observed in prior work by supporting multiclass and large-scale ensembles. Additionally, our evaluation shows that its benefits improve as tree depth and ensemble size increase.

VII. CONCLUSION

The widespread use of AI has increased the demand for faster and more explainable solutions. Tree ensemble models address the interpretability demand but remain costly at inference, a phase that dominates most of the model’s lifecycle. As a solution, we present FARM, a flexible PIM-based accelerator embedded in HBM that performs key inference computations in memory and uses the Skipped Query Groups (SQG) technique to eliminate unnecessary data movement. FARM can tackle any type of tree ensemble model, and achieves up to 12 \times performance improvement over a GPU–HBM baseline, and reduces system energy consumption by as much as 90%.

REFERENCES

- [1] M. W. Ahmad, M. Mourshed, and Y. Rezgui, "Trees vs Neurons: Comparison between random forest and ANN for high-resolution prediction of building energy consumption," *Energy and Buildings*, 2017.
- [2] A. Ali, R. Pinciroli, F. Yan, and E. Smirni, "BATCH: Machine Learning Inference Serving on Serverless Platforms with Adaptive Batching," in *Proc. SC*, 2020.
- [3] C. Bentéjac, A. Csörgő, and G. Martínez-Muñoz, "A comparative analysis of gradient boosting algorithms," *Artificial Intelligence Review*, 2021.
- [4] L. Breiman, "Random forests," *Machine learning*, 2001.
- [5] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proc. SIGKDD*, 2016.
- [6] A. Cutler, D. R. Cutler, and J. R. Stevens, "Random forests," *Ensemble machine learning: Methods and applications*, 2012.
- [7] F.-L. Fan, J. Xiong, M. Li, and G. Wang, "On interpretability of artificial neural networks: A survey," *IEEE Transactions on Radiation and Plasma Medical Sciences*, 2021.
- [8] A. Gajjar, P. Kashyap, A. Aysu, P. Franzon, S. Dey, and C. Cheng, "Faxid: Fpga-accelerated xgboost inference for data centers using hls," in *Proc. FCCM*, 2022.
- [9] L. Grinsztajn, E. Oyallon, and G. Varoquaux, "Why do tree-based models still outperform deep learning on typical tabular data?" *Advances in Neural Information Processing Systems*, 2022.
- [10] M. He, C. Song, I. Kim, C. Jeong, S. Kim, I. Park, M. Thottethodi, and T. N. Vijaykumar, "Newton: A DRAM-maker's Accelerator-in-Memory (AiM) Architecture for Machine Learning," in *Proc. MICRO*, 2020.
- [11] M. He, M. Thottethodi, and T. Vijaykumar, "Booster: An accelerator for gradient boosting decision trees training and inference," in *Proc. IPDPS*, 2022.
- [12] M. Imani, S. Gupta, Y. Kim, M. Zhou, and T. Rosing, "DigitalPIM: Digital-based Processing In-Memory for Big Data Acceleration," in *Proc. GLSVLSI*, 2019.
- [13] JEDEC, "High bandwidth memory (hbm3) dram," 2023.
- [14] M. Kang, S. K. Gonugondla, S. Lim, and N. R. Shanbhag, "A 19.4-nJ/Decision, 364-K Decisions/s, In-Memory Random Forest Multi-Class Inference Accelerator," *IEEE Journal of Solid-State Circuits*, 2018.
- [15] T. Kari, L. N. S. B. A. D. R. K. Jagannatha, and S. Natarajan, "An Accelerated Approach to Parallel Ensemble Techniques Targeting Healthcare and Environmental Applications," in *Proc. ICEPE*, 2021.
- [16] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," *Advances in neural information processing systems*, 2017.
- [17] Y. Kim, W. Yang, and O. Mutlu, "Ramulator: A fast and extensible dram simulator," *IEEE Computer Architecture Letters*, 2016.
- [18] Y.-B. Kim and T. W. Chen, "Assessing merged dram/logic technology," *Integration*, 1999.
