
# Kangaroo: A Private and Amortized Inference Framework over WAN for Large-Scale Decision Tree Evaluation

Wei Xu*, Hui Zhu*✉, Yandong Zheng*, Song Bian†, Ning Sun*, Hao Yuan*, Dengguo Feng‡, and Hui Li*
*Xidian University, {xuwei_1,sunning,yuan_hao}@stu.xidian.edu.cn,
{zhuhui,zhengyandong}@xidian.edu.cn,lihui@mail.xidian.edu.cn
†Beihang University, sbian@buaa.edu.cn
‡School of Cyber Science and Technology, fengdg@263.net

***Abstract*—With the rapid adoption of Models-as-a-Service, concerns about data and model privacy have become increasingly critical. To solve these problems, various privacy-preserving inference schemes have been proposed. In particular, due to the efficiency and interpretability of decision trees, private decision tree evaluation (PDTE) has garnered significant attention. However, existing PDTE schemes suffer from significant limitations: their communication and computation costs scale with the number of trees, the number of nodes, or the tree depth, which makes them inefficient for large-scale models, especially over WAN networks. To address these issues, we propose Kangaroo, a private and amortized decision tree inference framework build upon packed homomorphic encryption. Specifically, we design a novel model hiding and encoding scheme, together with secure feature selection, oblivious comparison, and secure path evaluation protocols, enabling full amortization of the overhead as the number of nodes or trees scales. Furthermore, we enhance the performance and functionality of the framework through optimizations, including same-sharing-for-same-model, latency-aware, and adaptive encoding adjustment strategies. Kangaroo achieves a $14\times$ to $59\times$ performance improvement over state-of-the-art (SOTA) one-round interactive schemes in WAN environments. For large-scale decision tree inference tasks, it delivers a $3\times$ to $44\times$ speedup compared to existing schemes. Notably, Kangaroo enables the evaluation of a random forest with $969$ trees and $411825$ nodes in approximately $60$ ms per tree (amortized) under WAN environments.***

## I. Introduction

With the rapid development of machine learning technologies, "Model as a Service" (MaaS) has been widely applied across various domains [1]–[3]. By deploying models on the server, clients can benefit from convenient and efficient services. Although the approach improves service efficiency, clients are required to upload sensitive personal data, such as medical diagnostic records, financial transaction details, and personal communication information, to the server, which increases the risk of privacy leakage. In contrast, running the model on the client side can effectively protect client's privacy. However, since the models hold significant intellectual property, service providers are often reluctant to share it. Therefore, achieving Model as a Service while ensuring both the client's data privacy and the server's model security has become a critical issue [4], [5].

Decision tree inference, as an intuitive and powerful machine learning tool, has been widely applied in various fields such as medical diagnosis, financial risk assessment, and customer behavior prediction [6]–[8]. Compared to deep neural networks (DNNs), decision trees and ensemble models (e.g., XGBoost, Random Forest) are often preferred in practice for their interpretability, better performance on tabular data, and ease of debugging, which are especially important in applications requiring transparency and regulatory compliance [6]–[8]. To ensure both the client's data privacy and the server's model security, numerous private decision tree evaluation schemes (PDTE) have been proposed [9]–[31]. These schemes usually consist of three steps: secure feature selection, oblivious comparison, and secure path evaluation. They leverage privacy-preserving techniques such as fully homomorphic encryption (FHE) [9]–[12] or secure multi-party computation (MPC) [13]–[31] to realize the three core steps, and have been widely adopted in client-server [9]–[21] and outsourced scenarios [22]–[31].

Existing PDTE schemes offer distinct advantages and have proven effective in safeguarding both model confidentiality and client data privacy. However, these approaches have not been thoroughly evaluated for large-scale decision tree evaluation, particularly in wide-area network (WAN) environments. As task demands grow and accuracy requirements increase, decision tree models are being deployed at larger scales to enhance prediction performance and generalization [32]–[34]. Although some model pruning techniques [35]–[37] have been proposed to reduce the size of decision trees, the tree depth and the total number of nodes often remain substantial. For instance, some pruned models still reach a maximum depth of 30 and consist of up to $10^6$ nodes [36]. These observations highlight the continued importance and relevance of large-scale decision tree evaluation [34]–[36], [38], [39].

Network and Distributed System Security (NDSS) Symposium 2026
23-27 February 2026, San Diego, CA, USA
ISBN 979-8-9919276-8-0
https://dx.doi.org/10.14722/ndss.2026.230892
www.ndss-symposium.org

However, in the current research on PDTE, most experimental evaluations are still limited to small-scale decision trees. This raises important questions about whether existing schemes can scale effectively to support large, deep, and complex models under real-world deployment scenarios, particularly in WAN environments with high latency and limited bandwidth, as commonly encountered in practical applications [40]–[42].

**Motivation:** Based on the above questions, we analyze the current state-of-the-art techniques. According to the number of communication rounds, existing schemes can be divided into depth-round evaluation schemes and constant-round evaluation schemes. Depth-round evaluation schemes [17]–[20], [22]–[24], [29] achieve sublinear decision tree evaluations, significantly reducing computation cost. However, as the depth of the trees increases, these schemes incur a growing number of communication rounds, leading to significant latency in WAN setting. Constant-round schemes [9]–[16], [21], [25]–[31] are immune to such communication cost, but their computation and communication costs grow linearly as the number of trees and decision nodes increases. As a result, the main motivation of this work is to design an efficient constant-round scheme that overcomes communication round limitations while effectively amortizing the costs introduced by large-scale tree models in WAN setting. Moreover, considering that the client-server model can be naturally extended to outsourced scenarios, this work primarily focuses on the former.

### A. Our Contributions

To solve the above problems, we propose Kangaroo, a privacy-preserving and efficient constant-round inference framework for large-scale decision tree evaluation. We leverage packed homomorphic encryption (PHE) to amortize both computation and communication costs. Although some schemes have explored the use of PHE for amortization in decision tree evaluation, they fail to fully exploit its amortization potential and suffer from significant performance degradation [10], [11], [46]. In contrast, Kangaroo introduces a novel set of single-instruction-multiple-data (SIMD) PHE protocols over encoded models, which fully leverage the amortization capability of PHE while significantly improving inference efficiency. The key idea is to treat each coefficient in the PHE ciphertext as a model node and combine it with secret sharing (SS) to achieve full amortization and scalability. The main contributions of this work are summarized as follows.

- ***A new two-party inference framework is proposed for large-scale decision tree evaluation over WAN.*** To the best of our knowledge, Kangaroo is the first scheme that achieves full amortization over PHE and is specifically tailored for efficient large-scale decision tree evaluation in WAN setting. In this framework, a novel model hiding and encoding technique is introduced, where decision tree nodes are mapped to polynomial coefficients to enable efficient amortized inference.
- ***A novel set of secure components is designed to support efficient decision tree evaluation.*** These components include packed feature selection (`PackFeatureSel`), packed oblivious comparison (`PackObliviousCom`), and packed path evaluation (`PackPathEva`), which securely realize the three basic steps of decision tree inference. Each component is optimized to enable effective amortization, substantially minimizing both computation and communication costs.
- ***Several optimization strategies are introduced to enhance the practicality of Kangaroo.*** Leveraging our model hiding and encoding technique, a same-sharing-for-same-model strategy is employed to enable real-time inference responses. Additionally, a latency-aware strategy is adopted to further optimize inference efficiency. Building upon our secure components, an adaptive encoding adjustment strategy is developed to achieve full amortization for large-scale decision tree evaluation.

**Benchmarks:** We have implemented Kangaroo, and the core code is available on GitHub[1]. Compared to state-of-the-art (SOTA) schemes, our secure components demonstrate superior amortization capabilities and eliminate the need for any offline preprocessing. We also conducted extensive experimental evaluations over WAN. Specifically, Kangaroo achieves a $14\times$ to $59\times$ improvement over SOTA one-round interaction schemes on small-scale datasets. For large-scale decision tree evaluations, Kangaroo outperforms existing SOTA schemes by $3\times$ to $44\times$. In evaluating a random forest consisting of $969$ trees and $411825$ nodes, Kangaroo achieves an amortized inference time of approximately $60$ ms per tree.

### B. Related Works

We summarize several representative schemes in TABLE I. Moreover, we provide a comprehensive review of private decision tree evaluation (PDTE) and their core components: oblivious comparison and secure path evaluation. Finally, we summarize the amortization techniques commonly adopted in privacy-preserving applications.

**1) Private Decision Tree Evaluation**

• **Client Server Model (2PC):** Existing 2PC schemes [9]–[21] can be divided into depth-round evaluation schemes [17]–[20] and constant-round schemes [9]–[16], [21]. Among depth-round evaluation schemes, the most representative work is proposed by Ma et al. [19], which enables evaluation by traversing only one path. At each layer, it requires only one $\binom{M}{1}$-oblivious transfer (OT), one conditional OT (COT), and one garbled circuit (GC) operation, which significantly improve the efficiency of the protocol. A limitation of this protocol is that it requires re-initialization of the decision tree in each evaluation to ensure security. In constant-round schemes, Kiss et al. [15] summarize a set of modular components that support the three key steos of decision tree evaluation, opening up a new perspective for the design of constant-round protocols. Subsequently, many researchers shifted their focus to one-round interaction schemes [9]–[12], among which Sortinghat [10] and Levelup [11] stand out as the most representative works.

[1]https://github.com/pigeon-xw/Kangaroo

TABLE I: A representative survey of private decision tree evaluation schemes for single tree.

| | Squirrel [43] | SGBoost [44] | FSSTree [24] | Cheng [31] | Zheng [26] | Zhao [45] | Yuan [23] | Ma [19] | Bai (HE) [20] | Tai (HHH) [14] | Kiss (HGH) [15] | Sorting-Hat [10] | Levelup [11] | Ours |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Primitives** | COT | LHE | RSS,FSS | RSS,FSS | SS,OT, TTP | AHE | AHE, PRF | SS,GC, OT | AHE,SS, OT | AHE | AHE,GC | FHE | FHE | PHE,SS |
| **Round** | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● |
| **Feature** | ○ | ○ | ● | ● | ● | ◐ | ● | ● | ● | ● | ● | ● | ● | ● |
| **Comparison** | ✗ | $O(\tau)$ | $O(D)$ | $O(2^D)$ | $O(2^D)$ | $O(2^D)$ | $O(D)$ | $O(D)$ | $O(D)$ | $O(\tau)$ | $O(\tau)$ | $O(\tau)$ | $O(\tau)$ | $O(1)$ |
| **Path** | COTPath | Poly | OnePath | OnePath* | Poly | OnePath* | OnePath | OnePath | OnePath | Path | Path | Poly | Path | MixPath |
| **Sparse** | ✓ | ✓ | ✓ | ✗ | ✗ | ✗ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| **OneTime** | ✓ | ✓ | ✗ | ✗ | ✓ | ✗ | ✗ | ✗ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| **Amortized** | ✓ | ✗ | ✓ | ✓ | ✓ | ✗ | ✗ | ✓ | ✓ | ✗ | ✓ | ✗ | ✗ | ✓ |
| **Scenario** | Fed | Fed | O-3PC | O-3PC | O-2PC | O-2PC | O-2PC | 2PC, O-2PC | 2PC | 2PC | 2PC | 2PC | 2PC | 2PC, O-2PC |

† Round: communication round, ●: $O(1)$; ●: $O(D)$; ●: $O(D+\log t)$; $D$: the maximum tree depth; $t$: the bit size of feature. Feature: secure feature selection, ○: no support; ◐: no oblivious; ●: oblivious. Comparison: the complexity of comparison, $\tau$: the number of decision node. Path: the path evaluation, COTPath: get weight by COT, Poly: polynomial-based evaluation, OnePath: get weight by one path; OnePath*: get weight by one path after tree permutation; Path: Path cost-based evaluation; MixPath: Combine Path with Poly. Sparse: sparse tree model. OneTime: one-time setup phase. Amortized: calculation amortization. ✓: support; ✗: no support. Fed: inference in federated learning; O: outsourcing. In the schemes [10], [11], we only analyze the operation costs of XCMP.

Both schemes optimize the XCMP protocol [9] and propose other amortizable techniques to accelerate the comparison.

