Published: 10 March 2025

RESEARCH-ARTICLE

VESTA: A Secure and Efficient FHE-based Three-Party Vectorized Evaluation System for Tree Aggregation Models

HAOSONG ZHAO, The Hong Kong University of Science and Technology (Guangzhou), Guangzhou, Guangdong, China

JUNHAO HUANG, Hong Kong Baptist University, Hong Kong, Hong Kong

ZIHANG CHEN, The Hong Kong University of Science and Technology (Guangzhou), Guangzhou, Guangdong, China

KUNXIONG ZHU, The Hong Kong University of Science and Technology (Guangzhou), Guangzhou, Guangdong, China

DONGLONG CHEN

ZHUORAN JI, Shandong University, Jinan, Shandong, China

View all

Open Access Support provided by:

Hong Kong Baptist University

The Hong Kong University of Science and Technology (Guangzhou)

Shandong University

VESTA: A Secure and Efficient FHE-based Three-Party Vectorized Evaluation System for Tree Aggregation Models

HAOSONG ZHAO, The Hong Kong University of Science and Technology (Guangzhou), China
JUNHAO HUANG, Guangdong Provincial/Zhuhai Key Laboratory of IRADS, BNU-HKBU United International College, China and Hong Kong Baptist University, Hong Kong
ZIHANG CHEN, The Hong Kong University of Science and Technology (Guangzhou), China
KUNXIONG ZHU, The Hong Kong University of Science and Technology (Guangzhou), China
DONGLONG CHEN, Guangdong Provincial/Zhuhai Key Laboratory of IRADS, BNU-HKBU United International College, China
ZHUORAN JI, School of Cyber Science and Technology, Shandong University, China
HONGYUAN LIU*, The Hong Kong University of Science and Technology (Guangzhou), China

Machine Learning as a Service (MLaaS) platforms simplify the development of machine learning applications across multiple parties. However, the model owner, compute server, and client user may not trust each other, creating a need for privacy-preserving approaches that allow applications to run without revealing proprietary data. In this work, we focus on a widely used classical machine learning model – tree ensembles. While previous efforts have applied Fully Homomorphic Encryption (FHE) to this model, these solutions suffer from slow inference speeds and excessive memory consumption. To address these issues, we propose VESTA, which includes a compiler and a runtime to reduce tree evaluation time and memory usage. VESTA includes two key techniques: First, VESTA precomputes a portion of the expensive FHE operations at compile-time, improving inference speed. Second, VESTA uses a partitioning pass in its compiler to divide the ensemble model into sub-models, enabling task-level parallelism. Comprehensive evaluation shows that VESTA achieves a 2.1× speedup and reduces memory consumption by 59.4% compared to the state-of-the-art.

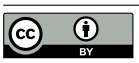
CCS Concepts: • **Security and privacy** → **Privacy-preserving protocols**; • **Computing methodologies** → *Classification and regression trees*; • **Software and its engineering** → Compilers.

Additional Key Words and Phrases: Domain-Specific Compiler, Homomorphic Encryption, Tree Ensembles, Parallelism, Vectorization

ACM Reference Format:
Haosong Zhao, Junhao Huang, Zihang Chen, Kunxiong Zhu, Donglong Chen, Zhuoran Ji, and Hongyuan Liu. 2025. VESTA: A Secure and Efficient FHE-based Three-Party Vectorized Evaluation System for Tree

*This work was completed while the author was affiliated with The Hong Kong University of Science and Technology (Guangzhou), China. The author is currently affiliated with Stevens Institute of Technology, USA.

Authors' Contact Information: Haosong Zhao, The Hong Kong University of Science and Technology (Guangzhou), Guangzhou, China, hzhao037@connect.hkust-gz.edu.cn; Junhao Huang, Guangdong Provincial/Zhuhai Key Laboratory of IRADS, BNU-HKBU United International College, Zhuhai, China and Hong Kong Baptist University, Hong Kong, Hong Kong; Zihang Chen, The Hong Kong University of Science and Technology (Guangzhou), Guangzhou, China; Kunxiong Zhu, The Hong Kong University of Science and Technology (Guangzhou), Guangzhou, China; Donglong Chen, Guangdong Provincial/Zhuhai Key Laboratory of IRADS, BNU-HKBU United International College, Zhuhai, China, donglongchen@uic.edu.cn; Zhuoran Ji, School of Cyber Science and Technology, Shandong University, Qingdao, China, zrji@sdu.edu.cn; Hongyuan Liu, The Hong Kong University of Science and Technology (Guangzhou), Guangzhou, China, hongyuanliu@hkust-gz.edu.cn.



ACM 2476-1249/2025/3-ART14
https://doi.org/10.1145/3711707

Aggregation Models. *Proc. ACM Meas. Anal. Comput. Syst.* 9, 1, Article 14 (March 2025), 26 pages. https://doi.org/10.1145/3711707

1 Introduction

Cloud computing platforms facilitate the development of machine learning applications by providing scalable, flexible, and cost-efficient infrastructure, along with tools and services that streamline the entire ML pipeline [1–3]. As a result, machine learning applications running in the cloud may involve models, client input data, and compute resources originating from different entities [46, 51]. However, entities may have privacy concerns and prefer to keep their data confidential from others [35]. For example, clients wish to keep their input data private from both the cloud server and the model provider, while the model provider seeks to safeguard the model's structure and weights from external exposure [35]. Preserving privacy is especially critical in domains such as finance (banking details), healthcare (patient information), government (citizen data), retail (customer information), and the workplace (employee data) [18, 27, 42, 45]. Machine learning services in these areas particularly rely on interpretable models for tabular data [26], such as decision trees [47] and tree ensembles like random forests [11] and gradient boosting trees [25].

The critical need to protect sensitive data in multi-party computations requires efficient secure machine learning protocols integrated with privacy-preserving techniques. Traditional cryptosystems like RSA or AES fall short because they require compute servers to decrypt data before processing, exposing it to potential breaches. To address these challenges, fully homomorphic encryption (FHE) [4] offers an attractive solution. FHE allows arbitrary computations directly on encrypted data without decryption, providing strong security [9, 28, 41, 48]. This enables compute servers to process encrypted data and produce encrypted results. FHE eliminates the need to trust cloud hardware, keeping data secure even if servers are compromised.

State-of-the-Art. While FHE can be used for certain machine learning models [23, 30, 39], applying it to tree ensembles remains challenging due to two main reasons: First, tree traversal, which is the fundamental operation in decision tree inference, is challenging to implement using homomorphic additions and multiplications. Second, tree inference involves irregular memory accesses and serial control flow, whereas FHE operations are typically executed in a parallel SIMD (Single Instruction, Multiple Data) fashion [14]. These challenges complicate the efficient adaptation of FHE for tree ensemble inference. To address these issues, COPSE [38] was proposed as the first FHE-based three-party secure inference system for tree ensembles. COPSE overcomes the challenges by transforming trained tree ensemble models into vectorized representations composed of vectors and matrices, enabling the use of matrix-vector multiplication for tree traversal. This approach aligns well with FHE operations and allows for parallelization. By encoding both the model and data as homomorphically encrypted vectors and matrices, COPSE enables secure computations on untrusted servers without privacy leakage.

Motivation. We systematically analyzed the COPSE system and observed that computing matrix operands within vectorized models predominantly increases its *runtime computation* and *memory consumption*. We identified two key reasons for these inefficiencies. First, the runtime performs multiple instances of matrix-vector multiplication over homomorphically encrypted data–one of the most time-consuming operations in FHE–which significantly slows down the evaluation. Second, each row of these matrix operands represents the tree structure in one-hot encoding. As the tree ensemble model grows, these matrices expand substantially, causing excessive memory usage and limited scalability. To address these inefficiencies, we propose two techniques: (1) **Compile-time Precomputation** and (2) **Runtime Batching**, and integrate them into our system VESTA.

Compile-time Precomputation. The key insight behind this technique is that complex computations performed in the COPSE runtime can be precomputed *in plaintext* during the compilation phase, rather than over ciphertexts at runtime. Specifically, COPSE runtime performs encrypted matrix-vector multiplications continuously as $A \times (B \times v)$, where A and B are both matrices in the vectorized model while v is an intermediate vector variable. In contrast, we can precompute the product $U = A \times B$ during compilation in plaintext so that the runtime computation can be reduced to a single encrypted matrix-vector multiplication $U \times v$. This precomputation not only eliminates one expensive matrix-vector multiplication at runtime but also decreases memory consumption by eliminating one matrix operand in the vectorized model.

Runtime Batching. Our key insight is that partitioning large tree ensemble models into smaller sub-models and batch-processing these at runtime can reduce the total memory footprint and decrease overall evaluation time by exploiting task parallelism. Specifically, in vectorized models, matrices encode the tree structures using one-hot encoding for each row. As the number of trees in the model increases, the size of these matrices grows quadratically with many redundant zeros—wasting computational resources and memory. By partitioning the tree ensemble into sub-models and processing them in batches, we effectively reduce the dimensions of the matrices, eliminating many of these redundant zeros. This leads to a smaller compiled vectorized model and lower memory consumption during runtime. Moreover, since each sub-model can run independently, runtime batching leverages task parallelism to speed up the inference process on multi-core processors. Overall, Runtime Batching reduces inefficiencies from one-hot encoding and boosts performance via parallelism, enhancing the scalability and efficiency of FHE-based tree ensemble inference.

Contributions. To the best of our knowledge, our proposed VESTA system is the most advanced FHE-based three-party secure inference system for tree ensemble models, offering the fastest evaluation speed and reduced memory consumption compared to existing systems, with the same level of security. Our contributions can be summarized as follows:

- We determined that one of the most time-consuming runtime computations could be precomputed at compilation time in plaintext, thereby reducing the runtime overhead by eliminating one matrix-vector multiplication. (Section 4)
- We observed that processing encrypted one-hot-encoded matrices of the trees requires excessive memory and redundant computations due to their sparsity. Therefore, we partitioned the model to reduce redundant zeros and enhance memory utilization. (Section 5)
- Our optimizations are integrated to VESTA. Compared to the state-of-the-art COPSE system, VESTA achieves an average speedup of up to 2.06× and a 59.4% reduction in memory usage. (Section 6 and Section 7)

2 Background and Preliminaries

2.1 Fully Homomorphic Encryption

Fully Homomorphic Encryption (FHE) [4] is a privacy-preserving technology that enables computations on encrypted data. Specifically, under an FHE scheme, performing operations on ciphertexts and then decrypting them yields a result equivalent to performing the same operations on corresponding plaintexts. This property can be expressed as $Dec\,(Enc(a)\ op\ Enc(b))$ = a *op* b where *Enc* represents encryption, *Dec* represents decrption, and *op* stands for addition and multiplication.

However, FHE has two limitations: high computation overhead and large memory usage. First, the compute time required for homomorphic operations on ciphertext is significantly greater than that for corresponding plaintext. Second, conducting homomorphic computations usually results in a notably large working set [5], so the memory consumption of executing an FHE-based system

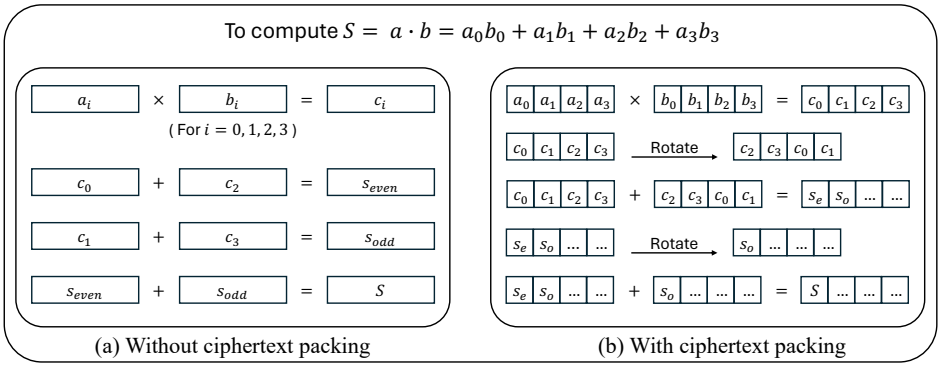


Fig. 1. Difference in computation and working set of inner product computation over homomorphic encrypted data with and without ciphertext packing technique

is substantially larger than the corresponding plaintext version. Therefore, an FHE-based system necessitates optimization technique in terms of computation overhead and memory utilization.