- [19] J. Kossen, N. Band, C. Lyle, A. N. Gomez, T. Rainforth, and Y. Gal, "Self-attention between datapoints: going beyond individual input-output pairs in deep learning," in *Proc. NeurIPS*, 2021.
- [20] S. Lee, S. Kang, J. Lee, H. Kim, E. Lee, S. Seo, H. Yoon, S. Lee, K. Lim, H. Shin, J. Kim, O. Seongil, A. Iyer, D. Wang, K. Sohn, and N. S. Kim, "Hardware architecture and software stack for pim based on commercial dram technology: Industrial product," in *Proc. ISCA*, 2021.
- [21] A. McCrabb, A. Ahmed, and V. Bertacco, "ACRE: Accelerating Random Forests for Explainability," in *Proc. MICRO*, 2023.
- [22] H. Nakahara, A. Jinguji, T. Fujii, and S. Sato, "An acceleration of a random forest classification using Altera SDK for OpenCL," in *Proc. FPT*, 2016.
- [23] NVIDIA, "Nvidia h100 gpu datasheet." [Online]. Available: <https://resources.nvidia.com/en-us-gpu-resources/h100-datasheet-24306>
- [24] M. O'Connor, N. Chatterjee, D. Lee, J. Wilson, A. Agrawal, S. Keckler, and W. Dally, "Fine-grained dram: Energy-efficient dram for extreme bandwidth systems," in *Proc. MICRO*, 2017.
- [25] A. Parmar, R. Katariya, and V. Patel, "A review on random forest: An ensemble classifier," in *Proc. ICICI*, 2019.
- [26] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in python," *Journal of machine learning research*, 2011.
- [27] G. Pedretti, J. Moon, P. Bruel, S. Serebryakov, R. M. Roth, L. Buonanno, A. Gajjar, L. Zhao, T. Ziegler, C. Xu *et al.*, "X-time: accelerating large tree ensembles inference for tabular data with analog cams," *IEEE Journal on Exploratory Solid-State Computational Devices and Circuits*, 2024.
- [28] J. Perez-Cerrolaza, J. Abella, M. Borg, C. Donzella, J. Cerquides, F. J. Cazorla, C. Englund, M. Tauber, G. Nikolakopoulos, and J. L. Flores, "Artificial intelligence for safety-critical systems in industrial and transportation domains: A survey," *ACM Comput. Surv.*, 2024.
- [29] S. Roy, M. Ali, and A. Raghunathan, "PIM-DRAM: Accelerating Machine Learning Workloads Using Processing in Commodity DRAM," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 2021.
- [30] C. Rudin, "Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead," *Nature machine intelligence*, 2019.
- [31] D. Sanchez and C. Kozyrakis, "Zsim: Fast and accurate microarchitectural simulation of thousand-core systems," *SIGARCH Computer Architecture News*, 2013.
- [32] M. Shah, R. Neff, H. Wu, M. Minutoli, A. Tumeo, and M. Becchi, "Accelerating Random Forest Classification on GPU and FPGA," in *Proc. ICPP*, 2023.
- [33] G. Singh, J. Gómez-Luna, G. Mariani, G. Oliveira, S. Corda, S. Stuijk, O. Mutlu, and H. Corporaal, "Napel: Near-memory computing application performance prediction via ensemble learning," in *Proc. DAC*, 2019.
- [34] A. Stillmaker and B. Baas, "Scaling equations for the accurate prediction of cmos device performance from 180nm to 7nm," *Trans. Integration*, 2017.
- [35] R. Struharik, "Decision tree ensemble hardware accelerators for embedded applications," in *Proc. Sisy*, 2015.
- [36] B. Van Essen, C. Macaraeg, M. Gokhale, and R. Prenger, "Accelerating a Random Forest Classifier: Multi-Core, GP-GPU, or FPGA?" in *Proc. FCCM*, 2012.
- [37] Z. Yang, A. Zhang, and A. Sudjianto, "Enhancing explainability of neural networks through architecture constraints," *IEEE Transactions on Neural Networks and Learning Systems*, 2021.
- [38] L. Zhao, Q. Deng, Y. Zhang, and J. Yang, "RFACC: A 3D ReRAM associative array based random forest accelerator," in *Proc. ICS*, 2019.