• **Two-party Outsourcing Computation (O-2PC):** Inspired by the design of scheme [15], many O-2PC protocols [19], [25]–[27], [30], [45] achieve outsourced computation by replacing 2PC components [14]–[17], [19]. For example, Zheng et al. [26] adopt the feature selection method from [19] and the MSB bit extraction protocol [47] to optimize the feature selection and comparison. Based on the above optimizations, the communication rounds for comparison in [25] are reduced from $O(t)$ to $O(\log t)$. By introducing a trusted third party (TTP), their scheme supports secure multiplication to enable polynomial-based evaluation as in [16], with a communication round of $O(D)$. However, to conceal the structure of the decision tree, the evaluation must be performed over a complete binary tree. Zhao et al. [45] adopt the private data comparison protocol [48] to perform decision tree evaluation. In this approach, the cloud only decrypts along a single path to obtain a randomized traversal result, with the path protected by a tree permutation mechanism [13], [19]. However, the feature selection is not performed obliviously, as it relies solely on hashing function rather than a formal oblivious selection protocol. Subsequently, Yuan et al. [23] employ a pseudorandom function (PRF) [49] to realize secure feature selection, and leverage the private data comparison protocol [48] along with tree permutation techniques [13], [19] to construct a depth-round evaluation scheme.

• **Other Scenarios:** With the increasing adoption of three-party secure computation [50]–[53] and function secret sharing (FSS) [54]–[56], many decision tree evaluation protocols have been designed based on three non-colluding cloud servers (O-3PC) [22], [24], [28], [29], [31]. These protocols can still be broadly categorized into depth-round [22], [24], [29] and constant-round [28], [29], [31] schemes. Leveraging the benefits of replicated secret sharing (RSS), several of these protocols are further enhanced to provide robustness against malicious cloud servers [22], [24]. Meanwhile, the growing popularity of federated learning [57]–[59] has sparked significant interest in federated decision tree inference (Fed) [43], [44], [60]–[62]. However, these schemes often overlook the privacy of features or comparisons. For instance, the scheme [43] only performs plaintext comparisons over decentralized models and requires $2(\tau+1)$ instances of COT to obtain the oblivious weight. Subsequently, the scheme [44] still does not incorporate feature selection. However, it uses the private comparison protocol [48] and performs polynomial-based evaluation [16] using a leveled homomorphic encryption (LHE) scheme [63].

Despite the large number of PDTE schemes that have been proposed, most of them either overlook the challenges of large-scale decision tree evaluation under WAN setting or exhibit poor performance in such scenarios.

**2) Oblivious Comparison**

As a fundamental building block in decision tree inference, oblivious comparison has been extensively studied and has found broad applications in privacy-preserving machine learning and beyond [9]–[11], [25], [26], [46]–[48], [53], [64], [65]. Based on their communication complexity, existing comparison protocols can be categorized into three types: $O(t)$-round protocols [25], $O(\log t)$-round protocols [26], [47], [64], [66], and $O(1)$-round protocols [9]–[11], [46], [48], [53], [65]. Considering the efficiency requirements in WAN setting, we focus on $O(1)$-round protocols. Among them, Wagh et al. [53] propose a protocol for computing the ReLU function that requires five rounds of communication, resulting in notable communication cost. Furthermore, protocols such as [9]–[11], [46] achieve non-interactive comparison by performing computations entirely over ciphertexts. While these non-interactive

protocols eliminate communication costs, they suffer from significant inefficiencies due to the computation cost associated with fully homomorphic operations. In contrast, the protocols in [48], [65] require only a single round of interaction. However, the protocol in [9] requires bit-wise encryption by DGK, which imposes a heavy bandwidth burden. On the other hand, the protocol proposed by Zheng et al. [25] offers a more balanced trade-off between computation and communication. However, it might be prone to sign errors caused by numerical overflow, and it lacks support for amortized computation, unlike [10], [11], [46], thereby limiting its application and performance in large-scale deployments.

**3) Secure Path Evaluation**

The current secure path evaluation for decision tree inference mainly contains four types: Poly, Path, OnePath, and OnePath$^*$ as TABLE I. The Poly approach requires performing ciphertext multiplications along each path, which demands sufficient multiplicative depth and incurs significant computation cost [10], [12], [16], [21], [25]–[27], [30], [44], [67]. On the other hand, Path approach necessitate aggregating results across all paths, followed by perturbation and random shuffling to obtain the true weight [9], [11], [14], [15], [25], [28], [29]. Such operations are not well-suited for evaluation over packed ciphertexts (PHE) [11], [46]. In particular, only using a single rotation to shuffle might inadvertently reveal the comparison results of individual paths [46]. OnePath$^*$ can reduce the decryption number of Path by enabling single-path evaluation, but its applicability is limited to outsourcing scenarios and relies on a complete binary tree to preserve security [13], [31], [45]. While OnePath achieves sublinear computation cost, its performance deteriorates with increasing tree depth, leading to substantial communication cost [17]–[20], [22]–[24], [29].

**4) Support Amortized Computation**

The amortized computation can be categorized into offline amortization and online amortization. In offline computation, the operations are independent of the inputs and can thus be preprocessed to reduce the cost of online computation. For example, in secret-sharing-based multiplication, the generation of multiplication triples can be performed in advance using TTP, OT, or HE [20], [25], [26], [68]. Similarly, in scenarios involving a large number of OT operations, a small number of public-key-based OTs can be used to extend a much larger number of symmetric-key-based OTs, significantly accelerating the online phase [64], [69]–[71]. In online computation, amortization is typically achieved through packed homomorphic encryption (PHE) with single instruction multiple data (SIMD) support, allowing multiple instances to be processed in parallel in a single execution and significantly improving computational efficiency. This approach has been widely adopted in neural network inference [64], [72]–[74], where it significantly enhances efficiency. Compared to OT-based protocols, PHE reduces communication cost and eliminates the need for any offline phase [74]. However, unlike neural network inference, decision tree evaluation involves strong inter-node dependencies, which limit the effective use of PHE. Consequently, PHE are either underutilized [9], [12], [14], [23], [44], [45] or not fully exploited [10], [11], [46] in PDTE.

## II. Preliminaries

In this section, we introduce decision tree evaluation, packed homomorphic encryption, private data comparison, and path evaluation.

### A. Decision Tree Evaluation

Decision tree is an efficient machine learning model and has been widely used in classification or regression tasks [75]. A decision tree $\mathcal{T}$ contains three types of nodes: root node, internal nodes, and leaf nodes. Both the root node and the internal nodes are decision nodes. For the $n$-th decision node, it owns the $m[n]$-th feature and a split threshold $y_n$, where $m[n] \in [1, \cdots, M]$ and $M$ is the total number of features. For a leaf node, it owns a leaf weight $w$. A binary tree with $\tau$ decision nodes has $\tau + 1$ leaf nodes. Moreover, a full binary tree of depth $D$ has $2^D - 1$ decision nodes and $2^D$ leaf nodes.

Given a feature vector $\mathcal{X} = \{x_1, x_2, \cdots, x_M\}$ with $M$ features, a decision tree evaluation with $\tau$ decision nodes can map the vector into the corresponding leaf weight. The evaluation starts from the root node, and it compares the split threshold $y_n$ of the root node ($n = 1$) with the feature value $x_{m[n]}$, where $m[n]$ is the feature index of the $n$-th decision node, $m[n] \in [1, \cdots, M]$, and $n \in \{1, \cdots, \tau\}$. Depending on whether the comparison result is 0 ($x_{m[n]} < y_n$) or 1 otherwise, the evaluation direction will go left or right to the next decision node, and continue this comparison until obtaining the evaluation result $\mathcal{T}(\mathcal{X}) \in \{w_1, \cdots, w_{\tau+1}\}$. Through the above rules, a decision path $\mathcal{P}$ with $\mathcal{T}(\mathcal{X})$ can be obtained for the input vector $\mathcal{X}$, and the evaluation is finished.

### B. Packed Homomorphic Encryption

Packed homomorphic encryption (PHE), such as BFV [76], BGV [77], CKKS [78], etc, is a homomorphic encryption technology that supports batch processing of multiple data. In this paper, we focus on BFV cryptosystem [76]. It consists of three algorithms, including Key Generation, Encryption, Decryption, and it operates over plaintext space $\mathcal{R}_q = \mathcal{Z}_q[x]/(x^{\mathtt{N}}+1)$ and ciphertext space $\mathcal{R}_Q = \mathcal{Z}_Q[x]/(x^{\mathtt{N}}+1)$, where $q$ is the plaintext modulo and $Q$ is the ciphertext modulo. Moreover, there exist several distributions in BFV as follows, **1)** $\mathcal{D}_1$ is a key distribution; **2)** $\mathcal{D}_2$ is an error distribution; **3)** $\mathcal{D}_Q$ is a uniform random distribution over the ciphertext space $\mathcal{R}_Q$.

- $\mathtt{KeyGen}(\lambda, \mathtt{N})$. Given a security parameter $\lambda$ and a polynomial size $\mathtt{N}$, generate the distributions $\{\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_Q\}$, the private key $\mathtt{s} \leftarrow \mathcal{D}_1$ and the public key $\mathtt{pk} = (\mathtt{b}, \mathtt{a}) = (\mathtt{as} + \mathtt{e}, \mathtt{a})$, where $\mathtt{e} \leftarrow \mathcal{D}_2$ and $\mathtt{a} \leftarrow \mathcal{D}_Q$.
- $\mathtt{Enc}(\mathtt{m}, \mathtt{pk})$. Given a plaintext $\mathtt{m} \in \mathcal{Z}_q^{\mathtt{N}}$, the algorithm encodes $\mathtt{m}$ into a polynomial $\mathbf{m}$ with SIMD [79]. It samples a random polynomial $\mathtt{r} \leftarrow \mathcal{D}_1$ and two noise polynomials $\mathtt{e}^{(1)}, \mathtt{e}^{(2)} \leftarrow \mathcal{D}_2$. Then, it encrypts $\mathbf{m}$ into the ciphertext $\mathtt{c} = [\![\mathtt{m}]\!] = (\mathtt{c}_0, \mathtt{c}_1) = (\mathtt{br} + \mathbf{m} + \mathtt{e}^{(1)}, \mathtt{ar} + \mathtt{e}^{(2)})$. Finally, it returns the ciphertext $\mathtt{c}$.
- $\mathtt{Dec}(\mathtt{c}, \mathtt{s})$. The polynomial $\mathbf{m}$ can be recovered as $\mathbf{m} = \mathtt{c}_0 + \mathtt{c}_1\mathtt{s}$. Finally, it returns the decoded plaintext $\mathtt{m}$.

We utilize $[\![\mathtt{m}_1]\!]+\mathbf{m}_2$, $[\![\mathtt{m}_1]\!]\circ\mathbf{m}_2$, $[\![\mathtt{m}_1]\!]+[\![\mathtt{m}_2]\!]$, and $[\![\mathtt{m}_1]\!]\circ[\![\mathtt{m}_2]\!]$ to represent plaintext-ciphertext addition, plaintext-ciphertext multiplication, ciphertext addition, and ciphertext multiplication, where $[\![\mathtt{m}_1]\!], [\![\mathtt{m}_2]\!]$ are two BFV ciphertexts, $\mathbf{m}_2$ is an encoded plaintext vector, and $\mathtt{m}_1, \mathtt{m}_2 \in \mathcal{Z}_q^{\mathtt{N}}$. We utilize $\mathtt{Rot}(\mathtt{c}, r)$ to represent the ciphertext rotation, where the ciphertext slots are shifted by a random integer $r$. We utilize $\mathtt{plainRot}(\mathtt{m}, r)$ to represent the plaintext rotation. When $r > 0$, the slots are shifted to the right, otherwise left. For simplicity, encoded vectors are denoted using bold uppercase symbols.

### C. Private Data Comparison

The private data comparison has recently gained popularity [44], [45], [48]. It enables one party $S_1$ with $\{[\![\mathtt{m}_1]\!], [\![\mathtt{m}_2]\!]\}$ and the other party $S_2$ with the secret key $s$ to compare $\mathtt{m}_1$ and $\mathtt{m}_2$. Finally, $S_1$ obtains the comparison result. It consists of two steps.

**Step 1.** $S_1$ calculates $[\![\mathtt{m}]\!] = a \cdot ([\![\mathtt{m}_1]\!] - [\![\mathtt{m}_2]\!]) + b$ and sends $[\![\mathtt{m}]\!]$ to $S_2$, where $\{a, b\}$ are selected from the plaintext space, with $a > b > 0$.

**Step 2.** After receiving $[\![\mathtt{m}]\!]$, $S_2$ calculates the comparison result $c$ and sends it to $S_1$. Here, $c = 0$ if $\mathtt{m} < 0$, indicating that $\mathtt{m}_1 < \mathtt{m}_2$, and $c = 1$ if $\mathtt{m} > 0$, indicating that $\mathtt{m}_1 \geq \mathtt{m}_2$.