2.2 Ciphertext Packing

Ciphertext packing [14] is a memory and computation optimization technique widely used in FHE-based systems. Ciphertext packing creates a ciphertext as a vector packed with multiple values. Computations on packed ciphertext is equivalent to pairwise computations on each value. This allows a SIMD-like vectorized computation approach. When computations within a FHE-based system can be effectively vectorized in a SIMD fashion, the utilization of ciphertext packing can alleviate memory usage constraints and reduce computational complexity. Unlike vectorized computaiton in plaintext, direct indexing to access an encrypted value stored within a packed ciphertext is not supported. Additional operations, such as **rotation** or **multiplication by a reorder matrix**, are required to manipulate its contained encrypted value.

Figure 1 illustrates the impact of ciphertext packing on the working set size and computational complexity when calculating the inner product of vectors $a = [a_0, a_1, a_2, a_3]$ and $b = [b_0, b_1, b_2, b_3]$. When examining binary operations within a naive implementation (Figure 1(a)) in an FHE-based system, every input value (i.e. a_i , b_i) and intermediate result (i.e. c_i) is encrypted or stored within a ciphertext. Therefore, a total of 15 ciphertexts are utilized during the computation of a homomorphic inner product operation. Furthermore, by considering the binary operators involved, the homomorphic operations required in a naive implementation consist of 4 multiplications and 3 additions. By utilizing ciphertext packing to pack operand vectors into a single ciphertext and perform computations between packed ciphertexts (Figure 1(b)), it is observed that only 7 ciphertexts are required. Additionally, the necessary computations consist of 1 multiplication, 2 additions, and 2 rotations. It is worth mentioning that rotation is a specialized operation for packed ciphertext, with a computational cost higher than addition but comparable to multiplication. Furthermore, operations on ciphertexts under the same FHE setting have the same time complexity, regardless of whether ciphertext packing is utilized or not. Therefore, ciphertext packing effectively reduces the size of the working set and the number of required homomorphic encryption operations.

2.3 Decision Tree and Tree Ensemble Models

Decision tree. Figure 4 (a) depicts an example of a decision tree. A decision tree comprises two types of nodes: internal nodes and leaf nodes. At each **internal node**, a feature of the input feature

vector is compared against the node's **threshold**. Each **leaf node** includes a **label** denoting the predictive result of the input sample. The inference of a decision tree starts from the **root** and ends at a **leaf node**. A **decision** is made based on the comparison of a feature with the node's threshold. Depending on this decision, either the left or right child node is evaluated in a similar manner. This recursive process continues until a leaf node is reached. The label within the leaf node serves as the output of the model. In Figure 4(b), a running example of decision tree inference is illustrated. In the case of our trained decision trees, the nodes are re-arranged such that the left child node, namely **false branch**, corresponds to the branch to be evaluated when the decision is false, while the right child node is the **true branch**. Through the recursive evaluation, the leaf node with label L_2 is eventually reached with input feature vector $[x, y] = [3, 6]$ (Fig. 4 (b)).

Tree ensemble models. In general, tree ensemble models [24, 40] combine the inference results generated by multiple decision trees to enhance generalizability and robustness compared to a single decision tree model. The two predominant tree ensemble models commonly utilized are gradient-boosted decision trees [25] and random forests [11]. Given that the primary distinctions among different tree ensemble models lie in the manner in which individual decision trees are trained, the resultant models exhibit the same behavior in inference.

The inference process of tree ensemble models comprises two stages: the individual inference of each decision tree and the combination of their predictions. The combination step involves executing a reduction operation to derive the output based on the predictions of each tree. Typically, a majority voting technique is employed, where the label predicted by the majority of decision trees is designated as the final output.

2.4 Three-party Privacy-preserving Tree Ensemble Model Inference

Recent research [7, 38, 57] has established a three-party privacy-preserving tree ensemble model evaluation framework, in which the three computation participants are identified as the model owner, data owner, and compute server. Throughout the collaborative computation process, the fundamental configurations of these participants are outlined as follows.

Three parties. The **model owner** possesses the trained tree ensemble models. The features included in the trained models, along with all potential inference result labels, are considered public information and therefore known to all parties. However, the tree structure and specific threshold details are regarded as confidential model information.

The **data owner** possesses the input data that requires evaluation utilizing the trained tree ensemble models. The features present in the input data are identical to those in the models, making them public. However, the specific values of the input data should be kept confidential.

The **compute server** does not possess any input data, but rather the computational resources that enable the model owner and data owner to offload the inference computation to it. Acting as an honest-but-curious adversary [44], the compute server adheres to the underlying privacy-preserving protocol while attempting to learn all feasible information during its computations. In a secure system, the compute server should have no access to any confidential information pertaining to model privacy or data privacy.

Our goal of privacy. The core security consideration under a three-party scenario is to keep the model privacy as well as data privacy while offloading the computation to the honest-but-curious server. The **model privacy** of a tree ensemble model encompasses two key aspects: node and structure [57]. Specifically, **model node privacy** refers to the protection of which feature is used and threshold value stored in the internal nodes, as well as the predicted values stored in the leaf nodes. **Model structure privacy**, also known as path privacy, involves safeguarding the entire

path from the root to each leaf node while traversing the tree structure to generate the inference result. In three-party scenario, a system with model privacy should prevent compute server learn anything confidential from the encrypted tree ensemble model representation. Simultaneously, the data owner should not infer any sensitive information about the model based on the output result.

Data privacy involves protecting both the input feature vector and the output inference result. In three-party scenario, a system with data privacy should prevent the compute server from learning sensitive information from the data owner's input feature vector and the output inference result.

FHE-enabled three-party secure inference system. FHE has garnered interest in cloud computing, especially in the domain of MLaaS, where the three-party secure inference system is required. By applying FHE, both the model provider and data owner can utilize the compute resources of cloud servers without trusting either the servers themselves, their underlying hardware, or even their CPUs. Compared to other secure computation techniques like oblivious transfer [47] and secret sharing [56] , FHE-based systems are acknowledged for offering the highest level of security. Besides, FHE is complete where any complex computation that can be broken down into addition and multiplication operations can be designed into an FHE circuit. A naive implementation can be accomplished by encrypting each operand into a ciphertext and utilizing homomorphic addition and multiplication to perform computations as if operating on their plaintext versions.

Challenges in designing FHE-enabled three-party secure evaluation systems for tree ensembles. Implementing a secure and efficient FHE-based three-party secure evaluation system for tree ensembles is challenging due to its inherent incompatibility of FHE adaptation, and the stringent privacy constraints present in a three-party setting. Unlike other machine learning models such as neural networks [41], support vector machines [9], or unsupervised learning models [28], which rely on mathematical computations that can be easily converted into homomorphic additions and multiplications, the evaluation of tree ensembles involves operations that are FHE-unfriendly, specifically comparison and tree traversal. This necessitates a careful and innovative design to support these computations.

Additionally, existing systems [6, 19, 36, 58] are primarily designed to offer privacy-preserving tree aggregation evaluation protocols within a two-party setting, wherein either the data owner or the model owner possesses their own computational resources and functions as the compute server. In a two-party setting, it is sufficient to protect either data privacy or model privacy, as the other is not at risk of leakage. A three-party setting imposes significantly higher privacy-preserving requirements compared to a two-party setting, which complicates the direct adaptation of two-party systems to three-party systems. These two challenges render the design of an FHE-based three-party secure evaluation system for tree aggregation models a valuable research topic. Otherwise, such simple, useful, and interpretable models cannot be employed when conducting privacy-preserving computations with an untrusted computation server.

3 Motivation and VESTA System Overview

This section provides an overview of our VESTA system design. We will start by revisiting the state-of-the-art COPSE system and analyzing its inefficiencies in computation and memory usage. Subsequently, we will discuss how our VESTA system addresses these inefficiencies.

3.1 Revisit the State-of-the-Art

COPSE system [38] is the state-of-the-art FHE-based three-party privacy-preserving tree ensemble inference system. The COPSE system vectorizes operations required for the inference of tree ensemble model. Through ciphertext packing, its approach results in significant speedup compared to prior works [7, 13].

COPSE system infers tree ensemble models through the following four steps: **First**, COPSE system organizes all threshold values present in the tree ensemble models, along with input feature values, into vectors. It then utilizes the packed ciphertext property to enable a single comparison operation to obtain all decisions across the tree ensemble models. **Secondly**, COPSE system will select and arrange the comparison results of each feature against each threshold to its corresponding tree. The **third** step is the core of the inference process. COPSE system conducts a level-by-level tree traversal by selecting the comparison results of the current level from the results of the second step. It then moves to either the true branch or false branch based on the decision value. Since all operations are vectorized during the level-by-level tree traversal, the **fourth** step involves conducting accumulation operations to obtain the inference result from each level result.

To implement the aforementioned four steps, COPSE system conducts a co-design of the COPSE compiler and COPSE runtime. The scheme of the COPSE system is shown in the left half of Figure 2. The **COPSE compiler** (Top left part of Fig. 2: COPSE Compiler) is deployed on the model owner's computer, namely, Maurice's computer. The COPSE compiler takes a trained tree ensemble model as input and compiles all the necessary information of the models during the inference process into vectors and matrices, creating a **vectorized COPSE model**. Specifically, for a trained tree ensemble model with d levels, the compiler prepares for the following four data structures:

(1) A vector containing all threshold values from the internal nodes of the trained tree ensemble model, known as the **threshold vector** (`Thresh`);
(2) A reorder matrix that selects each tree's decisions in the second step, referred to as the **reshuffle matrix** (`Reshuf`);
(3) d reorder matrices to select each level's decisions, known as the **level matrices** (`Lvls`);
(4) d bit-vectors to store the true branch and false branch information of each level, referred to as the **level masks** (`Masks`).

Moreover, Maurice needs to guide the data owner, Diane, to organize Diane's input data features into a vector, known as the **feature vector** (`Feats`), where `Thresh` and `Feats` should establish a one-to-one relationship, allowing for an element-wise comparison between them.

Besides, the **COPSE runtime** (Bottom left part of Fig. 2: COPSE Runtime) is deployed on the compute server (Sally). After receiving the homomorphically encrypted vectorized COPSE model from Maurice and the feature vector (`Feats`) from Diane, Sally runs the COPSE runtime to conduct the four-step operations to infer the results. Specifically, the first step, **comparison**, is to use a secure comparison protocol over homomorphically encrypted data to compare the `Feats` against the `Thresh`. The second step, **reordering**, is to utililize a matrix-vector multiplication with `Reshuf` to arrange and select decisions corresponding to each tree. Next, the third step, **level processing**, is to perform tree traversal level by level. This includes using a vector-matrix multiplication with each level's `Lvls` to obtain the decisions at that level, followed by a vector addition with that level's `Masks` to distinguish true and false branches. The last step, **accumulation**, entails using a point-wise multiplication to aggregate each level's results into the final result. This results in a vector known as the **label vector** (`Labels`) that stores the encrypted one-hot inference result, indicating which leaf node has been reached. The `Labels` will then be sent back to Diane. After decryption, Diane will know the inference result.