### D. Path Evaluation

In this section, we introduce the path cost-based evaluation (Path) [14] and polynomial-based evaluation (Poly) [16] in details. The concept of path cost-based evaluation is as follows: given a decision tree model $\mathcal{T}$ and a feature vector $\mathcal{X}$, we determine the comparison result $c_n$ (is $x_{m[n]} < y_n$ ? $0 : 1$) for each decision node. For the $n$-th decision node, the `node->left.cost` is set to $c_n$ and the `node->right.cost` is set to $1 - c_n$. As the cost of each node is assigned a value, the cost from the root node to a corresponding leaf node is defined as the sum of the costs along the path. It can be observed that only the sum of the true path equals zero, whereas the sums of all other paths are greater than zero. Thus, the weight of the leaf node along this path corresponds to the predicted weight. Unlike path cost-based evaluation, the $n$-th `node->left.cost` is set to $1 - c_n$ and the $n$-th `node->right.cost` is set to $c_n$. Moreover, the cost from the root node to a corresponding leaf node is defined as the product of the costs along the path. It can be observed that only the product of the true path equals one, while the products of all other paths are zero.

## III. Framework Overview

In this section, we begin by formulating the key challenges addressed by Kangaroo. We then present a high-level overview of its workflow, followed by a detailed description of our models and corresponding security analysis. TABLE VIII presents the corresponding notations used in Kangaroo.

### A. Technique Challenges and Observations

While prior studies have explored PHE-based model inference [9]–[12], [46], many of these solutions fail to fully exploit the potential of SIMD techniques [9]–[12] or suffer from low packing utilization and limited execution efficiency [10], [11], [46]. For example, the schemes [9]–[11] cannot support the amortized calculation because XCMP cannot support SIMD. t-SortingHat [10] supports only the comparison amortization, but its complexity remains high at $O(\frac{\tau \cdot t}{\log \tau})$. RCC-PDTE [11] and the scheme proposed in [46] support amortization for both comparison and path evaluation. However, due to limitations in numerical precision and path length, these schemes fail to fully leverage the parallelism available in each ciphertext slot. Therefore, we identify two core challenges: amortized utilization and computational efficiency. To enhance the amortization capability of PHE, we should treat each ciphertext slot as a representation of a decision tree node and avoid redundant node representations. Consequently, node information must be encoded into the coefficients of the ciphertexts, enabling efficient feature selection and secure comparison directly over packed ciphertexts. However, even with successful packing, efficient path evaluation over packed ciphertext remains a non-trivial problem, leading to many existing schemes to favor Paillier (AHE) [14], [23], [45] over PHE for constructing constant-round evaluation schemes. In what follows, we elaborate on the technical challenges and observations.

- **Challenge and Observation 1:** When using HE to encrypt each feature eliminates the need for feature selection and enables direct comparison. Therefore, using PHE introduces the first challenge: how to perform parallelized and secure feature selection within packed ciphertexts and efficiently complete the comparison. To tackle this, we quantize the feature vectors and redundantly encode them into polynomials. Furthermore, we design a non-interactive feature selection algorithm to enable batch processing of feature selection across nodes. It supports feature selection across arbitrary dimensions and places the results at fixed positions to hide the features of the nodes. To further enhance the amortization capability of PHE, we encode the nodes of multiple trees into a single polynomial and simultaneously pack multiple selected feature vectors into a single polynomial. This design allows us to treat each coefficient in the polynomial as a unique, non-redundant node, thereby fully leveraging the SIMD parallelism and amortization potential offered by PHE. Based on the packed selected features and thresholds, we integrate secret sharing and propose a packed oblivious comparison protocol to achieve correct and secure comparisons, which solve the limitation of the scheme [48] mentioned in Section I-B. Moreover, our protocol performs comparisons using SIMD in a single round of interaction, which further amortizes both computation and communication costs.
- **Challenge and Observation 2:** After obtaining the encrypted comparison results, the server typically performs either path cost-based evaluation or polynomial-based evaluation to determine the final path selection. For path cost-based evaluation over packed ciphertexts, it is necessary to sum the evaluation results across all nodes along each path, which requires a significant number of rotation and summation operations. Additionally, to conceal the final selected path, the selected weight must undergo extensive rotations, lead-

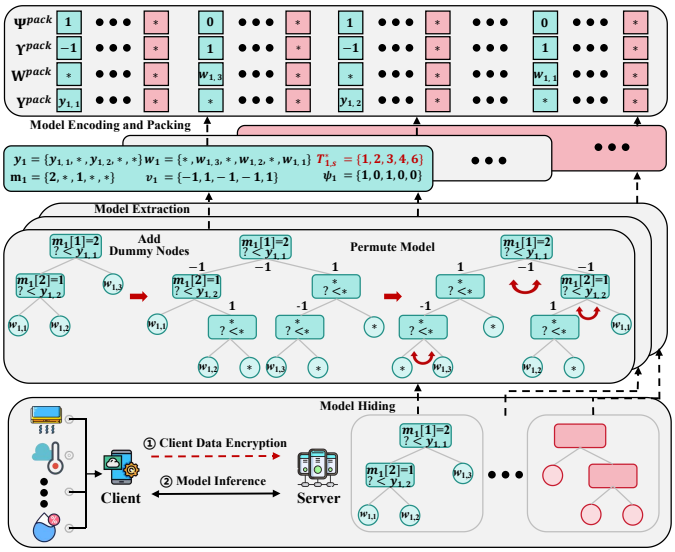



Fig. 1: High-level workflow of Kangaroo for client-server model, where $K = M$.

ing to an impractically high computation cost. In contrast, polynomial-based evaluation can generate a one-hot vector to get the selected weight, eliminating the aforementioned costs. However, it requires a multiplication depth corresponding to the tree depth for each path, and its computation cost cannot be amortized. As a result, both methods impose significant computation cost. To address this challenge, we propose a novel path evaluation protocol that enables the client and server to perform plaintext evaluations on an obfuscated tree model, yielding results similar to those of path cost-based evaluation. Simultaneously, we employ the oblivious comparison protocol to convert the evaluation result into a polynomial representation, thereby avoiding excessive rotations.

### *B. Kangaroo Workflow*

The processing flow of Kangaroo consists of four stages, model hiding and extraction, model encoding and packing, client data encryption, and model inference. Below, we provide a concise overview of the main operations involved in each phase, which are also depicted in Fig. 1.

• **Model Hiding and Extraction:** To finish the path evaluation, the server needs to share the tree model structure with the client. Directly sharing the model structure might expose the server's model privacy [72], [73], [80]. Expanding decision nodes into a full binary tree [13], [17], [25], [26], [45] offers a potential solution but introduces significant computation costs. To address it, we introduce a more efficient solution to hide the model structure. It avoids the need to expand the nodes into a full binary tree, and instead only requires publishing an obfuscated model structure to effectively protect the original model structure. Specifically, the server needs to generate some dummy nodes to hide the model structure. We assume that the total number of decision nodes, after padding with dummy nodes, is $\tau^*$. When a dummy node is created, its left leaf inherits the weight of the parent node, while its right leaf's weight is randomly generated. Then, the server confuses the tree model by a coin-flipping method. For each node, the server flips a coin to randomly select either 1 or $-1$. If $-1$ is selected, the server swaps the left and right child nodes of the given node; otherwise, no change is made.

During model extraction, the server extracts the model parameters of the $k$-th tree, where $1 \le k \le K$. Specifically, the server uses breadth-first to obtain the threshold $\mathbf{y}_k$, feature index $\mathbf{m}_k$, flip condition $\upsilon_k$, model structure index $\mathcal{T}^*_{k,s}$, and status of node $\psi_k$ and uses depth-first traversal to obtain the weight vector $\mathbf{w}_k$ of the leaf nodes for $k$-th tree. In the vector $\psi_k$, 0 indicates a dummy node and 1 indicates a real node. The model structure index $\mathcal{T}^*_{k,s}$ is obtained according to the following rule: for a node with index $i$, its left child index is $2i$, and its right child index is $2i+1$. At last, the model structure indices $\{\mathcal{T}^*_{k,s}\}_{k=1}^K$ are published.

• **Model Encoding and Packing:** To amortize the time cost of feature selection, comparison, and path evaluation, the server encodes the model parameters into a polynomial ring. First, the server chooses the corresponding the maximum and minimum quantization vectors $\mathcal{X}^{max} = \{x_1^{max}, x_2^{max}, \cdots, x_M^{max}\}$ and $\mathcal{X}^{min} = \{x_1^{min}, x_2^{min}, \cdots, x_M^{min}\}$ and the precision parameter $\zeta$. Then, each vector $\mathbf{y}_k$ is quantized by $\mathcal{X}^{max}$ and $\mathcal{X}^{min}$ with the precision parameter $\zeta$ to enable secure comparison. It is worth noting that the server needs to publish $\mathcal{X}^{max}$, $\mathcal{X}^{min}$, and $\zeta$. For non-sensitive features, such as human height, the valid range (e.g., 54.6 cm to 272 cm) can be directly published without privacy concerns. However, for certain sensitive features, the actual range can be deliberately extended to obscure the true value range and enhance privacy protection. Specifically, $\mathbf{y}_k$ is quantized as

$$\mathbf{y}_k = \{\frac{y_k[1] - x^{min}_{m_k[1]}}{x^{max}_{m_k[1]} - x^{min}_{m_k[1]}} \cdot \zeta, \cdots, \frac{y_k[\tau^*] - x^{min}_{m_k[\tau^*]}}{x^{max}_{m_k[\tau^*]} - x^{min}_{m_k[\tau^*]}} \cdot \zeta\}.$$

After it, the vectors $\mathbf{y}_k$, $\upsilon_k$, and $\psi_k$ are encoded into $\{\mathbf{Y}_k, \Upsilon_k, \Psi_k\}$ for $1 \le k \le K$, where each element of the original vectors is placed at positions $\{(n-1)M+1\}_{n=1}^{\tau^*}$. The vector $\mathbf{w}_k$ is encoded into $\mathbf{W}_k$ for $1 \le k \le K$, with elements placed at positions $\{(n-1)M+1\}_{n=1}^{\tau^*+1}$. The vector $\mathbf{m}_k$ is encoded into $\mathbf{M}_k = \{\mathbf{m}_k^1, \mathbf{m}_k^2, \cdots, \mathbf{m}_k^{\tau^*}\}$, where $\mathbf{m}_k^n$ is a vector of $M$ dimensions and $1 \le n \le \tau^*$. The $m_k[n]$-th value of $\mathbf{m}_k^n$ is set to 1, and other values are set to 0.

To further improve the utilization of the polynomial space, the server packs multiple tree models together for efficient inference. Specifically, the server merges the vectors as $\mathbf{Y}_\gamma^{pack} = \sum_{m=1}^M \texttt{plainRot}(\mathbf{Y}_{(\gamma-1)M+m}, m-1)$, $\mathbf{W}_\gamma^{pack} = \sum_{m=1}^M \texttt{plainRot}(\mathbf{W}_{(\gamma-1)M+m}, m-1)$, $\Upsilon_\gamma^{pack} = \sum_{m=1}^M \texttt{plainRot}(\Upsilon_{(\gamma-1)M+m}, m-1)$, and $\Psi_\gamma^{pack} = \sum_{m=1}^M \texttt{plainRot}(\Psi_{(\gamma-1)M+m}, m-1)$. For simplicity, we assume $K = \Gamma \cdot M$ and refer to $\{\mathbf{Y}_\gamma^{pack}, \mathbf{W}_\gamma^{pack}, \Upsilon_\gamma^{pack}, \Psi_\gamma^{pack}\}$ as the packed model, where $1 \le \gamma \le \Gamma$. After getting the packed model and $\{\mathbf{M}_k\}_{k=1}^K$, a same-sharing-for-same-model

strategy is employed to improve the efficiency of evaluation, which is introduced in Section VI-A.

• **Client Data Encryption:** During the data encryption, the client needs to initialize a BFV encryption system and generates the public key to request evaluation services. Then, the client utilizes the public range of the feature vector $\mathcal{X}^{max}$ and $\mathcal{X}^{min}$ to quantize its feature vector $\mathcal{X}$ as

$$\mathcal{X} = \{\frac{x_1 - x_1^{min}}{x_1^{max} - x_1^{min}} \cdot \zeta, \cdots, \frac{x_M - x_M^{min}}{x_M^{max} - x_M^{min}} \cdot \zeta\}.$$

After it, the client repeats the quantized feature vector $\tau^*$ times to an N-dimensional vector $X$, where $X = \{\mathcal{X}, \cdots, \mathcal{X}\}$. We assume $(\tau^* + 1) \cdot M \leq \mathtt{N}$, and N is the polynomial dimension of BFV. At last, the client encrypts the vector and sends $[\![X]\!]$ and the corresponding keys to the server.