3.2 Inefficiencies of COPSE System and Our Key Insights

During the preliminary experiment while reproducing the COPSE system, it was observed that running the COPSE system led to excessive memory usage and slow evaluation. Upon further characterization, it was identified that the computation steps involving matrix-vector multiplication with the reorder matrix had an excessive working set size and were the computation bottleneck. A

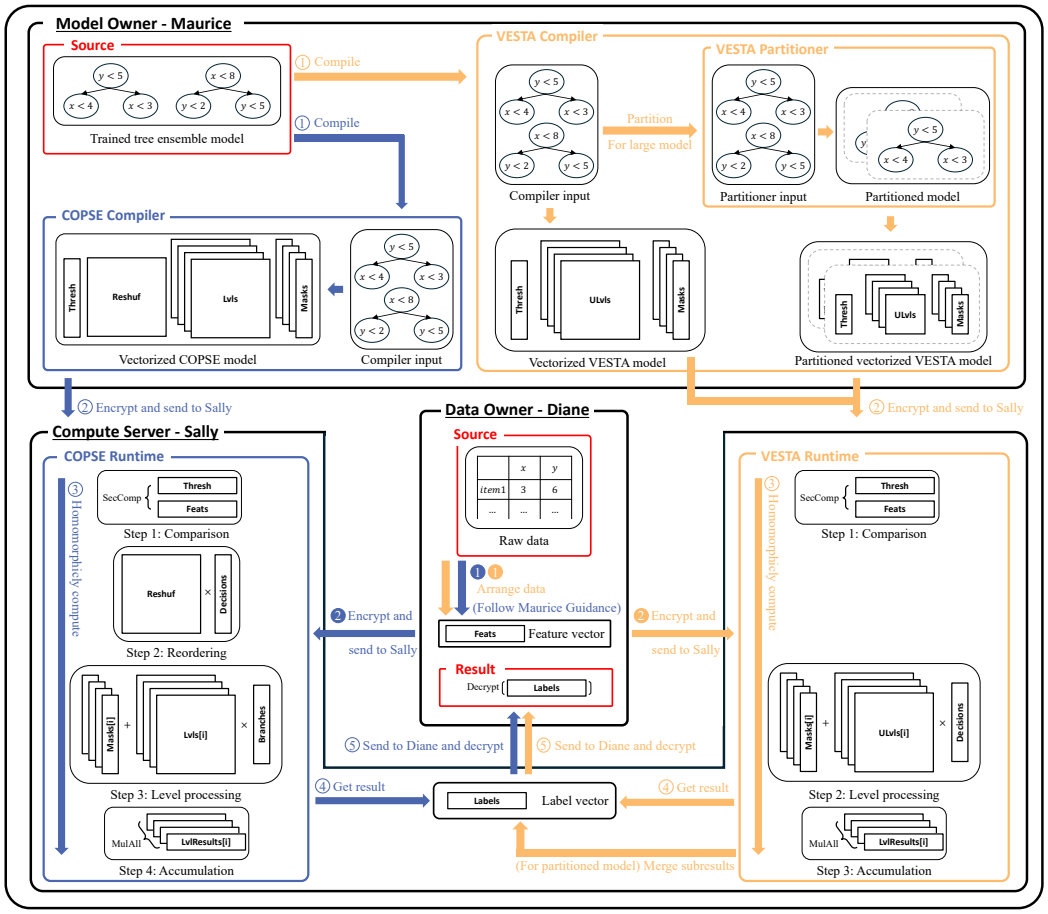


Fig. 2. Evaluation process of trained tree ensemble models by model owners, computer servers, and data owners within three-party secure inference systems (Left: COPSE system; Right: VESTA system)

brief introduction to two inefficiencies we found restricting the performance of the current COPSE system and a discussion about our proposed solutions are outlined below.

Consecutive matrix-vector multiplication with reorder matrices known during compilation time. The COPSE system involves two steps: reordering and level processing, both of which entail matrix-vector multiplication using the reorder matrix to select relevant decisions from the comparison results. Methodologically, decisions are initially selected and organized for each tree during the reordering step, followed by selection and arrangement for each level in the level processing step. One might assume that selection could be accomplished by directly accessing indices in the vectors. However, this is not feasible due to the limitations of ciphertext packing. As a result, two matrix-vector multiplications with reorder matrices are required to execute the two-step selections. However, by directly selecting each level's decisions from the comparison results, only one selection operation is necessary.

Furthermore, mathematically speaking, the implementation of these two steps involves first multiplying the `Reshuf` matrix by the comparison results (**decision vector**, *Decisions*), and then

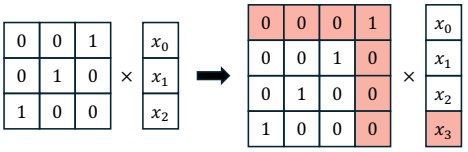


Fig. 3. An example of quadratic expansion of reorder matrix with operand vector size growth

multiplying the `Lvls` matrix by the resultant product of the previous multiplication. Consequently, the entire computational process for i-th level can be represented as "`Lvls`$[i]\times$`Reshuf`$\times$Decisions". Since the `Reshuf` and `Lvls` matrices are data structures of the vectorized COPSE model, known during the compilation process, it is advisable to perform their multiplication in the one-time compilation process rather than computing them at runtime using expensive homomorphic operations repeatedly. In summary, our initial optimization insight is to precompute the consecutive reorder matrix multiplication during compilation time to reduce one high-complexity runtime matrix-vector multiplication over homomorphically encrypted data.

Quadratic expansion of reorder matrix with model size growth. The COPSE system typically compiles each trained tree ensemble model into a single vectorized COPSE model. However, the size of the vectorized model will significantly increase as the model size grows due to the quadratic expansion of reorder matrices involved.

Figure 3 illustrates that when the size of the vector to be reordered increases by 1, the reorder matrix has to increase by a new column and a new row. Therefore, in our scenario, the reorder matrix grows quadratically in size as the number of tree nodes increases. As each row in a reorder matrix can have at most one non-zero value, the reorder matrices within the vectorized COPSE model compiled from large tree ensemble models will be excessively large and filled with numerous meaningless zeros. While those zeros could have been skipped considering the sparsity, as the matrices are encrypted, it is not feasible to leverage the sparsity. Thus, we consider to partition the large trained tree ensemble model into several sub-models. The benefits of doing it are two-fold: First, this approach ensures that the reorder matrices within the vectorized COPSE sub-models will have a smaller total size compared to the original model. Second, since the evaluation of sub-models is independent, partitioning the model introduces task-level parallelism that is inherent in ensemble models. In summary, our second optimization insight is to partition the large trained tree ensemble model and batch the inference process to improve memory and computation efficiency.

3.3 VESTA System

In response to the two inefficiencies mentioned earlier, we propose the VESTA system, a fast and memory-efficient three-party FHE-based secure inference system for tree ensemble models. Following a similar inference methodology as the COPSE system and combining the optimization techniques of precomputation and partitioning, our VESTA system also consists of a compiler and a runtime. In addition, we initially introduce a partitioner pass into our VESTA compiler design.

The scheme of VESTA system is shown in the right half of Figure 2. The **VESTA compiler** (Top right part of Fig. 2: VESTA Compiler) is deployed on Maurice's computer. When the trained tree ensemble model is received, VESTA compiler assesses the necessity for partitioning. In cases where large models require partitioning, VESTA compiler partitions the trained tree ensemble model into multiple sub-models in the **VESTA partitioner pass** (Top right part of Fig. 2: VESTA Partitioner). Next, VESTA compiler compiles the model or sub-models into **vectorized VESTA model(s)**.

During the compilation process, the optimization technique of precomputation is applied. As a result, the vectorized VESTA model contains only three data structures: `Thresh`, `ULvls`, and `Masks`. Note that `Thresh` and `Masks` remain the same as COPSE. The **unified level matrices** (`ULvls`) are the precomputation result, where `ULvls`$[i]$ =`Lvls`$[i]\times$`Reshuf` for the i-th level.

With the vectorized VESTA model containing precomputed results, the **VESTA runtime** (Bottom right part of Figure 2: VESTA Runtime) deployed on the Sally server only requires three computation steps. By utilizing the precomputed `ULvls`, the two consecutive matrix-vector multiplications carried out in both COPSE reordering and level processing steps can be consolidated into a single multiplication with `ULvls` within the VESTA level processing step. Therefore, the original reordering step in the COPSE runtime is eliminated in the VESTA runtime, while the comparison and accumulation steps remain unchanged.

For partitioned models, since the representation of vectorized models remains consistent, evaluating a partitioned model is similar to evaluating multiple non-partitioned models concurrently. In this scenario, Diane must generate `Feats` for each sub-model and then acquire the predicted results (`Labels`) for each model. Subsequently, a reduction operation, like majority voting, is performed by Diane to obtain the final result. Conversely, for Sally, each partitioned model operates in the same manner as before, making it challenging to differentiate between evaluating several independent models or evaluating a single large model that has been partitioned.

4 Compilation Process and Runtime Algorithm of VESTA System

In this section, we will delve into the technical details of the compilation process and runtime algorithm of our proposed VESTA system. We will provide a detailed example of how to use the VESTA system for conducting tree ensemble model inference. Following that, we will compare the VESTA system with the COPSE system in terms of memory usage, computational complexity, and security level.

4.1 Our Mechanism of Tree Evaluation Using VESTA System

Figure 4 illustrates a practical example of utilizing the VESTA system to perform tree ensemble model inference. It is important to note that all vectors and matrices involved in the runtime computation are encrypted in packed ciphertexts. For clarity and ease of comprehension, the corresponding plaintext values are depicted in our example. We consider a tree ensemble model comprising only a single decision tree (Figure 4(a)). When the input sample is $X = [x, y] = [3, 6]$, the evaluation result of this model should be L_2 (Figure 4(b)).

Featuring. Our approach organizes the threshold values of the internal nodes into a vector `Thresh`. During runtime, Diane prepares for a feature vector `Feats` using the sample for prediction. Each element in `Feats` must accord to an element in the `Thresh` such that the comparison results between features in the sample and the threshold values in the internal nodes could be acquired by an element-wise comparison.

Specifically, to organize the values in `Thresh`, our approach groups the thresholds by the feature to be compared with (i.e. 3 threshold values compared against x and 2 threshold values compared against y, Figure 4(c)), and then sorts them by the preorder traversal order of the trees (Figure 4(c)). To organize `Feats` vector, our approach duplicates each feature by the number of times it appears in internal nodes among the whole model and arrange them following the same feature organization order of `Thresh` (i.e. first x, then y, Figure 4(c)).

In the bottom right of Figure 4(c), a special case is illustrated where Maurice deliberately adds redundant values to obfuscate the actual number of nodes compared against a specific feature in the current model for security reasons. As long as Maurice instructs Diane to organize the `Feats`

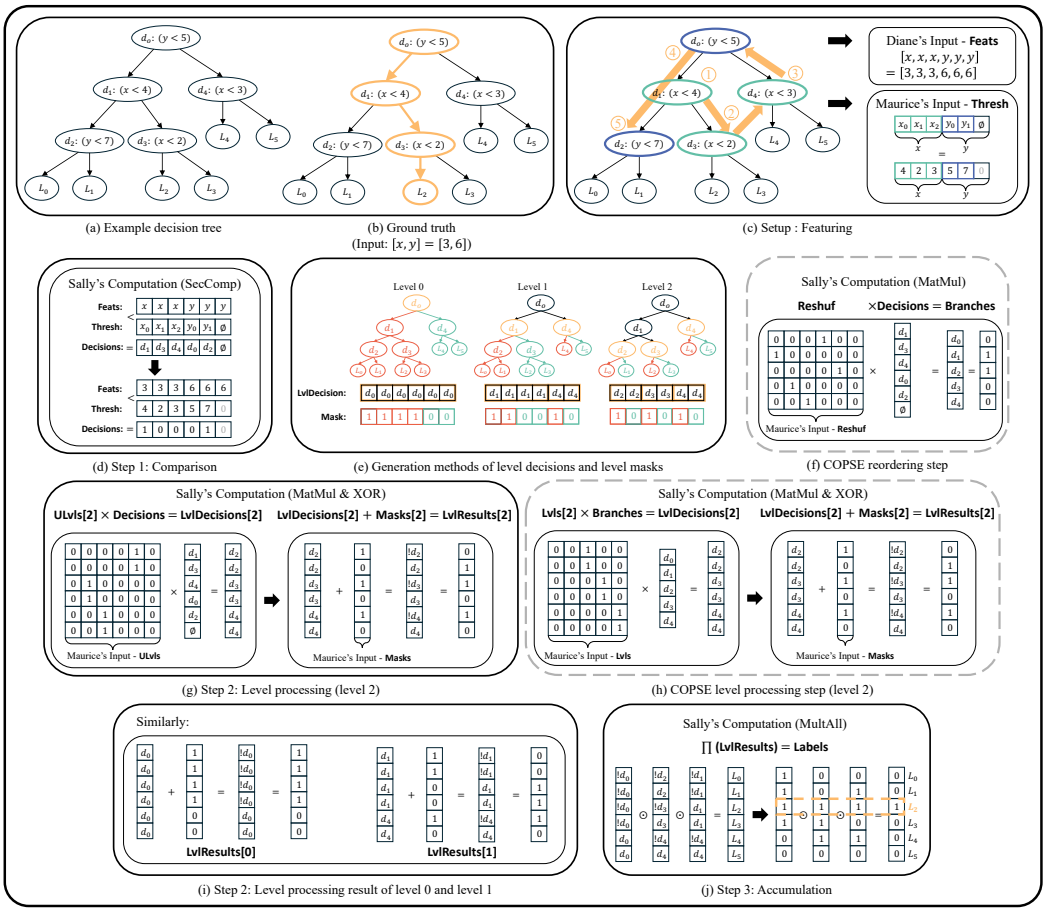


Fig. 4. A working example of VESTA system, showing the computation step differences compared to the COPSE system

vector with the same length and correct one-to-one mapping as shown in the top right of Figure 4 (c), the VESTA system will function properly.

Comparison. Diane passes `Feats` to the server (Sally) for evaluation. Sally compares `Feats` with `Thresh` using secure compare protocol. This results in a vector of evaluated 0 and 1 values, namely *Decisions* (Figure 4(d)). The values in the *Decisions* vector are then used for a level-by-level tree walk from the root node to a leaf node.

Level Processing. Our approach implements the tree traversal process by constructing a one-hot result vector that indicates the reachability of each leaf node. The width of this result vector is the same as the number of leaves (w). When a leaf node is reached, the corresponding element in the result vector is set to 1. To successfully reach a leaf (i.e. l_2, Figure 4(b)), all nodes along the path from the root to that leaf must be reachable (i.e. nodes contain d_0, d_1, d_3, Figure 4(b)). The reachability of an intermediate node (*LvlResults*) is determined by two factors: the decisions made at that node (i.e. *LvlDecisions*, Figure 4(e)), and whether its true or false branch should be traversed

(i.e. Masks, Figure 4(e)). To obtain the leaf reachability, we perform a sequence of matrix-vector multiplications and element-wise bit-vector additions.

First, we need to characterize "what" nodes could affect whether a leaf node could be reached. Thus, we construct a bit-vector *LvlDecisions*[i] of size w using the values stored in the *Decisions* vector for level i. The jth element in a *LvlDecisions*[i] is 1 if whether leaf node j could be reached is determined by a comparison result of a node. For example, the reachability of leaf L_1 is determined by the comparison result of the node in level 0 (i.e. d_0), the comparison result of the node in level 1 (i.e. d_1), and the comparison result of node in level 2 (i.e. d_2, Figure 4(e)). At runtime, constructing *LvlDecisions*[i] using values stored in the *Decisions* vector is implemented by multiplying the reorder matrix ULvls[i] (Left part of Figure 4(g)).

Second, we must characterize "how" the evaluation results (d_i) affect whether a leaf node is reached. Specifically, the comparison result of an internal node drives the tree walk to go to the left sub-tree or the right sub-tree, thus affecting whether a leaf node will be reached or not. In our settings, the tree walk goes to a node's right sub-tree if the comparison result is 1. Thus, we use an additional Masks[i] bit-vector for each level i. Specifically, for level i, the jth element in Masks[i] is 1 if the jth leaf node can be reached when the corresponding comparison results in *LvlDecisions*[i][j] is 0 (Figure 4(e)).

Having the *LvlDecisions* bit-vectors and Masks bit-vectors, we compute the *LvlResults*[i] bit-vector for each level i by adding *LvlDecisions* and Masks. The result bit-vector *LvlResults*[i] indicates the tree walk decisions in level i, with the jth element showing whether the tree walk will take the subtree that will reach leaf node j.

Accumulation. Since each level is computed independently, to compute the result vector, we conduct point-wise multiplication to reduce the *LvlResults* vectors into result vector, Labels. This finishes the inference process of a single tree.

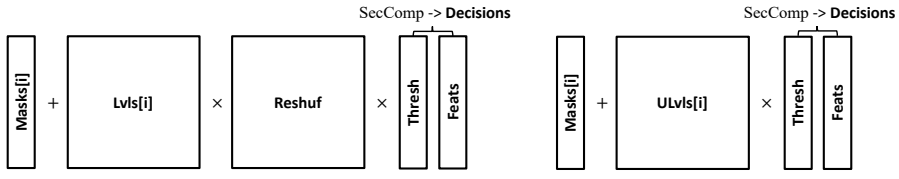
4.2 Comparison between COPSE and VESTA

This section presents a comprehensive comparison between our VESTA system and the COPSE system demonstrating the advancements of our system in terms of memory usage and computational complexity, while keeping the same security properties.

Key Difference to COPSE. Methodologically, our VESTA system and the COPSE system are constructed using distinct inference algorithms. Specifically, after obtaining all comparison results from the secure comparison protocol, the COPSE system performs a two-step selection process: it first assigns each decision to its corresponding tree to enhance locality, and subsequently allocates it to the appropriate node. In contrast, our VESTA system directly assigns each decision to its corresponding node.

As for implementation, in contrast to our approach, which directly selects *LvlDecisions*[i] from *Decisions* (Left part of Figure 4(g)), COPSE first arranges the *Decisions* into an intermediate vector called *Branches* (Figure 4(f)), following the preorder tree traversal visiting order during the reordering step. Subsequently, each level's *LvlDecisions*[i] are selected from *Branches* during the level processing step (Left part of Figure 4(h)). The arrangement and selection operations of these two steps are carried out by multiplying with the reorder matrix Reshuf (Figure 4(f)) and Lvls[i] (Left part of Figure 4(h)).

Some may question whether the lack of locality will diminish the performance of the VESTA runtime. However, following partitioning with the VESTA partitioner pass (Section 5), which can significantly enhance locality, any potential side effects can be mitigated.



(a) COPSE Runtime (b) VESTA Runtime

Fig. 5. Complete computation process for the i-th level's *LvlResults* in two systems

Complexity Metrics. We compare both memory consumption and computation complexity with COPSE using the size of the vectors and matrices contained in vectorized models. As previously mentioned in Section 2.2, all system inputs and intermediate variables are encrypted into packed ciphertexts, and all operations performed during runtime are executed through fixed-time [31] vector-wise computations between these packed ciphertexts. Therefore, both memory consumption and computational complexity are directly proportional to the size of the vectorized model, which ultimately determines the number of packed ciphertexts utilized during runtime.

As for the components of vectorized models compiled by the two systems, their dimensions and quantities are determined by three parameters of the corresponding tree ensemble models: the internal node number (b), leaf number (width, w), and level number (depth, d).

It is important to note that Maurice can add an unlimited number of redundant values to the `Thresh` to convert it into a length-q vector where q is an arbitrary number larger than b. This approach allows the actual numbers of a feature used to compare against among all nodes to be hidden from Diane while organizing the `Feats` (i.e. In Figure 4(c), y is actually compared twice in current model, but Diane may perceive it as being compared with the thresholds of three different nodes). However, this setup can lead to reduced computational efficiency as redundant computations are carried out. In our analysis, we establish a **standard setting** where q is equal to b, as we prioritize evaluating the model's performance rather than enhancing the security level by obfuscating the actual number of features being compared in the current model.

Size of the Vectorized Models. According to the working example shown in Figure 4, when working on the same inference task with our standard setting, the vectorized model data structures, `Thresh` and `Masks`, are the same for two systems. Specifcally, `Thresh` is a length-b vector (Figure 4(c)), and `Masks` is a $d \times w$ matrix that stores d levels' length-w level mask vectors (Figure 4(g), (h) and (i)). Besides, the shape of the reorder matrix that transforms a length-u vector into a length-v vector is $v \times u$. Thus, we can infer the shape of the reorder matrix contained in the COPSE system: `Reshuf` is a $b \times b$ matrix (Figure 4(f)) and `Lvls` is a $d \times w \times b$ tensor that contains d levels' $w \times b$ level matrices (Figure 4 (h)). As for the VESTA system, the `ULvls` matrix is a $d \times w \times b$ tensor that contains d levels' $w \times b$ unified level matrices (Figure 4(g)). Here, we can observe that even though the value of `Lvls` in COPSE is different from `ULvls` in VESTA, they have the same shape. Therefore, we can conclude that compared with COPSE, the compiled vectorized model of VESTA reduces a $b \times b$ matrix component with no other trade-off in size.

Comparisons in runtime computation complexity. We illustrate the difference between the COPSE runtime algorithm and the VESTA runtime algorithm for computing each level's *LvlResults*[i] in Figure 5(a) and (b), respectively. To get all *LvlResults*, the computation conducted in the COPSE runtime (Figure 5(a)) includes one-time invocation of the secure compare protocol of two length-b vectors (Figure 4(d)), one-time multiplication between a $b \times b$ matrix and a length-b vector

(Figure 4(f)), d times multiplication of a $w \times b$ matrix and a length-b vector (Figure 4(h)), and d times vector addition of a length-w vector (Figure 4(h) and (i)).

In the VESTA runtime (Figure 5), the difference is that it does not compute *Branches* but uses another matrix operand `ULvls` and vector operand *Decisions* to compute *LvlDecisions* (Figure 4(g)). Considering that `Lvls` and `ULvls` have the same amount and size, our VESTA runtime has a lower runtime computation complexity during this process compared to COPSE runtime since it reduces one-time multiplication of a $b \times b$ matrix and a length-b vector with no other trade-off in computation due to the compilation time precomputation.

After obtaining all *LvlResults*, an accumulation operation is performed with $w - 1$ times point-wise vector multiplication of length-w vectors (Figure 4(j), which remains the same for both COPSE runtime and VESTA runtime.

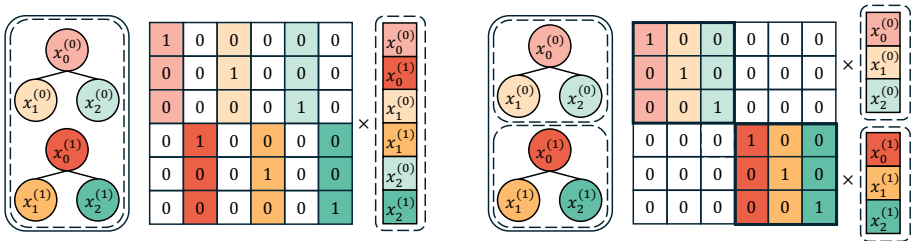
Comparisons in inference accuracy. In general, under consistent precision for input data and thresholds, evaluating the same model using the COPSE system and the VESTA system will yield identical results to those produced by the original plaintext model. Consequently, the inference accuracy of VESTA is equivalent to that of COPSE, both of which match the accuracy of the originally trained model. Methodologically, both the COPSE and VESTA systems utilize an initial comparison step to determine all decisions made at each node with respect to the current input features. The subsequent steps simulate the tree traversal process, mapping these decisions to the corresponding leaf nodes to identify which leaf is reached. Since the comparison step in both systems is identical, and each system can utilize its own reorder matrices to select decisions and map them to the appropriate leaf, the output `Labels` vectors will be the same. Additionally, the properties of FHE ensure that operations performed on encrypted data are equivalent to those conducted on the corresponding plaintext. Furthermore, encryption and decryption do not alter the underlying values. As a result, the output of the secure compare protocol for both systems will align with the comparison results obtained from the plaintext model. Consequently, both the COPSE and VESTA systems demonstrate the same accuracy, which is equivalent to that of the original model.