• **Model Inference:** During the model inference, the server and the client jointly execute feature selection, comparison, and path evaluation, and the client obtains the inference result. The building blocks involved in the inference contain three steps: ① `PackFeatureSel`, ② `PackObliviousCom`, and ③ `PackPathEva`. They serve as the core operations for performing inference on a single decision tree. `PackFeatureSel` is used to select the corresponding feature over the encrypted and packed feature vector, with the selected values placed in fixed positions to hide the node's feature information. `PackObliviousCom` then performs packed comparisons between feature thresholds and selected features for each node and obtains the encrypted comparison signs. In the signs, 0 indicates that the client's feature is less than the threshold, and 1 otherwise. Finally, `PackPathEva` is applied to obtain the evaluation result based on the encrypted comparison outcomes. The weight corresponding to the true decision path is retrieved, while the weights at all other positions are set to 0. Based on the building blocks, a complete scheme for large-scale evaluation is proposed, along with several optimization techniques to accelerate the process. Moreover, some additional functionalities are presented in the appendix, and interested readers are encouraged to refer to it for more details.

### C. Models and Security

The Kangaroo mainly works in the client-server model. The server holds $K$ decision tree models $\{\mathcal{T}_k\}_{k=1}^K$, and the client holds a feature vector $\mathcal{X}$. Our security goals are to protect the server's models from the semi-honest client and the client's data from the semi-honest server. We consider semi-honest adversaries, which are similar to previous works [9]–[11], [14]–[16], [18]. Specifically, the following private data should be protected.

**1) The Private Data of Server**

① $\mathbf{m}_k$ : The node feature vector is private for server as it reflects the splitting preferences of the decision tree and might expose important feature distributions.

② $\mathbf{y}_k$ : The feature threshold vector is private for server as it determines how data points are classified, and leakage could expose critical decision boundaries.

③ $\mathbf{w}_k$ : The weight vector is private for server as it encapsulates critical information about the model.

④ $\mathcal{T}_{k,s}, \upsilon_k, \psi_k$ : The model structure index, flip condition, and node status are private for server as they represent the server's revenue model, and revealing them could leak some sensitive information about the private dataset used to train the tree model [72], [73], [80].

**2) The Private Data of Client**

⑤ $\mathcal{X}$ : The feature vector is private for client as it contains the client's private input data used for inference.

**3) The Private Data during Inference**

⑥ The evaluation result is private for client as it reveals the models' decision on the client's private feature vector.

⑦ The decision path is sensitive for server and client as it reveals how the client's input traverses each decision tree and exposes the corresponding weights to the client.

⑧ The comparison sign is sensitive for server and client as it might reveal the decision direction.

⑨ The difference of comparison is sensitive for server and client as it directly involves private feature value and its relation to the decision threshold.

**Security of Kangaroo:** To protect the above privacy, the private data of client should be encrypted and the private data of server should be blinded [14], [15], [80]. We present the standard definition of semi-honest security and conduct a rigorous analysis using the simulation-based real/ideal world model [81]. In this setting, the client and server are assumed to strictly follow the protocol but may attempt to infer sensitive information as defined in our threat model. In real-world scenarios, however, a curious client may craft inputs in a malicious way to extract model parameters. To mitigate such risks, it is essential that all intermediate outputs in our protocol remain oblivious [53], [82]. In either of the above cases, our scheme should be designed to robustly protect private data ① - ⑨. Due to space limitation, the details can be found in Section C-A and C-B.

Kangaroo also supports the outsourcing scenarios, which operates in the single-cloud assisted model [45]. The server outsources the model to a cloud service provider (CSP) to offload client-side computation and reduce its own computational burden. It can also be extended to a double-cloud model [19], [25], [26]. However, considering the cost of leasing servers and the security issues associated with the double-cloud model, we have chosen the single-cloud assisted outsourcing scheme. The interested readers can refer to the full version [83] for more details.

## IV. Building Blocks

In this section, we introduce our building blocks for single tree inference, including packed feature selection `PackFeatureSel`, packed oblivious comparison `PackObliviousCom`, and packed path evaluation `PackPathEva`, to achieve the amortized computation.

### A. Packed Feature Selection: `PackFeatureSel`

The `PackFeatureSel` is used to select the corresponding features from the client's packed feature vector. The server

**Algorithm 1** `I-PackFeatureSel`

**SInput:** The encrypted vector $[\![X]\!]$, feature size $M$, and encoded feature index vector $\mathbf{M}$.
**SOutput:** The selected encrypted vector $[\![X']\!]$.

1: ▷ *The server executes:* ◁
2: $[\![X']\!] = [\![X]\!] \circ \mathbf{M}$.
3: $bool = (M > 0 \ \&\ (M \ \&\ (M-1)) == 0)$. ▷ *Determine whether $M$ is a power of 2.*
4: **if** $bool == true$ **then**
5: **for** $i = 0, i < \log M, i++$ **do**
6: $[\![X']\!] = [\![X']\!] + \texttt{Rot}([\![X']\!], -2^i)$.
7: **else**
8: $[\![X'']\!] = [\![X']\!] + \texttt{Rot}([\![X']\!], -1)$.
9: **if** $M \mod 2 == 0$ **then**
10: $[\![X']\!] = [\![X'']\!]$.
11: **for** $i = 1, i < \lceil \log M \rceil - 1, i++$ **do**
12: $M = M - \lfloor \frac{M}{2} \rfloor$.
13: **if** $M \mod 2 == 0$ **then**
14: $[\![X']\!] = [\![X'']\!] + \texttt{Rot}([\![X']\!], -2^i)$.
15: $[\![X'']\!] = [\![X'']\!] + \texttt{Rot}([\![X'']\!], -2^i)$.
16: $[\![X']\!] = [\![X'']\!] + \texttt{Rot}([\![X']\!], -2^{\lceil \log M \rceil - 1})$.
17: The server gets $[\![X']\!]$.

inputs an encrypted vector $[\![X]\!]$, a feature size $M$, and an encoded feature index vector $\mathbf{M}$, and then obtains a selected encrypted vector $[\![X']\!]$. In $[\![X']\!]$, the selected features $x_{m[n]}$ are placed at positions $\{(n-1)M+1\}_{n=1}^{\tau^*}$ to hide the features of the nodes. We give a non-interactive feature selection algorithm `I-PackFeatureSel` in Algorithm 1. First, the server computes $[\![X']\!] = [\![X]\!] \circ \mathbf{M}$ to extract the corresponding feature values from the client at each node as line 2. To accumulate each vector of length $M$ to its first value, the server checks whether $M$ is a power of 2 as line 3. If yes, the server performs $\log M$ rotate-and-sum operations on $[\![X']\!]$ to place the extracted results at positions $\{(n-1)M+1\}_{n=1}^{\tau^*}$ as lines $4-6$. If no, a division-based approach is applied to compute the selected encrypted vector as lines $7-16$. In the process, the server sums every two adjacent data. Therefore, it needs to determine whether $M$ is odd or even for the vector of length $M$. If $M$ is odd, the value at the last position will be retained in $[\![X']\!]$. If $M$ is even, every two adjacent data will be summed. Using the division approach, the server can obtain the sums of the first $\frac{M}{2}$ values into $[\![X'']\!]$ and stores the remaining sums in $[\![X']\!]$. Finally, the server rotates $[\![X']\!]$ and adds it to $[\![X'']\!]$, obtaining the final result. The correctness of `PackFeatureSel` is proven in Theorem 1. Moreover, we present a fully amortized feature selection as Algorithm 2.

### B. Packed Oblivious Comparison: `PackObliviousCom`

The `PackObliviousCom` is used to achieve the oblivious comparison between the node thresholds and client's selected features. The server inputs $[\![X']\!]$ and an encoded vector $\mathbf{Y}$, and then obtains the encrypted comparison result vector $[\![C]\!]$, where $0 \le X'[(n-1)M+1] = x_{m[n]}, Y[(n-1)M+1] = y_n \le \zeta$. In $[\![C]\!]$, $C[(n-1)M+1] = 0$ if $x_{m[n]} - y_n < 0$, 1 otherwise, where $1 \le n \le \tau^*$. The `PackObliviousCom` is given in Fig. 2. First, the server selects two random vectors $A$ and $B$ to blind the difference result of each node as line 1. Then, the server randomly selects either 1 or $-1$ for each node by flipping a coin to protect the result's sign as line 2. Next, the server blinds the difference result $[\![X']\!] - \mathbf{Y}$ and gets $[\![V]\!]$ as line 3. The $[\![V]\!]$ is sent to the client. On receiving the $[\![V]\!]$, the client decrypts it and obtains the blinded comparison result at each node. The client sets $v'_n$ to 0 if $V[(n-1)M+1] < 0$, 1 otherwise, to get the vector $V'$ as line 4. Then, the client encrypts $V'$ and returns it to the server as line 5. After obtaining $[\![V']\!]$, the server generates the vector $C'$ as line 6. At last, the server utilizes $C'$ and $R$ to recover the comparison result vector $[\![C]\!]$. The correctness of `PackObliviousCom` is proven in Theorem 2.

*Remark 1.* It is obvious that our `PackObliviousCom` can also support comparison of two ciphertexts. In addition, we can attach specific values to $*$ for more amortization.

`PackObliviousCom` **Protocol**

**SInput:** The encrypted and encoded vectors $[\![X']\!]$, $\mathbf{Y}$.
**SOutput:** The encrypted comparison result vector $[\![C]\!]$.

▷ *The server executes:*
1: $A \leftarrow \{a_1, *, \cdots, *, \cdots, a_{(\tau^*-1)M+1}, *, \cdots, *\}$, $B \leftarrow \{b_1, *, \cdots, *, \cdots, b_{(\tau^*-1)M+1}, *, \cdots, *\}$, where $\zeta > A[(n-1)M+1] > B[(n-1)M+1] > 0$ and $1 \le n \le \tau^*$. ▷ *Random the differences results.*
2: $R \leftarrow \{r_1, *, \cdots, *, \cdots, r_{(\tau^*-1)M+1}, *, \cdots, *\}$, where $R[(n-1)M+1] \leftarrow \{1, -1\}$ by flipping a coin. ▷ *Random the signs of comparison results.*
3: $[\![V]\!] \leftarrow \mathbf{A} \circ \mathbf{R} \circ ([\![X']\!] - \mathbf{Y}) + \mathbf{B} \circ \mathbf{R} \Rightarrow$ the client.
▷ *The client executes:*
4: $V \leftarrow \texttt{Dec}([\![V]\!], \texttt{s})$, $V' \leftarrow \{v'_1, *, \cdots, *, \cdots, v'_{(\tau^*-1)M+1}, *, \cdots, *\}$, where $V'[(n-1)M+1] = 0$ if $V[(n-1)M+1] < 0$, 1 otherwise.
5: $[\![V']\!] \leftarrow \texttt{Enc}(V', \texttt{pk}) \Rightarrow$ the server.
▷ *The server executes:*
6: $C' \leftarrow \{c'_1, *, \cdots, *, \cdots, c'_{(\tau^*-1)M+1}, *, \cdots, *\}$, where $C'[(n-1)M+1] = 1$ if $R[(n-1)M+1] = -1$, 0 otherwise.
7: The server gets $[\![C]\!] \leftarrow \mathbf{C}' + \mathbf{R} \circ [\![V']\!]$.

Fig. 2: Packed oblivious comparison protocol for single tree.

### C. Packed Path Evaluation: `PackPathEva`

The `PackPathEva` is used to get the evaluation result. The server inputs $[\![C]\!]$, an structure index $\mathcal{T}_s^*$, and an encoded weight vector $\mathbf{W}$, and the client inputs $\mathcal{T}_s^*$. After executing it, the server obtains the encrypted evaluation result vector $[\![T]\!]$. In $T$, only one element is the actual weight value, and the remaining elements are 0. Due to space limitation, the `PackPathEva` is given in Fig. 8. First, the server selects a random vector $R'$ to blind the comparison result vector $[\![C]\!]$.

TABLE II: Online operation costs for two-party inference schemes: $\tau$ : the number of decision nodes; $M$ : the feature dimension; $t$ : the bit size of feature; $K$ : the number of trees; $D$ : the tree depth; $\texttt{Mul}^*$ : plaintext-ciphertext multiplication. SOS: Shard Oblivious Selection Protocol [20]; DGK: Private Comparison Protocol [84].

| | **Primitives** | **Selection** | **Comparison** | **Path evaluation** | **Round** |
|---|---|---|---|---|---|
| Ma [19] (Sparse) | SS,GC,OT | $KD\cdot$(SS, $\binom{M}{1}$-OT) | $KD\cdot$ (GC, $\binom{2}{1}$-OT) | / | 2D-1 |
| Bai [20] (HE-SOS) | SS,OT,AHE | $KD\cdot$ SOS | $KD\cdot$ (GC, SOS) | / | 8D |
| Cong [10] (Sortinghat) | PHE | / | $K\tau\cdot\texttt{Mul}^*$ | $O(K\tau)(\texttt{Mul}+\texttt{Add})$ | 1 |
| Mahdavi [11] (Levelup) | PHE | / | $K\tau\cdot\texttt{Mul}^*$ | $O(K\tau)(\texttt{Add}+\texttt{Mul}^*)$ | 1 |
| Tai [14] (HHH) | AHE | / | $K\tau\cdot$ DGK | $O(K\tau)(\texttt{Add}+\texttt{Mul}^*)$ | 2 |
| Kiss [15] (GGH) | GC,AHE,OT | $KMt\cdot$ (GC + $\binom{2}{1}$-OT) | $K\tau\cdot$ GC | $O(K\tau)(\texttt{Add}+\texttt{Mul}^*)$ | 2 |
| Kiss [15] (HGH) | GC,SS,AHE | $KM\tau\cdot(\texttt{Add}+\texttt{Dec})$ | $K\tau\cdot$ GC | $O(K\tau)(\texttt{Add}+\texttt{Mul}^*)$ | 3 |
| **Ours (Kangaroo)** | **PHE,SS** | $2K\cdot\texttt{Mul}^*+O(K\log M)\texttt{Rot}$ | $\lceil\frac{K}{M}\rceil(2\cdot\texttt{Mul}^*+\texttt{Enc}+\texttt{Dec})$ | $2\lceil\frac{K}{M}\rceil(\texttt{Mul}^*+\texttt{Enc}+\texttt{Dec})$ | **4** |

[1] In the schemes [10], [11], we only analyze the operation costs of XCMP.

Then, the server sends the blinded result $[\![I']\!]$ to the client as line 1. The client constructs a tree by $\mathcal{T}_s^*$. After receiving $[\![I']\!]$, the client decrypts and assigns the corresponding left cost and right cost for the $n$-th decision node as line 2. The client sums costs along each path to obtain $I''$, encrypts and sends $[\![I'']\!]$ to the server as lines $3-4$. Similarly, the server also constructs a tree by $\mathcal{T}_s^*$ and sums the perturbation costs as lines $5-6$. After getting $R''$, the server recovers the cost accumulation result $[\![I]\!]$ for each path as line 7. Moreover, the server takes $-[\![I]\!]$ and $\mathbf{0}$ as the input of `PackObliviousCom` to transform path cost-based evaluation into polynomial-based evaluation as line 8. It is worth noting that the value on the correct path is 0, and the values of other paths are negative. Therefore, after executing `PackObliviousCom`, the value on the correct path is 1, and the values of other paths are 0. By multiplying the weight vector, only the correctly indexed weight is preserved. The correctness of `PackPathEva` is proven in Theorem 3.

## V. Kangaroo for Large-Scale Evaluation

In this section, we present the inference protocol of Kangaroo under the client-server model for large-scale evaluation, such as random forest. Then, we analyze its computational complexity and compare it with the existing schemes.

### A. The Inference Protocol for Random Forests

When dealing with large-scale models, such as random forests or a deep and large decision tree, the number of decision nodes increases significantly. Kangaroo are capable of handling these models, and we use random forests as an example. Due to space limitation, the inference protocol for random forests is shown in Fig. 9. First, the server performs `FeatureSelPack` to select and merge relevant features for multiple tree models. `FeatureSelPack` is optimized by `I-PackFeatureSel` and is mentioned in Algorithm 2. Then, the server and the client jointly execute `PackObliviousCom` to complete the comparison. To protect the model structure, the tree model is hiding by introducing dummy nodes and randomly swapping the internal nodes. Therefore, the computations need to be adjusted to eliminate the influence introduced by these modifications, as highlighted in blue. First, compute $R_\gamma * \Upsilon_\gamma^{pack}$ to determine whether the comparison result of each node needs to be flipped. Second, since the true weights are on the left child of the dummy node, the comparison result will be 0. Third, if the left and right child nodes of the dummy node have been swapped, the comparison result will be 1. Based on three rules, the comparison results can be recovered. Next, `PackPathEva` is executed to obtain the encrypted evaluation results $\{[\![T_\gamma]\!]\}_{\gamma=1}^{\Gamma}$. It is worth noting that each ciphertext vector packs the results of $M$ models. Therefore, for each `PackPathEva`, the client and the server jointly build $M$ tree structures using the structural indices in plaintext, perform path cost-based evaluations over plaintext, and then pack the evaluation results into a single ciphertext vector. Subsequently, `PackObliviousCom` is employed to convert all evaluation outcomes into polynomial-based evaluations, and $[\![T_\gamma]\!]$ is calculated. To respond the result, the server generates a random mask vector $T'$ and sends $\sum_{\gamma=1}^{\Gamma}[\![T_\gamma]\!]+\mathbf{T}'$, $\sum_{i=1}^{(\tau^*+1)M}T'[i]$ to the client. The client decrypts and obtains the inference result $\pi=\sum_{i=1}^{(\tau^*+1)M}T''[i]-\sum_{i=1}^{(\tau^*+1)M}T'[i]$.

### B. Complexity Analysis

We conduct a detailed analysis and comparison of the online operation costs for two-party inference schemes across two categories of inference: depth-round [19], [20] and constant-round schemes [10], [11], [14], [15]. The comparative results are summarized in TABLE II. It is evident that depth-round schemes incur a significant increase in communication rounds as the model depth $D$ grows, leading to considerable communication cost in real-world networks. In contrast, constant-round protocols are unaffected by communication latency. However, their performance is heavily influenced by the number of tree nodes $\tau$ and the number of trees $K$, making them unsuitable for evaluating large-scale random forests. Through careful design, our Kangaroo framework significantly reduces the aforementioned computation cost by leveraging efficient amortization. Moreover, Kangaroo can further achieve full amortization through the adaptive encoding adjustment strategy mentioned in Section VI-C.

## VI. Enhancement For practical Applications

To enhance practicality, we introduce several optimization strategies, including same-sharing-for-same-model, latency-aware strategy, and adaptive encoding adjustment.

### A. Same-Sharing-for-Same-Model

In Kangaroo, the server executes the model hiding and extraction to obtain $\{\mathbf{M}_k\}_{k=1}^{K}$, $\{\mathcal{T}^*_{k,s}\}_{k=1}^{K}$, and $\{\mathbf{Y}_\gamma^{pack}, \mathbf{W}_\gamma^{pack}, \Upsilon_\gamma^{pack}, \Psi_\gamma^{pack}\}_{\gamma=1}^{\Gamma}$ and publishes $\mathcal{T}^*_{k,s}$ to the client. We observe that, due to the obfuscated nature of our `PackPathEva`, the client cannot distinguish which leaf node is accessed from the paths. Therefore, publishing same model structure indices $\{\mathcal{T}^*_{k,s}\}_{k=1}^{K}$ can protect the evaluation results while maximizing the entropy of the tree structure [85], [86], thus preserving the privacy of the original model structures. Additionally, we note that once the packed model is determined, the polynomial length `N` and plaintext modulo $q$ are fixed, which also determines the encoder. This implies that the server does not need to perform additional encoding on the packed model when responding to any client request, thereby reducing computation cost. It also ensures that the parameter $\zeta$ is uniquely determined, where we set $\zeta = 2^{\frac{\log q}{2}-1}$ to ensure the correctness. Similarly, same-sharing-for-same-model can be also applicable to outsourced scheme, enhancing its practicality.

### B. Latency-Aware Strategy

In client-server model, the server needs to generate certain parameters to complete the model evaluation. To ensure the protection of sensitive information, these parameters must be randomly generated each time. However, we note that the generation of these parameters, such as $\{\mathbf{E}, \mathbf{A}, \mathbf{B}, \mathbf{R}, \mathbf{R}', \mathbf{R}''\}$, is both independent of the model and the client's data, and computationally inexpensive. Therefore, these operations can be amortized and executed during the waiting time for responses from the other party. We refer to this as a latency-aware strategy, which is also applied in other areas. For instance, Kangaroo utilizes encryption operations such as $[\![X]\!] = [\![0]\!] + \mathbf{X}$, where $[\![0]\!]$ can be precomputed during the waiting period. During the process, we also aim to minimize operations on the encoding and ciphertext as much as possible. For example, the server can compute $B \circ R - A \circ R \circ Y$ and encode the result to reduce the number of encodings. When using $(-[\![I]\!], \mathbf{0})$ as the input of `PackObliviousCom`, the server can compute $-A \circ R$ and encode it to minimize the negation operations on ciphertext. These optimizations further enhance the practicality. Similarly, the latency-aware strategy can be also applicable to outsourced scheme.

### C. Adaptive Encoding Adjustment

In large-scale evaluation, each tree is often pruned to prevent overfitting, resulting in varying structures across different trees [33], [35]–[37]. Padding all trees with dummy nodes to ensure that each tree contains exactly $\lceil \frac{\mathtt{N}}{M} \rceil$ nodes would lead to inefficient ciphertext packing and significantly reduce the actual packing utilization of Kangaroo. Fortunately, we can easily adjust the packing size for each tree to solve the problem. This aligns with our design principle of treating each ciphertext slot as a representation of a node. For clarity, we present in Fig. 3 a toy example without applying model hiding. The core idea is to encode features from multiple trees into a single polynomial. During feature selection, the algorithm simultaneously selects node features across multiple trees. The selected features are then further packed and compared. During the path evaluation, it is only necessary to identify which positions correspond to nodes in specific trees. It is evident that the strategy further reduces the number of feature selection and fully exploits every coefficient of the polynomial for inference. Thus, it can achieve the full amortization. The strategy can also be applied to single tree evaluation. It is worth noting that this process does not leak any structural information about the model, as the tree structures are randomly permuted before publishing, thereby meeting our security requirements. Moreover, it is observed that during feature selection, the number of rotations is significantly reduced when $M$ is a power of 2. Therefore, by adjusting $M$ to $M^*$ and padding with a few dummy features, the inference efficiency can be improved, where $M^*$ is usually a power of 2. This optimization is also demonstrated in our experiments.

## VII. Performance Evaluation

In evaluating Kangaroo within client-server model, we aim to answer the following two main research questions (RQs).

- **RQ1** : What are the characteristics and advantages of the individual components in Kangaroo compared to recent state-of-the-art (SOTA) methods?
- **RQ2** : How efficient is Kangaroo in handling large-scale tree models under a WAN setting compared to existing SOTA two-party PDTE schemes?

### A. Experimental Setting

- **Implementation:** The Kangaroo is implemented using Microsoft SEAL version 4.1 [87], which implements the BFV cryptosystem in polynomial and batching technique [79]. The data is encrypted with 128-bit security and the default degree of the polynomial `N` is set to 8192. The plaintext modulo $q$ is set to 50-bit. We provide a clear implementation on GitHub to facilitate verification of the correctness and performance of our schemes. We compare Kangaroo with the SOTA schemes [11], [14], [15], [19], [20]. The comparison is conducted using the following implementations [88]–[92]. To ensure the reproducibility of our experimental results, all experiments were conducted in a controlled environment with simulated network conditions, unless stated otherwise. We adopt three different bandwidth (bit/second) and round-trip time (RTT) over local-area network (LAN, 1 Gbps, RTT: 0.1 ms) metropolitan-area network (MAN, 100 Mbps, RTT: 6 ms), and wide-area network (WAN, 40 Mbps, RTT: 80 ms) settings to compare their performance. All experiments are conducted on two ThinkPad-P53 machines with single thread, 23.1 GB RAM, and an Intel Core i5-9400H 2.50GHz processor running

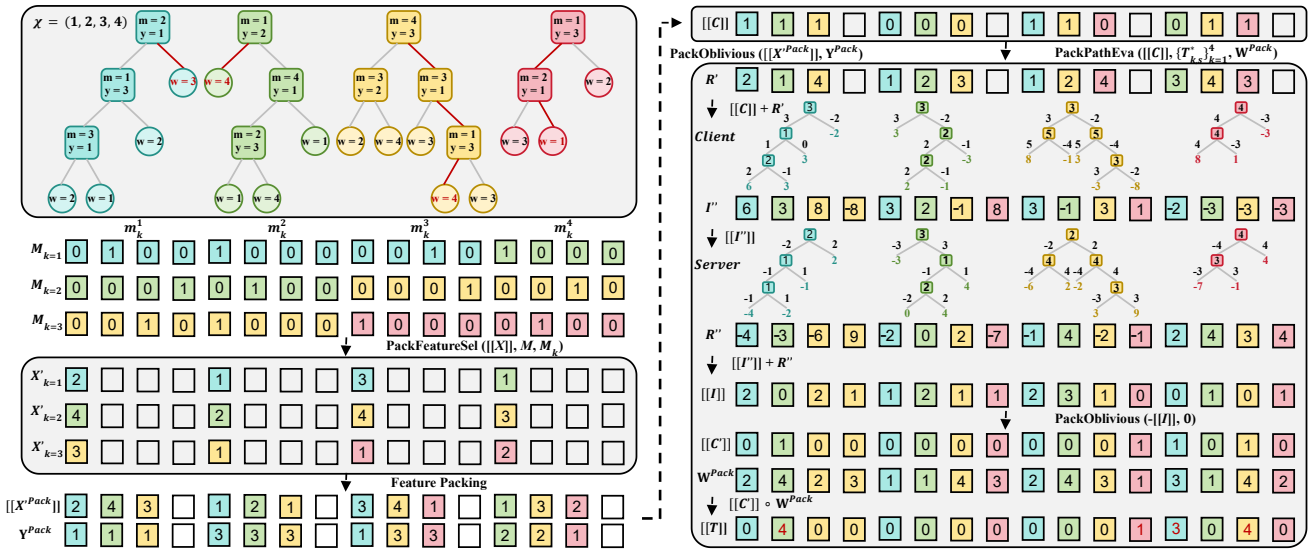
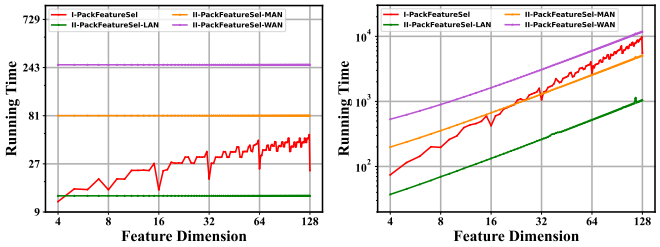



Fig. 3: Toy example without model hiding for adaptive encoding adjustment, where $\mathtt{N} = 16$, $M = 4$, and $K = 4$.