Comparison in Security Level. In general, the VESTA system possesses the same security properties as the COPSE system, as it does not disclose any additional information compared to the COPSE system and effectively safeguards both model privacy and data privacy. Specifically, from the standpoint of information leakage, certain parameters related to the tree ensemble model may be inadvertently revealed through the shape of vectorized model components. Consequently, the parameters b (internal node number), w (leaf number), and d (level number) are inevitable to leakage when utilizing the COPSE system. Since the components within the VESTA models have the same shape as those with similar names in the COPSE models, no additional parameters will be leaked from the VESTA system. Furthermore, all operations performed during VESTA runtime can follow the same manner as those in COPSE runtime, thus no additional security risks are introduced. From a privacy protection standpoint, the VESTA system encodes sensitive model structures, threshold values, and input data values into encrypted vectors and matrices, and subsequently performs fixed-time branch-less homomorphic computations to get an encrypted output. In this way, both model privacy and data privacy are maintained under all circumstances.

5 Enabling Runtime Batching Through Model Partitioning

This section introduces how we designed the VESTA partitioner to reduce the memory footprint and speed up the inference process. First, we will analyze the influence of partitioning on reorder matrices in the vectorized model, and then we will discuss the design of the VESTA partitioner.

5.1 Key Insight



(a) Reorder step without partition (b) Reorder step with partition

Fig. 6. Impact of Partitioning on the Reordering Matrix

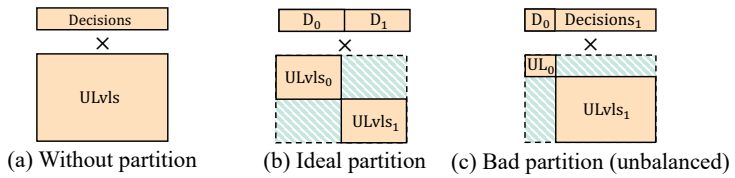
As the vectorized model adapted FHE ciphertext packing does not support random access, one needs to use reorder matrices to arrange values to new vectors. Hence, the inference of the tree ensemble requires a number of reorder matrices as introduced in Section 3. For example, in VESTA runtime, level processing uses reorder matrices `ULvls` to arrange evaluated values of internal nodes stored in *Decisions* into new vectors *LvlDecisions* (Figure 4(g)).

Figure 6 illustrates how partitioning affects the reorder matrices. As illustrated in Figure 6(a), in the input vector of the COPSE reordering step (Figure 4(f)), the decision results contained in *Decisions* are organized based on the features used for comparison (i.e. compare with x_0, x_1, and then x_2). However, the desired output vector of this step, *Branches* for COPSE runtime $[x_0^{(0)}, x_1^{(0)}, x_2^{(0)}, x_0^{(1)}, x_1^{(1)}, x_2^{(1)}]$, is structured according to the preorder tree traversal order, which is determined by the tree structure. Consequently, the potential non-zero values will be distributed across the `Reshuf` matrix. After partitioning, as shown in Figure 6(b), the potential non-zero values are closely clustered in a specific area, allowing for a reduction in the size of the reordering matrix by retaining only that specific area. The same effect occurs in the VESTA runtime algorithm, where the *Decisions* vector is reordered into the *LvlDecisions* vector.

In summary, partitioning improves the *locality* of tree ensemble model information stored in vectorized model data structures, allowing for a reduction in reorder matrix size by only keeping the area containing non-zero values.

5.2 Design of Partitioner

Observation. While considering the model partitioning process, the goal is to reduce the size of reorder matrices contained in vectorized model to the largest extend. Based on our observation, the ideal partition that minimizes memory usage and computation workload will divide the tree ensemble model into exactly equal-sized halves. This type of optimal partition is referred to as "**sub-model balance**".



(a) Without partition (b) Ideal partition (c) Bad partition (unbalanced)

Fig. 7. Influence of partition on VESTA runtime computation

In Figure 7, we illustrate the influence of sub-model balance on reducing the size of reorder matrices by considering the matrix-vector multiplication involved in the level processing step of the VESTA runtime (Figure 4(g)). The green shaded area indicates that up to 50% of the original reorder matrix can be reduced when the sub-models are of exactly the same size (Figure 7(b)). However, if the partition results in unbalanced sub-model sizes (Figure 7(c)), the reduced area decreases. Consequently, the partitioning of the tree ensemble model should strive to achieve sub-model balance. As previously mentioned, the sizes of two systems' reorder matrices, `Reshuf`, `Lvls`, `ULvls`, are all related to internal node number b. Therefore, an optimal partition that achieves sub-model balance can be determined by tallying the internal nodes in each decision tree and grouping them into sub-models with equivalent total numbers of internal nodes.

How do We Partition? As illustrated in Figure 2, when a large, trained tree ensemble model is provided to the VESTA compiler, the VESTA partitioner pass will perform partitioning on the trained plaintext model before the compilation process. In the process of partitioning a tree ensemble model comprising n decision trees with b nodes in total, into m sub-models, it is essential to achieve sub-model balance. Consequently, each partitioned sub-model should contain approximately $\frac{b}{m}$ nodes. Thus, the VESTA partitioner operates as follows: First, it processes the input models to determine the number of nodes in each decision tree and the total number of nodes in the current tree ensemble model (b). Second, it groups the n decision trees into m sets, ensuring that the total number of nodes in each set is either exactly or closely approximates b/m. Finally, it converts the decision trees within each set into a sub-model.

To identify a set of decision trees with a total number of nodes close to $\frac{b}{m}$, due to the offline preprocessing nature of the partition task, a **grid search** approach can be employed to test every possible grouping result until a desired one is found. While this method guarantees a sub-model balance partition, the preprocessing time may become prohibitively long as n and m increase. In practice, considering the typical training process of tree ensemble models and the interpretability of decision tree models, we find that sizes of individual decision trees within the ensemble are similar to each other. With this observation, a naive **random partition** algorithm can be proposed, which randomly selecting $\frac{n}{m}$ trees to form a sub-model. Although the naive algorithm may not guarantee sub-model balance, empirically it yields a partition that is relatively close to the ideal one. In our proposed experiments, the memory usage and computation time required for evaluating a randomly partitioned model exhibit negligible differences when compared to those needed for evaluating a partitioned model using the grid search method. Therefore, we have decided to utilize this method in our VESTA partitioner pass.

Accuracy. It is worth mentioning that partitioning will not affect the accuracy of the partitioned models. While evaluating a partitioned model, Diane will receive m `Labels` corresponding to each sub-model. Since no modifications have been made to any decision tree, the evaluation results of the sub-models will be a subset of the original model's evaluation results. Therefore, the majority voting outcome among all sub-results will remain unchanged.

5.3 How Many Sub-models do We Partition?

To support level processing, the **minimal partition unit** is a single decision tree contained in the tree ensemble model. Therefore, the maximum number of sub-models equals to the number of decision trees, where each sub-model contains one decision tree. The number of sub-models, m, is a hyper-parameter of the partitioner. The selection of m can be based on several aspects, including memory, and security considerations. For example:

Memory and Computation Performance. In general, an increase in the number of sub-models, m, leads to a reduction in both model size and runtime memory consumption due to the mathematical properties of the reorder matrix. Therefore, to run an extremely large tree ensemble model on a computer with limited memory, it is advisable to set a sufficiently large m. Besides, tree ensemble model partitioning also allows the server to exploit task-level parallelism as sub-models could be inferences independently.

Security. As the partitioning occurs initially, no changes are made to the protocol. Thus, the security level of a partitioned sub-model is equivalent to that of the original model. However, under the partitioned model evaluation setting described in Section 3.3, Diane is required to prepare input `Feats` and receive the corresponding output `Labels` for each sub-model. So, Diane will have knowledge of the exact values of b and w for each sub-model, as well as the value of m. Fortunately, none of this information will compromise the protection of model privacy or data privacy defined in Section 2.4.

6 Evaluation Methodology

This section presents the evaluation methodology used to compare our VESTA with the state-of-the-art COPSE system, focusing on computational and memory efficiency.

6.1 System Implementation and Discussion

Implementation of the VESTA Compiler and Partitioning Pass. The VESTA compiler processes the original tree ensemble models and outputs vectorized models after performing compile-time precomputations. A user-defined hyperparameter m (introduced in Section 5) controls the number of sub-models generated by the partitioning pass. This partitioning pass is general and can be embedded into either VESTA compiler or COPSE compiler; we will evaluate both configurations.

Implementation of VESTA Runtime. The VESTA runtime follows COPSE's settings, using the BGV scheme for FHE and HELib for homomorphic operations, with identical FHE configurations and encryption parameters. We use SecComp [7] for secure comparison. To highlight VESTA improvements, all computation kernels, such as matrix-vector multiplication and vector addition, are identical to COPSE's implementation. HELib's BGV supports ciphertext packing, and NTL [54] enables parallel computations for reductions and matrix-vector multiplication.

Generality of VESTA. The design of the VESTA system is highly flexible, supporting implementation in any programming language and with any FHE scheme. The VESTA runtime includes SecComp, matrix-vector multiplication, vector addition, and point-wise multiplication, all implemented using homomorphic addition and multiplication over scalar values. The partitioner pass preprocesses the trained tree ensemble model, further enhancing flexibility. As a result, VESTA operates more as a protocol than a fixed system, adaptable to various programs and FHE schemes.

6.2 Experimental Setup

Configurations. To evaluate the effect on utilizing parallelism in hardware, we use four experimental configurations. Table 1 outlines the four evaluated configurations. The main difference between the four schemes is whether multi-threading or multi-processing is enabled. Specifically, the **multi-threading execution** is enabled using the NTL multi-threading execution to assign the computation work contained in pre-defined kernels, such as matrix-vector multiplication, to all available threads and perform parallel computation (i.e. data parallelism). **Multi-process execution** assigns each sub-model (determined by the hyperparameter m) to a separate process, with each process bound to a distinct core, enabling task-level parallelism.

Table 1. Description of the evaluated configurations

Setting	Multi-threading	Multi-process
S1	✓	✓
S2	×	✓
S3	✓	×
S4	×	×

Specifically, the **S1** setting enables the use of all optimization techniques adapted in our VESTA system, demonstrating the case when we exploit all available parallelism. The **S2** setting exclusively leverages task-level parallelism achieved through partitioning with multi-process execution, highlighting the optimization facilitated by the VESTA partitioner. The **S3** and **S4** settings were initially utilized in COPSE system evaluation.

Evaluation Methodology. We evaluate two key metrics: end-to-end execution time and peak memory usage. For partitioned models with multi-process execution, we measure the total execution time across all concurrently running processes. The memory footprint is captured by recording the maximum resident set size across the entire computation process, including all sub-model inferences.

Evaluated Models. The models used to evaluate the VESTA system contain all trained tree ensemble models used in COPSE [38]. These models consist of two types: the synthetic models and real-world models. The synthetic models vary in the number of levels, number of branches, and the precision used for expressing thresholds. The real-world benchmark models are trained from open-source machine learning datasets.