(a) One Feature Selection (b) Fully Feature Selection

Fig. 4: The running time (ms) under different network.

on Ubuntu 18.04.6. The experiments show average results, and each is repeated five times independently.

• **Datasets and Tree Models:** We utilize datasets from the UCI Machine Learning Repository to verify the correctness of our schemes and evaluate their performance. Due to space limitation, details of the datasets and tree models are provided in schemes [11], [15]. In addition, we also provide detailed model structures and plaintext evaluation interfaces in our GitHub. In Kangaroo, all tree models are initialized with dummy nodes and randomized. The model is encoded and packed as mentioned in Section III-B and VI-C.

### *B. Microbenchmarks*

To answer **RQ1**, we perform a comprehensive set of microbenchmarks to test the efficiency for each of the basic operators in Kangaroo, including secure feature selection, oblivious comparison, and secure path evaluation.

• **Secure Feature Selection:** Kangaroo supports two secure feature selection approaches, including `I-PackFeatureSel` (Algorithm 1) and `II-PackFeatureSel` (Fig. 10). The `I-PackFeatureSel` enables non-interactive feature selection but incurs a computation cost of $\mathtt{Mul}^* + O(\log M)\mathtt{Rot}$. In

TABLE III: The online running time and communication cost for 1-out-of-2 feature selection.

| | **Slient OT [71]** | | `I-PackFeatureSel` | |
|---|---|---|---|---|
| | 4192 | Amortized | 4192 | Amortized |
| Online Time | 15.00 (ms) | 3.66 (us) | 7.27 (ms) | 1.77 (us) |
| Online Comm. | 71.54 (KB) | 17.89 (B) | 0 | 0 |

contrast, `II-PackFeatureSel` scheme requires one interaction with communication cost of $2 \cdot \frac{2N\bar{Q}}{8192}$ (KB), but only involves $\mathtt{Mul}^* + \mathtt{Dec} + \mathtt{Enc}$ operations. We evaluate the performance of single and fully amortized feature selection, as shown in Fig. 4. The fully amortized feature selection is achieved by Algorithm 2. From the results, we observe that adjusting $M$ can reduce the number of rotations required. Overall, `I-PackFeatureSel` is suitable for scenarios with low feature dimensions and constrained bandwidth, whereas `II-PackFeatureSel` performs better in high-bandwidth environments with high feature dimensions. We also compare `I-PackFeatureSel` with Slient OT [71] for 1-out-of-2 to finish the feature selection. The result is shown in TABLE III. As shown in TABLE III, `I-PackFeatureSel` offers two primary advantages: it is non-interactive and particularly well-suited for small-scale data selection. Moreover, it requires any offline operations, making it lightweight and easy to deploy.

• **Oblivious Comparison:** As shown in TABLE IV, Kangaroo achieves the lowest amortized online time for oblivious comparison `PackObliviousCom`, outperforming DGK [14], ABY [66], and even the recent Cheetah [43]. In their scheme, the bit size $t$ is set to 32-bit. Different from ABY and Cheetah, which require significant offline communication or setup phases, `PackObliviousCom` does not rely on offline computation or communication. Furthermore, their communication rounds scale with $O(\log t)$, while `PackObliviousCom`

TABLE IV: The running time and communication (KB) for oblivious comparison.

| | DGK [14] | ABY [66] | Cheetah [43] | Ours |
|---|---|---|---|---|
| Offline Time | 0 | 39.31 (ms) | 1363.92 (ms) | 0 |
| Offline Comm. | 0 | 1501.48 | 1438.32 | 0 |
| Amortized Online Time | 142.98 (ms) | 7.66 (us) | 10.37 (us) | 0.88 (us) |
| Amortized Online Comm. | 8.25 | 1.50 | 0.040 | 0.10 |
| Online Round | 1 | $O(\log t)$ | $O(\log_4 t)$ | 1 |

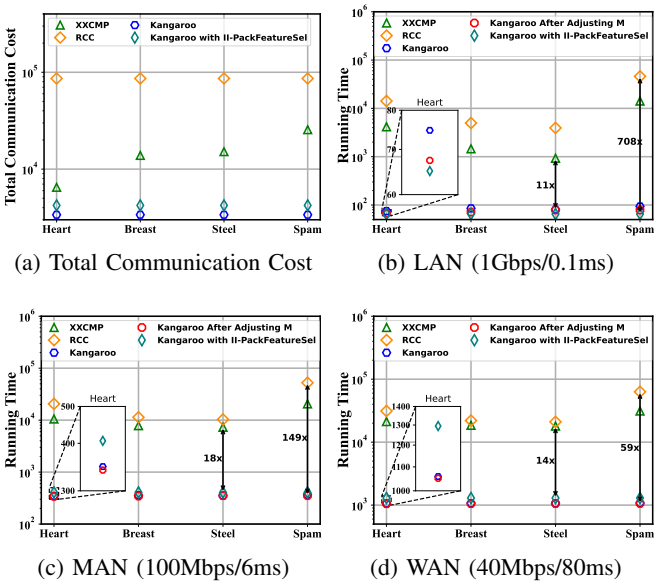
TABLE V: The amortized running time and amortized ciphertext size (bit) for different operations.

| | Enc | Add | Plain-Mul | Dec |
|---|---|---|---|---|
| Paillier (4096) | 5.80 (ms) | 3.84 (us) | 42.86 (us) | 5.80 (ms) |
| BFV (422.18) | 0.51 (us) **11370**× | 0.08 (us) **48**× | 0.31 (us) **138**× | 0.18 (us) **32220**× |

requires only a single round of communication. This makes `PackObliviousCom` particularly well-suited for real-world, large-scale inference tasks.

• **Secure Path Evaluation:** During the path evaluation, the path cost-based evaluation is more efficient than polynomial-based evaluation. This is because polynomial-based evaluation requires ciphertexts with a multiplicative depth of $D$, as in the scheme [44], which significantly impacts inference efficiency. In contrast, path cost-based evaluation, such as in [14], only involves ciphertext addition and plaintext multiplication. However, as shown in TABLE II, this approach still incurs considerable computation cost. While Boolean comparison results can be transformed into arithmetic values to enable path evaluation over non-packed ciphertexts, this method incurs substantial overhead due to the numerous encryption, decryption, addition, and multiplication operations required, all of which grow proportionally with the number of nodes. As illustrated in TABLE V, Paillier with 128-bit security level [93] performs poorly in large-scale computations compared to BFV. In Kangaroo, `PackPathEva` uses PHE to amortize the costs, resulting in a small number of multiplications, encryptions, and decryptions, as shown in TABLE II.

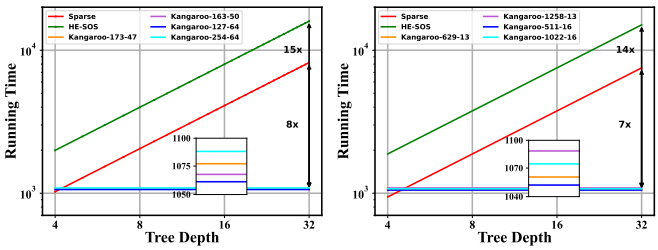
Built upon low-latency, efficient, and fully amortized components, Kangaroo is particularly well-suited for large-scale decision tree evaluation in WAN setting. Even when compared with lower-latency schemes [11], such as one-round communication protocols, RCC and XXCMP, Kangaroo still outperforms them significantly for small-scale decision tree evaluation, achieving up to $14\times$-$59\times$ performance improvements in WAN scenarios. The results are presented in Fig. 5. Clearly, in WAN setting with large-scale decision tree evaluation, Kangaroo will show more superior performance, as its amortization capabilities are fully leveraged under such scenarios. Additionally, we can also observe that adjusting $M$ can improve inference efficiency, such as $M$ from 13 to 16 on the Heart dataset in LAN setting.




(a) Total Communication Cost (b) LAN (1Gbps/0.1ms)

(c) MAN (100Mbps/6ms) (d) WAN (40Mbps/80ms)

Fig. 5: The comparison of total communication cost (KB) and running time (ms) with one round interactive schemes. The $(M, D, T)$ are as follows: Heart $(13, 3, 5)$, Breast $(30, 7, 17)$, Steel $(33, 5, 6)$, and Spam $(57, 16, 58)$.




(a) $M = 47$ (b) $M = 13$

Fig. 6: The running time (ms) for large-scale single tree evaluation across different tree depths in WAN setting. Kangaroo-$a$-$b$: where $a$ denotes the maximum number of decision nodes supported, and $b$ represents the adjusted feature dimension $M$.

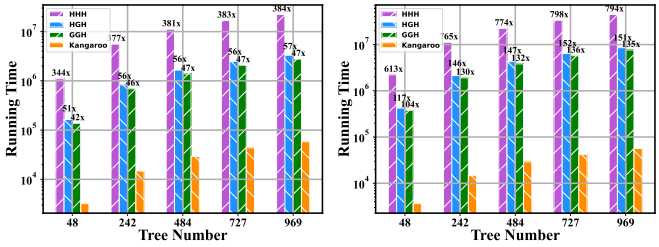
### C. Large-Scale Tree Models Benchmarks

To answer the **RQ2**, we evaluate the end-to-end inference latency under WAN setting with large-scale tree models, such as large-scale single tree evaluation and large-scale random forest evaluation, which measures from the time the feature vector is sent until the inference result is received. We also compare Kangaroo with existing two-party inference schemes [14], [15], [19], [20] to show the efficiency of Kangaroo.

• **Large-Scale Single Tree Evaluation:** As shown in TABLE VI, Kangaroo is highly efficient and well-suited for WAN-based large-scale single tree evaluation. Compared to prior SOTA schemes [14], [15], [19], [20], Kangaroo achieves

TABLE VI: Total communication cost (KB) and online runtime (ms) for large-scale single tree evaluation in WAN setting.

| Datasets $(M, D, \tau)$ | HHH [14] | | HGH [15] | | GGH [15] | | Sparse [19] | | HE-SOS [20] | | Kangaroo | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Comm. | Time | Comm. | Time | Comm. | Time | Comm. | Time | Comm. | Time | Comm. | Time |
| **Digits** $(47, 15, 168)$ | 1163 | 23190 | 1299 | 3578 | 4472 | 3100 | 131 | 3844 | 14079 | 7486 | **3379** | **1068** (**3**× ∼ **22**×) |
| **Diabetes** $(10, 28, 393)$ | 2227 | 43122 | 2973 | 8320 | 9868 | 7525 | 249 | 6614 | 26545 | 13270 | **3379** | **1050** (**6**× ∼ **41**×) |
| **Boston** $(13, 30, 425)$ | 2419 | 46416 | 3157 | 8997 | 10370 | 8138 | 264 | 7058 | 28513 | 14117 | **3379** | **1050** (**7**× ∼ **44**×) |




(a) Digits (b) Boston

Fig. 7: The running time (ms) for large-scale random forest evaluation across different tree numbers in WAN setting.