Hardware Platform. All experiments were performed on a 32-core, 2.0 GHz Intel Xeon Gold 6338 server with 1 TB of RAM. Each core has 1.25 MB of L2 cache, and all 32 cores share a 48 MB last-level-cache (LLC).

7 Experiment Results

7.1 Speedup and Memory Optimization

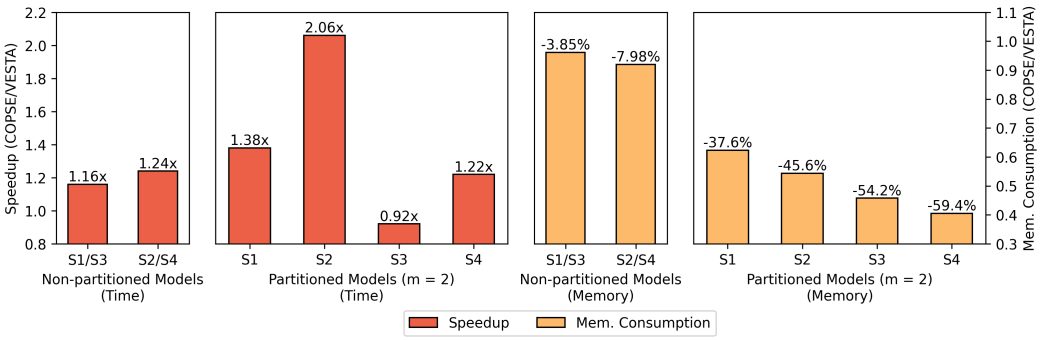


Fig. 8. Average speedup and memory consumption reduction achieved by VESTA compared with COPSE across the evaluated configurations

Figure 8 illustrates the average speedup and reduction in memory consumption attained when utilizing the VESTA system to evaluate all models with and without partitioning across aforementioned four settings. In general, the VESTA system enhances both computational and memory efficiency when evaluating non-partitioned models across all four settings. For partitioned models, the VESTA system can optimize memory and speed at the same time under settings S1, S2, and S4, while striking a balance in setting S3 to notably decrease memory usage with a slight reduction in speed. Here, we will briefly analyze how the VESTA system achieved these optimizations.

Effect of Compile-time Precomputation. Generally, the optimization presented here is attributed to the VESTA runtime, which eliminates the reordering step present in the COPSE runtime without any additional trade-offs. Specifically, the **speedup** achieved is positively correlated with the time taken by the original reordering step. When multi-thread execution is enabled, a smaller percentage of time is allocated to computing well-parallel matrix-vector multiplication compared to SecComp and reduction-like operations. Therefore, the speedup is more pronounced in the S2 and S4 experiments.

Additionally, the reduced **memory consumption** is closely tied to the memory allocation for the matrix operand `Reshuf` and the intermediate result of the original reordering step. The size of `Reshuf` remains constant irrespective of the multi-thread setting, whereas multi-thread execution leads to a higher overall memory usage. Consequently, the enhancement in memory efficiency is more pronounced in the S2 and S4 experiments.

Effect of Runtime Batching. In general, when utilizing the VESTA partitioner to split a large real-world model into two sub-models and evaluate them, the VESTA system can consistently reduce memory consumption by approximately half. However, the speedup achieved through partitioning depends on whether the additional task-level parallelism is effectively utilized or not. Regarding **speed**, when the additional task-level parallelism is leveraged (S1 and S2), the speedup achieved is more notable in comparison to the non-partitioned case. In the case of S1, where partial parallelism is utilized alongside multi-thread execution, the speedup is less than the theoretical 2× speedup seen in the S2 case. In the S3 and S4 scenarios where partitioned models are evaluated sequentially, despite the reduction in size of all matrix operands due to partitioning, the overall computation time still increases compared to the non-partitioned cases. (Further elaboration will be provided in the subsequent paragraph.) Regarding memory utilization, the VESTA system achieves a substantial reduction across all four settings, surpassing the theoretical 50% threshold in settings S3 and S4. This notable outcome is attributed to the combination of smaller partitioned sub-models and the elimination of `Reshuf`. It is worth noting that the utilization of multi-thread and multi-process execution necessitates larger memory compared to serial cases, thereby influencing the effectiveness of partitioning across the four settings.

Discussion: The reason for the decrease in speed under S3 and S4 setting. In general, the reduction in speed observed in settings S3 and S4 compared to non-partitioned cases can be attributed to the implementation of packed ciphertext and SecComp. Specifically, the adaptation of ciphertext packing results in the computation complexity of matrix-vector multiplication being determined by the number of rows in the matrix operand. Despite partitioning reducing the size of the plaintext matrix operand, the total number of packed ciphertexts involved in the computation, or in other words, the total number of rows in all sub-matrices, remains the same. However, evaluating two sub-models necessitates two SecComp invocations, the complexity of which is determined by precision rather than input vector size. Therefore, each partition incurs an extra SecComp invocation as a penalty, leading to an increase in overall computation time when the additional parallelism is not utilized. It is important to note that the speedup provided by the VESTA runtime persists when

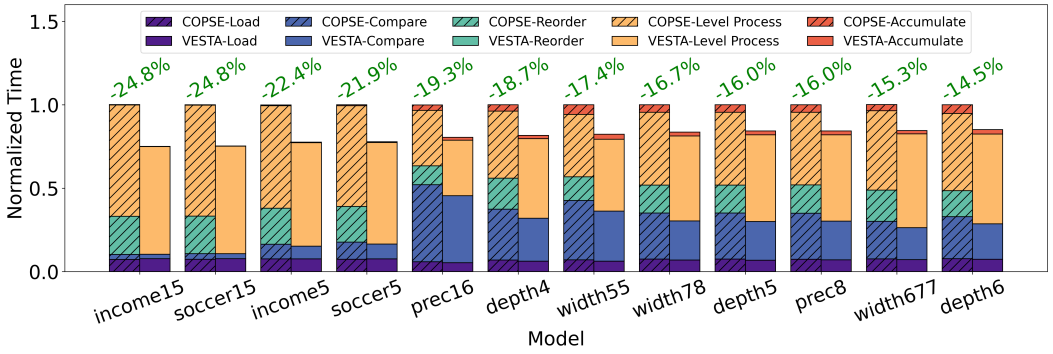


Fig. 9. Comparison of computation time of each step while evaluating non-partitioned models: COPSE runtime versus VESTA runtime (under S2 setting)

evaluating partitioned models. In the case of multi-thread execution (S3), the reduction in the reordering step is outweighed by the addition of SecComp invocations, resulting in a negative impact on speed. Conversely, in setting S4, where the reordering step is more computationally intensive than SecComp, a speedup is still achieved.

7.2 Example: How Time and Memory Efficiency are Improved under S2 Setting

Since the VESTA system demonstrates the most notable enhancements in computation and memory efficiency on average under the S2 setting, we will provide detailed experiment results conducted under the S2 setting to showcase the effectiveness of the VESTA runtime and VESTA partitioner in achieving speedup and memory optimization.

How VESTA runtime speedups the inference process under S2 setting. Figure 9 displays the normalized time required for each computation step of the VESTA runtime in comparison to the COPSE runtime when evaluating all models **without partitioning** under S2 setting. Here, we would like to emphasize two observations: firstly, the experimental results presented here clearly demonstrate that the removal of the reorder step in the VESTA runtime does not incur any additional trade-offs. In other words, the computation time required for each step of the VESTA runtime is no greater than that of the corresponding step in the COPSE runtime. Secondly, the VESTA runtime achieves a higher speedup when evaluating large real-world models compared to smaller microbenchmark models. This is because using the original COPSE runtime to evaluate large models would require more time for the reordering step, whereas the elimination of this step in the VESTA runtime leads to a significant speedup.

How partitioning improves the computation speed and memory efficiency under S2 setting. Figure 10 and Figure 11 illustrate the speedup and reduction in memory consumption achieved by using the VESTA partitioner to divide real-world models with different m settings and then evaluating the sub-models using VESTA runtime in comparison to the COPSE system under the S2 setting. Here, we would like to emphasize two observations: firstly, it is important to note that partitioning can greatly enhance both computation and memory efficiency. In general, partitioning a large model into m sub-models and evaluating them under the S2 setting can result in approximately an m-fold speedup with only $\frac{1}{m}$ of the original memory consumption. Secondly, the sub-models generated by the VESTA partitioner can be evaluated using the COPSE system, resulting in notable speedup and reduction in memory consumption. However, the performance of the COPSE system integrated

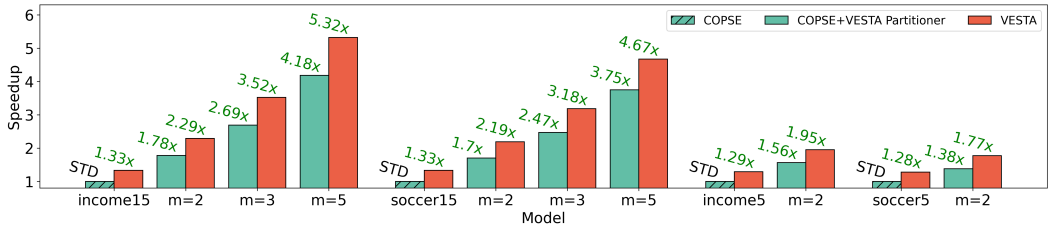


Fig. 10. Comparison of computation speed while evaluating m-partitioned real-world models: COPSE system versus VESTA system (under S2 setting)

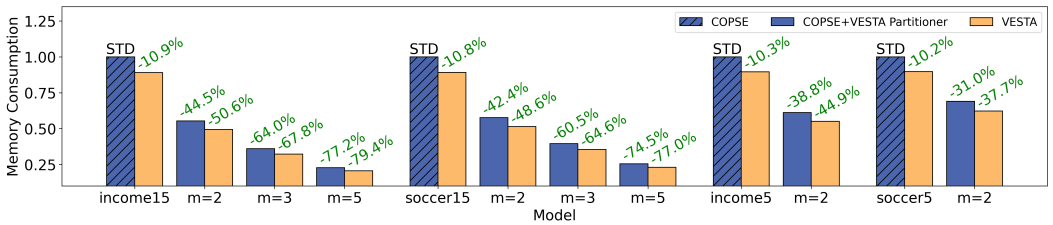


Fig. 11. Comparison of memory consumption while evaluating m-partitioned real-world models: COPSE system versus VESTA system (under S2 setting)

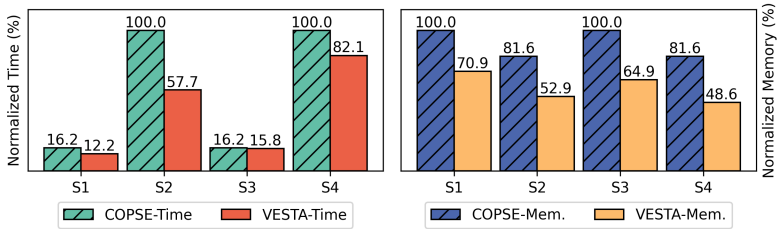


Fig. 12. Comparison of average time and memory consumption for evaluating all models using VESTA system and COPSE system across four settings

with the VESTA partitioner is not as efficient as the standalone VESTA system due to the additional reordering step.

7.3 Discussion: Comparison of Speed and Memory between Four Settings

Figure 12 displays the average time and memory requirements when conducting inference tasks using the VESTA system and the COPSE system across four different settings. Here, we would like to emphasize two observations: Firstly, when a setting enables faster evaluation speed, it usually necessitates more runtime memory, regardless of whether the COPSE system or the VESTA system is being utilized. Secondly, in all settings, utilizing the VESTA system will consistently lead to faster speed and lower memory consumption. The enhancement is notably significant for inference tasks performed in settings that initially exhibit slow evaluation speed or high memory consumption. In summary, the proposed optimizations can enhance computation and memory efficiency without incurring any additional costs, making VESTA consistently effective across the four experimental configurations.

8 Related Work

FHE-based Tree Ensemble Inference. To design an FHE-based secure inference system for tree ensembles, the initial challenge is to devise a protocol that utilizes homomorphic addition and multiplication to carry out the tree walk inference process. Bost et al. [13] first offer a method to encode decision trees into polynomials, thereby facilitating their adaptation for FHE. Building on this concept, Wu et al. [55] develop a two-party secure inference protocol along with an algorithm that enables the comparison of encrypted values. Later, Tai et al. [49] improved the polynomial-like representation by transforming it into linear functions that has less linear dependency. Furthermore, Aloufi et al. [7] propose an enhanced comparison protocol called "SecComp" that reduces the cost required. Up to this point, the computational cost remains exceedingly high due to the sequential nature of polynomial computation. Then, COPSE system [38] is introduced as a three-party secure inference system for tree ensembles, optimizing the computation through vectorization. Its multi-threaded version has achieved state-of-the-art performance, surpassing the previous system more than 10× in terms of speed. When compared to the aforementioned systems, the VESTA system demonstrates the highest performance levels in speed and memory efficiency.

Compilers for FHE Applications. Developing an FHE-based application necessitates expertise in the relevant cryptographic system and meticulous programming to attain satisfactory computational efficiency, thus giving rise to a specialized category of FHE compilers. For instance, various compilers offer domain-specific languages that enable users to write code in plaintext form, which is subsequently converted into a homomorphic version. Notable examples include RAMPARTS [8], HEIR [10], E3 [16], ALCHEMY [21], HECO [52] and Marble [53]. Another category of FHE compilers functions as optimizers that produce efficient homomorphic computation kernels by employing techniques like vectorization. Examples of such compilers include Porcupine [20], EVA [22], Fhelipe [32], the compiler proposed in [33], HECATE [34] and Coyote [37]. Furthermore, certain compilers are specifically tailored for particular FHE-based operations or applications, such as polynomial computation (HEaaN.MLIR [43]) and deep neural networks (nGraph-HE [12], AHEC [15], and CHET [23]). The primary objective of all these compilers is to produce an efficient homomorphic computation kernel based on a known plaintext FHE-friendly implementation. However, they are not applicable for tasks such as tree ensemble inference, which cannot be directly implemented using homomorphic addition and multiplication.

Privacy-preserving Tree Ensemble Inference. Privacy-preserving tree ensemble inference protocols are categorized based on the specific application scenarios they are designed for. Research works [13, 17, 29, 49, 50, 55] introduce early-stage systems that rely on interactive secure comparison protocols. However, interactive secure comparison protocols have high communication complexity, making them unsuitable for low-computation-power servers with limited bandwidth. Recent researches on non-interactive system [6, 19, 36, 58] primarily concentrate on two-party scenarios, where the model owner also serves as the compute server. When comparing the two-party setting with our intended three-party setting, it is evident that the three-party protocol is more intricate. This complexity arises from the challenge of safeguarding model privacy from the third-party server involved in the computation. Hence, the existing systems cannot serve as substitutes for our VESTA system.

9 Conclusions

In this paper, we present VESTA, an FHE-based three-party secure inference system for tree ensemble models. To reduce inference time and memory consumption, we propose *Compile-time Precomputation* and *Runtime Batching*. The former converts an expensive runtime step computed in

ciphertext format into compile-time plaintext precomputation, while the latter partitions the tree ensemble model into sub-models and executes them independently in batches. Evaluation results show that the two techniques reduce both computational complexity and memory consumption significantly, with the same level of security. Our VESTA system is designed to be flexible, general, and efficient. It only requires support for two fundamental operations, addition and scalar multiplication, which are available in all FHE schemes.

To the best of our knowledge, our VESTA system offers a practical solution for three-party secure evaluation of tree ensembles, achieving the fastest computation speed and the smallest memory footprint. Its design philosophy, which involves precomputing costly runtime operations during offline compilation through a compiler-runtime co-design, could serve as a paradigm for the development of other privacy-preserving machine learning systems.

Acknowledgments

We sincerely thank our shepherd, Jun Zhao, and the anonymous reviewers for their insightful feedback, which greatly enhanced the quality of this paper. Haosong Zhao, Zihang Chen, Kunxiong Zhu, and Hongyuan Liu were supported by the National Natural Science Foundation of China (62302416). Zhuoran Ji was supported by the Natural Science Foundation of China (62402282) and the Shandong Provincial Natural Science Foundation (ZR2024QF237). Junhao Huang and Donglong Chen were supported by Guangdong Provincial Key Laboratory of IRADS, BNU-HKBU United International College (2022B1212010006), Guangdong Basic and Applied Basic Research Foundation (2024A1515011274), Guangdong Province General Universities Key Field Project (New Generation Information Technology) (2023ZDZX1033), and UIC Research Grant (UICR04202401-21). Corresponding author: Hongyuan Liu.

References

[1] 2024. Amazon Machine Learning. https://docs.aws.amazon.com/machine-learning.

[2] 2024. Azure AI. https://azure.microsoft.com/en-us/overview/ai-platform/.

[3] 2024. Google Cloud Vertex AI. https://cloud.google.com/vertex-ai.

[4] Abbas Acar, Hidayet Aksu, A. Selcuk Uluagac, and Mauro Conti. 2018. A Survey on Homomorphic Encryption Schemes: Theory and Implementation. *ACM Comput. Surv.* 51, 4 (2018), 79:1–79:35. doi:10.1145/3214303

[5] Rashmi Agrawal, Leo De Castro, Chiraag Juvekar, Anantha Chandrakasan, Vinod Vaikuntanathan, and Ajay Joshi. 2023. MAD: Memory-Aware Design Techniques for Accelerating Fully Homomorphic Encryption. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*.

[6] Adi Akavia, Max Leibovich, Yehezkel S. Resheff, Roey Ron, Moni Shahar, and Margarita Vald. 2022. Privacy-Preserving Decision Trees Training and Prediction. *ACM Trans. Priv. Secur.* 25, 3 (2022), 24:1–24:30. doi:10.1145/3517197

[7] Asma Aloufi, Peizhao Hu, Harry W. H. Wong, and Sherman S. M. Chow. 2021. Blindfolded Evaluation of Random Forests with Multi-Key Homomorphic Encryption. *IEEE Trans. Dependable Secur. Comput.* 18, 4 (2021), 1821–1835. doi:10.1109/TDSC.2019.2940020

[8] David W. Archer, José Manuel Calderón Trilla, Jason Dagit, Alex J. Malozemoff, Yuriy Polyakov, Kurt Rohloff, and Gerard W. Ryan. 2019. RAMPARTS: A Programmer-Friendly System for Building Homomorphic Encryption Applications. In *Proceedings of the 7th ACM Workshop on Encrypted Computing & Applied Homomorphic Cryptography, WAHC@CCS 2019, London, UK, November 11-15, 2019*, Michael Brenner, Tancrède Lepoint, and Kurt Rohloff (Eds.). ACM, 57–68. doi:10.1145/3338469.3358945

[9] Jean-Claude Bajard, Paulo Martins, Leonel Sousa, and Vincent Zucca. 2020. Improving the Efficiency of SVM Classification With FHE. *IEEE Trans. Inf. Forensics Secur.* 15 (2020), 1709–1722. doi:10.1109/TIFS.2019.2946097

[10] Song Bian, Zian Zhao, Zhou Zhang, Ran Mao, Kohei Suenaga, Yier Jin, Zhenyu Guan, and Jianwei Liu. 2024. HEIR: A Unified Representation for Cross-Scheme Compilation of Fully Homomorphic Computation. In *31st Annual Network and Distributed System Security Symposium, NDSS 2024, San Diego, California, USA, February 26 - March 1, 2024*. The Internet Society. https://www.ndss-symposium.org/ndss-paper/heir-a-unified-representation-for-cross-scheme-compilation-of-fully-homomorphic-computation/

[11] Gérard Biau and Erwan Scornet. 2016. A random forest guided tour. *Test* 25 (2016), 197–227.

[12] Fabian Boemer, Yixing Lao, Rosario Cammarota, and Casimir Wierzynski. 2019. nGraph-HE: a graph compiler for deep learning on homomorphically encrypted data. In *Proceedings of the 16th ACM International Conference on Computing Frontiers, CF 2019, Alghero, Italy, April 30 - May 2, 2019*, Francesca Palumbo, Michela Becchi, Martin Schulz, and Kento Sato (Eds.). ACM, 3–13. doi:10.1145/3310273.3323047

[13] Raphael Bost, Raluca Ada Popa, Stephen Tu, and Shafi Goldwasser. 2015. Machine Learning Classification over Encrypted Data. In *22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11, 2015*. The Internet Society. https://www.ndss-symposium.org/ndss2015/machine-learning-classification-over-encrypted-data

[14] Zvika Brakerski, Craig Gentry, and Shai Halevi. 2013. Packed Ciphertexts in LWE-Based Homomorphic Encryption. In *Public-Key Cryptography - PKC 2013 - 16th International Conference on Practice and Theory in Public-Key Cryptography, Nara, Japan, February 26 - March 1, 2013. Proceedings (Lecture Notes in Computer Science, Vol. 7778)*, Kaoru Kurosawa and Goichiro Hanaoka (Eds.). Springer, 1–13. doi:10.1007/978-3-642-36362-7_1

[15] Huili Chen, Rosario Cammarota, Felipe Valencia, Francesco Regazzoni, and Farinaz Koushanfar. 2020. AHEC: End-to-end Compiler Framework for Privacy-preserving Machine Learning Acceleration. In *57th ACM/IEEE Design Automation Conference, DAC 2020, San Francisco, CA, USA, July 20-24, 2020*. IEEE, 1–6. doi:10.1109/DAC18072.2020.9218508

[16] Eduardo Chielle, Oleg Mazonka, Nektarios Georgios Tsoutsos, and Michail Maniatakos. 2018. E^3: A Framework for Compiling C++ Programs with Encrypted Operands. *IACR Cryptol. ePrint Arch.* (2018), 1013. https://eprint.iacr.org/2018/1013

[17] Martine De Cock, Rafael Dowsley, Caleb Horst, Raj S. Katti, Anderson C. A. Nascimento, Wing-Sea Poon, and Stacey Truex. 2019. Efficient and Private Scoring of Decision Trees, Support Vector Machines and Logistic Regression Models Based on Pre-Computation. *IEEE Trans. Dependable Secur. Comput.* 16, 2 (2019), 217–230. doi:10.1109/TDSC.2017.2679189

[18] Ben Collins and Brandy Zadrozny. 2020. Cyberattack Hits Major U.S. Hospital System. https://www.nbcnews.com/tech/security/cyberattack-hits-major-u-s-hospital-system-n1241254.

[19] Kelong Cong, Debajyoti Das, Jeongeun Park, and Hilder V. L. Pereira. 2022. SortingHat: Efficient Private Decision Tree Evaluation via Homomorphic Encryption and Transciphering. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*, Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi (Eds.). ACM, 563–577. doi:10.1145/3548606.3560702

[20] Meghan Cowan, Deeksha Dangwal, Armin Alaghi, Caroline Trippel, Vincent T. Lee, and Brandon Reagen. 2021. Porcupine: a synthesizing compiler for vectorized homomorphic encryption. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation (PLDI)*.