TABLE VII: Total online runtime for privacy-preserving applications under real-world WAN conditions.

| Applications | JD Cloud (5 Mbps), TP (1 Mbps), RTT (30 ms) | Ali Cloud (100 Mbps), TP (1 Mbps), RTT (40 ms) |
|---|---|---|
| Image Recognition | 6.96 (s) | 3.18 (s) |
| Medical Diagnostics | 5.92 (s) | 2.52 (s) |
| Financial Forecasting | 5.98 (s) | 3.06 (s) |

[1] We leverage the Digits $(47, 15, 168)$, Diabetes $(10, 28, 393)$, and Boston datasets $(13, 30, 425)$ to demonstrate Kangaroo's applicability in image recognition, medical diagnostics, and financial forecasting, respectively.

$3\times$-$22\times$ speedup on the Digits dataset, $6\times$-$41\times$ on the Diabetes dataset, and $7\times$-$44\times$ on the Boston dataset, where $M$ is adjusted to 50, 16, and 16. Furthermore, we construct tree models with varying depths and control the number of decision nodes through model pruning. As shown in Fig. 6, compared with schemes [19], [20], Kangaroo achieves an improvement of $8\times$-$15\times$ when $M = 47$, $D = 32$, and $7\times$-$14\times$ when $M = 13$, $D = 32$. Moreover, Kangaroo can flexibly control the number of decision nodes by adjusting $M$. Furthermore, different from prior schemes [15], [19], [20], Kangaroo operates without any offline computation or communication, making it particularly well-suited for handling large-scale multiple client-side inference requests.

• **Large-Scale Random Forest Evaluation:** We also construct large-scale random forest by Digits and Diabetes datasets with different tree numbers and compare Kangaroo with the schemes [14], [15]. The results are shown in Fig. 7. When the forest consists of 969 trees, containing 162792 nodes for the Digits dataset and 411825 nodes for the Boston dataset, Kangaroo achieves approximately $47\times$-$384\times$ speedup over existing schemes on Digits, and about $135\times$-$794\times$ improvement on Boston. Moreover, the average inference time per tree is about 60 ms. All these advantages stem from Kangaroo's full amortization capability, which enables it to consistently achieve the best performance among constant-round schemes.

### *D. Further Experiments and Discussion*

Kangaroo demonstrates strong performance in enabling large-scale, privacy-preserving decision tree evaluation over WAN environments. It can be seamlessly integrated into a wide range of decision tree-based applications where data privacy is a critical concern. In particular, it is well-suited for scenarios involving sensitive information, such as image recognition, medical diagnostics, and financial forecasting. To demonstrate Kangaroo's capability under real-world WAN conditions, we deploy servers across different cloud platforms. One server is hosted on JD Cloud, equipped with single thread, 3.8 GB RAM, and an Intel(R) Xeon(R) Gold 6148 @ 2.40 GHz processor, running Ubuntu 22.04.5. The other server is deployed on Alibaba Cloud, featuring a single CPU thread, 4 GB RAM, and an Intel(R) Xeon(R) Platinum @ 2.50 GHz processor, also running Ubuntu 22.04.5. The client is still deployed on the ThinkPad-P53 (TP) with single thread. As shown in TABLE VII, Kangaroo still demonstrates strong efficiency even under constrained bandwidth conditions. It is worth noting, however, that Kangaroo's performance on small-scale models in LAN settings may be slightly lower than that of schemes specifically optimized for such environments (e.g., [19]). To improve performance in these scenarios, several optimization strategies, such as reducing the ciphertext modulus and decreasing the polynomial degree, can be employed to further accelerate computation.

## VIII. Conclusion

In this work, we present Kangaroo, a private and amortized two-party inference framework over WAN for large-scale decision tree evaluation. By utilizing PHE to design a set of new secure feature selection, oblivious comparison, and secure path evaluation protocols, we fully exploit each coefficient in the PHE ciphertexts to efficiently pack deci-

sion tree nodes, thereby amortizing both computation and communication costs. Experimental results demonstrate that Kangaroo's core components deliver strong performance, and Kangaroo achieves significant speedups over SOTA two-party PDTE schemes in WAN setting with large-scale models.

In future work, we plan to build on the design principles of Kangaroo to explore more secure and efficient inference mechanisms, such as those for deep neural networks (DNNs), and develop corresponding optimization techniques to accelerate their performance. We also intend to investigate stronger threat models in outsourced settings, including malicious adversaries, with a focus on ensuring both data privacy and the correctness of inference results.

## ACKNOWLEDGMENT

We thank all anonymous reviewers and shepherds for their helpful feedback. This work was supported by National Cryptologic Science Fund of China (2025NCSF02015), National Natural Science Foundation of China (U22B2030, 62572020), Shaanxi Provincial Key Research and Development Program(2024SF2-GJHX-37), Shenzhen Science and Technology Program (JGJZD20240729142310014), Young Elite Scientists Sponsorship Program by CAST (2023QNRC001), and Fundamental Research Funds for the Central Universities (YJSJ25011). Hui Zhu is the corresponding author.

## REFERENCES

[1] W. Gan, S. Wan, and P. S. Yu, "Model-as-a-service (maas): A survey," in *IEEE Big Data*. IEEE, 2023, pp. 4636–4645.

[2] J. L. C. Bárcena, P. Ducange, F. Marcelloni, and A. Renda, "Increasing trust in AI through privacy preservation and model explainability: Federated learning of fuzzy regression trees," *Inf. Fusion*, vol. 113, p. 102598, 2025.

[3] N. Arndt, P. Molitor, and R. Usbeck, "Machine learning applications," *it Inf. Technol.*, vol. 65, no. 4-5, pp. 139–141, 2023.

[4] J. Hou, H. Liu, Y. Liu, Y. Wang, P. Wan, and X. Li, "Model protection: Real-time privacy-preserving inference service for model privacy at the edge," *IEEE Trans. Dependable Secur. Comput.*, vol. 19, no. 6, pp. 4270–4284, 2022.

[5] Q. Jia, L. Guo, Z. Jin, and Y. Fang, "Preserving model privacy for machine learning in distributed systems," *IEEE Trans. Parallel Distributed Syst.*, vol. 29, no. 8, pp. 1808–1822, 2018.

[6] U. Park, Y. Kang, H. Lee, and S. Yun, "A stacking heterogeneous ensemble learning method for the prediction of building construction project costs," *Applied sciences*, vol. 12, no. 19, p. 9729, 2022.

[7] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *KDD*. ACM, 2016, pp. 785–794.

[8] V. G. Costa and C. E. Pedreira, "Recent advances in decision trees: an updated survey," *Artif. Intell. Rev.*, vol. 56, no. 5, pp. 4765–4800, 2023.

[9] W. Lu, J. Zhou, and J. Sakuma, "Non-interactive and output expressive private comparison from homomorphic encryption," in *AsiaCCS*. ACM, 2018, pp. 67–74.

[10] K. Cong, D. Das, J. Park, and H. V. L. Pereira, "Sortinghat: Efficient private decision tree evaluation via homomorphic encryption and transciphering," in *CCS*. ACM, 2022, pp. 563–577.

[11] R. A. Mahdavi, H. Ni, D. Linkov, and F. Kerschbaum, "Level up: Private non-interactive decision tree evaluation using levelled homomorphic encryption," in *CCS*. ACM, 2023, pp. 2945–2958.

[12] A. Akavia, M. Leibovich, Y. S. Resheff, R. Ron, M. Shahar, and M. Vald, "Privacy-preserving decision trees training and prediction," *ACM Trans. Priv. Secur.*, vol. 25, no. 3, pp. 24:1–24:30, 2022.

[13] D. J. Wu, T. Feng, M. Naehrig, and K. E. Lauter, "Privately evaluating decision trees and random forests," *Proc. Priv. Enhancing Technol.*, vol. 2016, no. 4, pp. 335–355, 2016.

[14] R. K. H. Tai, J. P. K. Ma, Y. Zhao, and S. S. M. Chow, "Privacy-preserving decision trees evaluation via linear functions," in *ESORICS (2)*, ser. Lecture Notes in Computer Science, vol. 10493. Springer, 2017, pp. 494–512.

[15] Á. Kiss, M. Naderpour, J. Liu, N. Asokan, and T. Schneider, "Sok: Modular and efficient private decision tree evaluation," *Proc. Priv. Enhancing Technol.*, vol. 2019, no. 2, pp. 187–208, 2019.

[16] R. Bost, R. A. Popa, S. Tu, and S. Goldwasser, "Machine learning classification over encrypted data," in *NDSS*. The Internet Society, 2015.

[17] M. Joye and F. Salehi, "Private yet efficient decision tree evaluation," in *DBSec*, ser. Lecture Notes in Computer Science, vol. 10980. Springer, 2018, pp. 243–259.

[18] A. Tueno, F. Kerschbaum, and S. Katzenbeisser, "Private evaluation of decision trees using sublinear cost," *Proc. Priv. Enhancing Technol.*, vol. 2019, no. 1, pp. 266–286, 2019.

[19] J. P. K. Ma, R. K. H. Tai, Y. Zhao, and S. S. M. Chow, "Let's stride blindfolded in a forest: Sublinear multi-client decision trees evaluation," in *NDSS*. The Internet Society, 2021.

[20] J. Bai, X. Song, S. Cui, E. Chang, and G. Russello, "Scalable private decision tree evaluation with sublinear communication," in *AsiaCCS*. ACM, 2022, pp. 843–857.

[21] A. Tueno, Y. Boev, and F. Kerschbaum, "Non-interactive private decision tree evaluation," in *DBSec*, ser. Lecture Notes in Computer Science, vol. 12122. Springer, 2020, pp. 174–194.

[22] J. Bai, X. Song, X. Zhang, Q. Wang, S. Cui, E. Chang, and G. Russello, "Mostree: Malicious secure private decision tree evaluation with sublinear communication," in *ACSAC*. ACM, 2023, pp. 799–813.

[23] S. Yuan, H. Li, X. Qian, M. Hao, Y. Zhai, and G. Xu, "Efficient and privacy-preserving outsourcing of gradient boosting decision tree inference," *IEEE Trans. Serv. Comput.*, vol. 17, no. 5, pp. 2334–2348, 2024.

[24] J. Fu, K. Cheng, Y. Xia, A. Song, Q. Li, and Y. Shen, "Private decision tree evaluation with malicious security via function secret sharing," in *ESORICS (2)*, ser. Lecture Notes in Computer Science, vol. 14983. Springer, 2024, pp. 310–330.

[25] Y. Zheng, H. Duan, C. Wang, R. Wang, and S. Nepal, "Securely and efficiently outsourcing decision tree inference," *IEEE Trans. Dependable Secur. Comput.*, vol. 19, no. 3, pp. 1841–1855, 2022.

[26] Y. Zheng, C. Wang, R. Wang, H. Duan, and S. Nepal, "Optimizing secure decision tree inference outsourcing," *IEEE Trans. Dependable Secur. Comput.*, vol. 20, no. 4, pp. 3079–3092, 2023.

[27] L. Liu, R. Chen, X. Liu, J. Su, and L. Qiao, "Towards practical privacy-preserving decision tree training and evaluation in the cloud," *IEEE Trans. Inf. Forensics Secur.*, vol. 15, pp. 2914–2929, 2020.

[28] Z. Zhang, H. Zhang, X. Song, J. Lin, and F. Kong, "Secure outsourcing evaluation for sparse decision trees," *IEEE Transactions on Dependable and Secure Computing*, 2024.

[29] K. Ji, B. Zhang, T. Lu, L. Li, and K. Ren, "UC secure private branching program and decision tree evaluation," *IEEE Trans. Dependable Secur. Comput.*, vol. 20, no. 4, pp. 2836–2848, 2023.

[30] L. Liu, J. Su, R. Chen, J. Chen, G. Sun, and J. Li, "Secure and fast decision tree evaluation on outsourced cloud data," in *ML4CS*, ser. Lecture Notes in Computer Science, vol. 11806. Springer, 2019, pp. 361–377.

[31] N. Cheng, N. Gupta, A. Mitrokotsa, H. Morita, and K. Tozawa, "Constant-round private decision tree evaluation for secret shared data," *Proc. Priv. Enhancing Technol.*, vol. 2024, no. 1, pp. 397–412, 2024.

[32] J. M. Klusowski and P. M. Tian, "Large scale prediction with decision trees," *Journal of the American Statistical Association*, vol. 119, no. 545, pp. 525–537, 2024.

[33] J. Catlett, "Overprvning large decision trees." in *IJCAI*, 1991, pp. 764–769.

[34] L. Breiman, "Random forests," *Machine learning*, vol. 45, pp. 5–32, 2001.

[35] S. Zhou and L. Mentch, "Trees, forests, chickens, and eggs: when and why to prune trees in a random forest," *Statistical Analysis and Data Mining: The ASA Data Science Journal*, vol. 16, no. 1, pp. 45–64, 2023.