[21] Eric Crockett, Chris Peikert, and Chad Sharp. 2018. ALCHEMY: A Language and Compiler for Homomorphic Encryption Made easY. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang (Eds.). ACM, 1020–1037. doi:10.1145/3243734.3243828

[22] Roshan Dathathri, Blagovesta Kostova, Olli Saarikivi, Wei Dai, Kim Laine, and Madan Musuvathi. 2020. EVA: an encrypted vector arithmetic language and compiler for efficient homomorphic computation. In *Proceedings of the 41st ACM SIGPLAN International Conference on Programming Language Design and Implementation, PLDI 2020, London, UK, June 15-20, 2020*, Alastair F. Donaldson and Emina Torlak (Eds.). ACM, 546–561. doi:10.1145/3385412.3386023

[23] Roshan Dathathri, Olli Saarikivi, Hao Chen, Kim Laine, Kristin E. Lauter, Saeed Maleki, Madanlal Musuvathi, and Todd Mytkowicz. 2019. CHET: an optimizing compiler for fully-homomorphic neural-network inferencing. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2019, Phoenix, AZ, USA, June 22-26, 2019*, Kathryn S. McKinley and Kathleen Fisher (Eds.). ACM, 142–156. doi:10.1145/3314221.3314628

[24] Kocev Dragi, Vens Celine, Struyf Jan, and Dzeroski Saso. 2013. Tree Ensembles for Predicting Structured Outputs. *Pattern Recognition* 46, 3 (2013), 817–833.

[25] Jerome H Friedman. 2001. Greedy function approximation: a gradient boosting machine. *Annals of statistics* (2001), 1189–1232.

[26] Leo Grinsztajn, Edouard Oyallon, and Gael Varoquaux. 2022. Why do tree-based models still outperform deep learning on typical tabular data?. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*.

[27] Chris Isidore. 2014. Target Says 70 Million People Hit in Data Breach. https://money.cnn.com/2014/01/10/news/companies/target-hacking/.

[28] Angela Jäschke and Frederik Armknecht. 2018. Unsupervised Machine Learning on Encrypted Data. In *Selected Areas in Cryptography - SAC 2018 - 25th International Conference, Calgary, AB, Canada, August 15-17, 2018, Revised Selected Papers (Lecture Notes in Computer Science, Vol. 11349)*, Carlos Cid and Michael J. Jacobson Jr. (Eds.). Springer, 453–478. doi:10.1007/978-3-030-10970-7_21

[29] Marc Joye and Fariborz Salehi. 2018. Private yet Efficient Decision Tree Evaluation. In *Data and Applications Security and Privacy XXXII - 32nd Annual IFIP WG 11.3 Conference, DBSec 2018, Bergamo, Italy, July 16-18, 2018, Proceedings (Lecture Notes in Computer Science, Vol. 10980)*, Florian Kerschbaum and Stefano Paraboschi (Eds.). Springer, 243–259.

doi:10.1007/978-3-319-95729-6_16

[30] Dongwoo Kim and Cyril Guyot. 2023. Optimized Privacy-Preserving CNN Inference With Fully Homomorphic Encryption. *IEEE Trans. Inf. Forensics Secur.* 18 (2023), 2175–2187. doi:10.1109/TIFS.2023.3263631

[31] Paul C. Kocher. 1996. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings (Lecture Notes in Computer Science, Vol. 1109)*, Neal Koblitz (Ed.). Springer, 104–113.

[32] Aleksandar Krastev, Nikola Samardzic, Simon Langowski, Srinivas Devadas, and Daniel Sánchez. 2024. A Tensor Compiler with Automatic Data Packing for Simple and Efficient Fully Homomorphic Encryption. *Proc. ACM Program. Lang.* 8, PLDI (2024), 126–150. doi:10.1145/3656382

[33] DongKwon Lee, Woosuk Lee, Hakjoo Oh, and Kwangkeun Yi. 2020. Optimizing homomorphic evaluation circuits by program synthesis and term rewriting. In *Proceedings of the 41st ACM SIGPLAN International Conference on Programming Language Design and Implementation, PLDI 2020, London, UK, June 15-20, 2020*, Alastair F. Donaldson and Emina Torlak (Eds.). ACM, 503–518. doi:10.1145/3385412.3385996

[34] Yongwoo Lee, Seonyeong Heo, Seonyoung Cheon, Shinnung Jeong, Changsu Kim, Eunkyung Kim, Dongyoon Lee, and Hanjun Kim. 2022. HECATE: Performance-Aware Scale Optimization for Homomorphic Encryption Compiler. In *IEEE/ACM International Symposium on Code Generation and Optimization, CGO 2022, Seoul, Korea, Republic of, April 2-6, 2022*, Jae W. Lee, Sebastian Hack, and Tatiana Shpeisman (Eds.). IEEE, 193–204. doi:10.1109/CGO53902.2022.9741265

[35] Bo Liu, Ming Ding, Sina Shaham, Wenny Rahayu, Farhad Farokhi, and Zihuai Lin. 2022. When Machine Learning Meets Privacy: A Survey and Outlook. *ACM Comput. Surv.* 54, 2 (2022), 31:1–31:36. doi:10.1145/3436755

[36] Rasoul Akhavan Mahdavi, Haoyan Ni, Dimitry Linkov, and Florian Kerschbaum. 2023. Level Up: Private Non-Interactive Decision Tree Evaluation using Levelled Homomorphic Encryption. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security, CCS 2023, Copenhagen, Denmark, November 26-30, 2023*, Weizhi Meng, Christian Damsgaard Jensen, Cas Cremers, and Engin Kirda (Eds.). ACM, 2945–2958. doi:10.1145/3576915.3623095

[37] Raghav Malik, Kabir Sheth, and Milind Kulkarni. 2023. Coyote: A Compiler for Vectorizing Encrypted Arithmetic Circuits. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3, ASPLOS 2023, Vancouver, BC, Canada, March 25-29, 2023*, Tor M. Aamodt, Natalie D. Enright Jerger, and Michael M. Swift (Eds.). ACM, 118–133. doi:10.1145/3582016.3582057

[38] Raghav Malik, Vidush Singhal, Benjamin Gottfried, and Milind Kulkarni. 2021. Vectorized secure evaluation of decision forests. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*. 1049–1063.

[39] Souhail Meftah, Benjamin Hong Meng Tan, Chan Fook Mun, Khin Mi Mi Aung, Bharadwaj Veeravalli, and Vijay Chandrasekhar. 2021. DOReN: Toward Efficient Deep Convolutional Neural Networks with Fully Homomorphic Encryption. *IEEE Trans. Inf. Forensics Secur.* 16 (2021), 3740–3752. doi:10.1109/TIFS.2021.3090959

[40] Velibor V. Misic. 2020. Optimization of Tree Ensembles. *Operations Research* 68, 5 (2020), 1605–1624.

[41] Karthik Nandakumar, Nalini K. Ratha, Sharath Pankanti, and Shai Halevi. 2019. Towards Deep Neural Network Training on Encrypted Data. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops 2019, Long Beach, CA, USA, June 16-20, 2019*. Computer Vision Foundation / IEEE, 40–48. doi:10.1109/CVPRW.2019.00011

[42] Capital One. 2024. Capital One Announces Data Security Incident. https://www.capitalone.com/about/newsroom/capital-one-announces-data-security-incident/.

[43] Sunjae Park, Woosung Song, Seunghyeon Nam, Hyeongyu Kim, Junbum Shin, and Juneyoung Lee. 2023. HEaaN.MLIR: An Optimizing Compiler for Fast Ring-Based Homomorphic Encryption. *Proc. ACM Program. Lang.* 7, PLDI (2023), 196–220. doi:10.1145/3591228

[44] Andrew Paverd, Andrew Martin, and Ian Brown. 2014. Modelling and Automatically Analysing Privacy Properties for Honest-but-curious Adversaries. *Tech. Rep* (2014).

[45] Nicole Perlroth. 2019. Capital One Data Breach Compromises 106 Million People. *The New York Times* (2019). https://www.nytimes.com/2019/07/29/business/capital-one-data-breach-hacked.html

[46] Daniel Pop. 2016. Machine learning and cloud computing: Survey of distributed and saas solutions. *arXiv preprint arXiv:1603.08767* (2016).

[47] Yan-Yan Song and LU Ying. 2015. Decision tree methods: applications for classification and prediction. *Shanghai archives of psychiatry* 27, 2 (2015), 130.

[48] Xiaoqiang Sun, Peng Zhang, Joseph K. Liu, Jianping Yu, and Weixin Xie. 2020. Private Machine Learning Classification Based on Fully Homomorphic Encryption. *IEEE Trans. Emerg. Top. Comput.* 8, 2 (2020), 352–364. doi:10.1109/TETC.2018.2794611

[49] Raymond K. H. Tai, Jack P. K. Ma, Yongjun Zhao, and Sherman S. M. Chow. 2017. Privacy-Preserving Decision Trees Evaluation via Linear Functions. In *Computer Security - ESORICS 2017 - 22nd European Symposium on Research in Computer Security, Oslo, Norway, September 11-15, 2017, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 10493)*, Simon N. Foley, Dieter Gollmann, and Einar Snekkenes (Eds.). Springer, 494–512. doi:10.1007/978-3-319-66399-9_27

[50] Anselme Tueno, Florian Kerschbaum, and Stefan Katzenbeisser. 2019. Private Evaluation of Decision Trees using Sublinear Cost. *Proc. Priv. Enhancing Technol.* 2019, 1 (2019), 266–286. doi:10.2478/POPETS-2019-0015

[51] Joost Verbraeken, Matthijs Wolting, Jonathan Katzy, Jeroen Kloppenburg, Tim Verbelen, and Jan S. Rellermeyer. 2021. A Survey on Distributed Machine Learning. *ACM Comput. Surv.* 53, 2 (2021), 30:1–30:33. doi:10.1145/3377454

[52] Alexander Viand, Patrick Jattke, Miro Haller, and Anwar Hithnawi. 2023. HECO: Fully Homomorphic Encryption Compiler. In *32nd USENIX Security Symposium, USENIX Security 2023, Anaheim, CA, USA, August 9-11, 2023*, Joseph A. Calandrino and Carmela Troncoso (Eds.). USENIX Association, 4715–4732. https://www.usenix.org/conference/usenixsecurity23/presentation/viand

[53] Alexander Viand and Hossein Shafagh. 2018. Marble: Making Fully Homomorphic Encryption Accessible to All. In *Proceedings of the 6th Workshop on Encrypted Computing & Applied Homomorphic Cryptography, WAHC@CCS 2018, Toronto, ON, Canada, October 19, 2018*, Michael Brenner and Kurt Rohloff (Eds.). ACM, 49–60. doi:10.1145/3267973.3267978

[54] Victor Shoup. 2021. *Number Theory Library (NTL).* https://libntl.org

[55] David J. Wu, Tony Feng, Michael Naehrig, and Kristin E. Lauter. 2016. Privately Evaluating Decision Trees and Random Forests. *Proc. Priv. Enhancing Technol.* 2016, 4 (2016), 335–355. doi:10.1515/POPETS-2016-0043

[56] Andrew Chi-Chih Yao. 1986. How to Generate and Exchange Secrets. In *27th Annual Symposium on Foundations of Computer Science, Toronto, Canada, 27-29 October 1986*. IEEE Computer Society, 162–167.

[57] Shuai Yuan, Hongwei Li, Xinyuan Qian, Meng Hao, Yixiao Zhai, and Guowen Xu. 2024. Efficient and Privacy-preserving Outsourcing of Gradient Boosting Decision Tree Inference. *IEEE Transactions on Services Computing* (2024), 1–14. doi:10.1109/TSC.2024.3395928

[58] Shuai Yuan, Hongwei Li, Xinyuan Qian, Meng Hao, Yixiao Zhai, and Guowen Xu. 2024. Efficient and Privacy-preserving Outsourcing of Gradient Boosting Decision Tree Inference. *IEEE Transactions on Services Computing* (2024), 1–14. doi:10.1109/TSC.2024.3395928

Received October 2024; revised January 2025; accepted January 2025