[36] S. Ren, X. Cao, Y. Wei, and J. Sun, "Global refinement of random forest," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 723–730.

[37] R. Duroux and E. Scornet, "Impact of subsampling and tree depth on random forests," *ESAIM: Probability and Statistics*, vol. 22, pp. 96–128, 2018.

[38] A. J. Izenman, *Modern multivariate statistical techniques*. Springer, 2008, vol. 1.

[39] G. James, D. Witten, T. Hastie, R. Tibshirani *et al.*, *An introduction to statistical learning*. Springer, 2013, vol. 112, no. 1.

[40] A. Diaa, L. Fenaux, T. Humphries, M. Dietz, F. Ebrahimianghazani, B. Kacsmar, X. Li, N. Lukas, R. A. Mahdavi, S. Oya, E. Amjadian, and F. Kerschbaum, "Fast and private inference of deep neural networks by co-designing activation functions," in *USENIX Security Symposium*. USENIX Association, 2024.

[41] Z. Zhang, R. Rathi, S. Perez, J. Bukhari, and Y. Zhong, "ZCNET: achieving high capacity in low power wide area networks," *IEEE/ACM Trans. Netw.*, vol. 30, no. 5, pp. 2032–2045, 2022.

[42] N. S. Chilamkurthy, O. J. Pandey, A. Ghosh, L. R. Cenkeramaddi, and H. Dai, "Low-power wide-area networks: A broad overview of its different aspects," *IEEE Access*, vol. 10, pp. 81 926–81 959, 2022.

[43] W. Lu, Z. Huang, Q. Zhang, Y. Wang, and C. Hong, "Squirrel: A scalable secure two-party computation framework for training gradient boosting decision tree," in *USENIX Security Symposium*. USENIX Association, 2023, pp. 6435–6451.

[44] J. Zhao, H. Zhu, W. Xu, F. Wang, R. Lu, and H. Li, "Sgboost: An efficient and privacy-preserving vertical federated tree boosting framework," *IEEE Trans. Inf. Forensics Secur.*, vol. 18, pp. 1022–1036, 2023.

[45] J. Zhao, H. Zhu, F. Wang, R. Lu, and H. Li, "Efficient and privacy-preserving tree-based inference via additive homomorphic encryption," *Inf. Sci.*, vol. 650, p. 119480, 2023.

[46] H. Shin, J. Choi, D. Lee, K. Kim, and Y. Lee, "Fully homomorphic training and inference on binary decision tree and random forest," in *ESORICS (3)*, ser. Lecture Notes in Computer Science, vol. 14984. Springer, 2024, pp. 217–237.

[47] A. Patra, T. Schneider, A. Suresh, and H. Yalame, "ABY2.0: improved mixed-protocol secure two-party computation," in *USENIX Security Symposium*. USENIX Association, 2021, pp. 2165–2182.

[48] Y. Zheng, H. Zhu, R. Lu, Y. Guan, S. Zhang, F. Wang, J. Shao, and H. Li, "Pgsim: Efficient and privacy-preserving graph similarity query over encrypted data in cloud," *IEEE Trans. Inf. Forensics Secur.*, vol. 18, pp. 2030–2045, 2023.

[49] M. C. Pike, "Remark on algorithm 235 [G6]: random permutation," *Commun. ACM*, vol. 8, no. 7, p. 445, 1965.

[50] P. Mohassel and P. Rindal, "Aby$^3$: A mixed protocol framework for machine learning," in *CCS*. ACM, 2018, pp. 35–52.

[51] E. Boyle, N. Gilboa, Y. Ishai, and A. Nof, "Practical fully secure three-party computation via sublinear distributed zero-knowledge proofs," in *CCS*. ACM, 2019, pp. 869–886.

[52] J. Furukawa, Y. Lindell, A. Nof, and O. Weinstein, "High-throughput secure three-party computation for malicious adversaries and an honest majority," in *EUROCRYPT (2)*, ser. Lecture Notes in Computer Science, vol. 10211, 2017, pp. 225–255.

[53] S. Wagh, D. Gupta, and N. Chandran, "Securenn: 3-party secure computation for neural network training," *Proc. Priv. Enhancing Technol.*, vol. 2019, no. 3, pp. 26–49, 2019.

[54] N. Gilboa and Y. Ishai, "Distributed point functions and their applications," in *EUROCRYPT*, ser. Lecture Notes in Computer Science, vol. 8441. Springer, 2014, pp. 640–658.

[55] E. Boyle, N. Gilboa, and Y. Ishai, "Function secret sharing," in *EUROCRYPT (2)*, ser. Lecture Notes in Computer Science, vol. 9057. Springer, 2015, pp. 337–367.

[56] ——, "Function secret sharing: Improvements and extensions," in *CCS*. ACM, 2016, pp. 1292–1303.

[57] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *AISTATS*, ser. Proceedings of Machine Learning Research, vol. 54. PMLR, 2017, pp. 1273–1282.

[58] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *ACM Trans. Intell. Syst. Technol.*, vol. 10, no. 2, pp. 12:1–12:19, 2019.

[59] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," *CoRR*, vol. abs/1610.05492, 2016.

[60] Y. Wu, S. Cai, X. Xiao, G. Chen, and B. C. Ooi, "Privacy preserving vertical federated learning for tree-based models," *Proceedings of the VLDB Endowment.*, vol. 13, no. 11, pp. 2090–2103, 2020.

[61] W. Fang, D. Zhao, J. Tan, C. Chen, C. Yu, L. Wang, L. Wang, J. Zhou, and B. Zhang, "Large-scale secure XGB for vertical federated learning," in *CIKM*. ACM, 2021, pp. 443–452.

[62] Y. Zheng, S. Xu, S. Wang, Y. Gao, and Z. Hua, "Privet: A privacy-preserving vertical federated learning service for gradient boosted decision tables," *IEEE Trans. Serv. Comput.*, vol. 16, no. 5, pp. 3604–3620, 2023.

[63] H. Mahdikhani, R. Lu, Y. Zheng, J. Shao, and A. A. Ghorbani, "Achieving o(log$^3$n) communication-efficient privacy-preserving range query in fog-based iot," *IEEE Internet Things J.*, vol. 7, no. 6, pp. 5220–5232, 2020.

[64] Z. Huang, W. Lu, C. Hong, and J. Ding, "Cheetah: Lean and fast secure two-party deep neural network inference," in *USENIX Security Symposium*. USENIX Association, 2022, pp. 809–826.

[65] T. Veugen, "Improving the DGK comparison protocol," in *WIFS*. IEEE, 2012, pp. 49–54.

[66] D. Demmler, T. Schneider, and M. Zohner, "Aby-a framework for efficient mixed-protocol secure two-party computation." in *NDSS*, 2015.

[67] Y. Zheng, S. Xu, S. Wang, Y. Gao, and Z. Hua, "Privet: A privacy-preserving vertical federated learning service for gradient boosted decision tables," *IEEE Trans. Serv. Comput.*, vol. 16, no. 5, pp. 3604–3620, 2023.

[68] D. Rathee, M. Rathee, N. Kumar, N. Chandran, D. Gupta, A. Rastogi, and R. Sharma, "Cryptflow2: Practical 2-party secure inference," in *CCS*. ACM, 2020, pp. 325–342.

[69] B. Pinkas, T. Schneider, and M. Zohner, "Faster private set intersection based on OT extension," in *USENIX Security Symposium*. USENIX Association, 2014, pp. 797–812.

[70] G. Couteau, L. Devadas, S. Devadas, A. Koch, and S. Servan-Schreiber, "Quietot: Lightweight oblivious transfer with a public-key setup," in *ASIACRYPT (2)*, ser. Lecture Notes in Computer Science, vol. 15485. Springer, 2024, pp. 197–231.

[71] S. Raghuraman, P. Rindal, and T. Tanguy, "Expand-convolute codes for pseudorandom correlation generators from LPN," in *CRYPTO (4)*, ser. Lecture Notes in Computer Science, vol. 14084. Springer, 2023, pp. 602–632.

[72] C. Juvekar, V. Vaikuntanathan, and A. P. Chandrakasan, "GAZELLE: A low latency framework for secure neural network inference," in *USENIX Security Symposium*. USENIX Association, 2018, pp. 1651–1669.

[73] Q. Zhang, C. Xin, and H. Wu, "GALA: greedy computation for linear algebra in privacy-preserved neural networks," in *NDSS*. The Internet Society, 2021.

[74] W. Lu, Z. Huang, Z. Gu, J. Li, J. Liu, C. Hong, K. Ren, T. Wei, and W. Chen, "Bumblebee: Secure two-party inference framework for large transformers," in *NDSS*. The Internet Society, 2025.

[75] B. De Ville, "Decision trees," *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 5, no. 6, pp. 448–455, 2013.

[76] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," *IACR Cryptol. ePrint Arch.*, p. 144, 2012.

[77] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(leveled) fully homomorphic encryption without bootstrapping," *ACM Trans. Comput. Theory*, vol. 6, no. 3, pp. 13:1–13:36, 2014.

[78] J. H. Cheon, A. Kim, M. Kim, and Y. S. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *ASIACRYPT (1)*, ser. Lecture Notes in Computer Science, vol. 10624. Springer, 2017, pp. 409–437.

[79] N. P. Smart and F. Vercauteren, "Fully homomorphic SIMD operations," *Des. Codes Cryptogr.*, vol. 71, no. 1, pp. 57–81, 2014.

[80] J. Liu, M. Juuti, Y. Lu, and N. Asokan, "Oblivious neural network predictions via minionn transformations," in *CCS*. ACM, 2017, pp. 619–631.

[81] O. Goldreich, *Foundations of Cryptography, Volume 2*. Cambridge university press Cambridge, 2004.

[82] T. Araki, J. Furukawa, Y. Lindell, A. Nof, and K. Ohara, "High-throughput semi-honest secure three-party computation with an honest majority," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 805–817.

[83] W. Xu, H. Zhu, Y. Zheng, S. Bian, N. Sun, H. Yuan, D. Feng, and H. Li, "Kangaroo: A private and amortized inference framework over wan for large-scale decision tree evaluation," *arXiv preprint*, 2025, https://arxiv.org/abs/2509.03123.

[84] T. Veugen, "Improving the DGK comparison protocol," in *WIFS*. IEEE, 2012, pp. 49–54.

[85] L. Duan, X. Chen, W. Liu, D. Liu, K. Yue, and A. Li, "Structural entropy based graph structure learning for node classification," in *AAAI*. AAAI Press, 2024, pp. 8372–8379.
[86] A. Farzaneh, M. Badiu, and J. P. Coon, "On random tree structures, their entropy, and compression," *CoRR*, vol. abs/2309.09779, 2023.
[87] https://github.com/microsoft/SEAL.
[88] https://github.com/secretflow.
[89] https://github.com/osu-crypto/libOTe.
[90] https://github.com/encryptogroup/ABY.
[91] https://github.com/encryptogroup/PDTE.
[92] https://github.com/RasoulAM/private-decision-tree-evaluation.
[93] H. Ma, S. Han, and H. Lei, "Optimized paillier's cryptosystem with fast encryption and decryption," in *ACSAC*. ACM, 2021, pp. 106–118.

TABLE VIII: Notations for Kangaroo

| Notations | Definition |
|---|---|
| $M, D, t$ | Feature dimension, tree depth, bit size of feature |
| $K, k$ | Total number of tree, $1 \leq k \leq K$ |
| $\tau, \tau^*$ | Total decision node number before and after hiding |
| $\mathcal{T}_{k,s}, \mathcal{T}^*_{k,s}$ | Model structure index vector before and after hiding for $k$-th tree |
| $\mathbf{m}_k, \mathbf{y}_k, \mathbf{w}_k, \upsilon_k, \psi_k$ | Feature index, threshold, weight, flip condition, and node status vectors after hiding for $k$-th tree |
| $\Gamma, \gamma$ | Total number after packing, $1 \leq \gamma \leq \Gamma$ |
| $\mathbf{M}_k$ | Encoded feature index vector for $k$-th tree |
| $\mathbf{Y}^{pack}_{\gamma}$ | Encoded and packed feature threshold vector |
| $\mathbf{W}^{pack}_{\gamma}$ | Encoded and packed weight vector |
| $\Upsilon^{pack}_{\gamma}$ | Encoded and packed flip condition vector |
| $\Psi^{pack}_{\gamma}$ | Encoded and packed node status vector |
| $\mathbf{m}^n_k$ | $M$-size one-hot vector for $n$-th node of $k$-th tree |
| $\mathcal{X}$ | Client's feature vector $\{x_1, x_2, \cdots, x_M\}$ |
| $\mathtt{N}, q, Q$ | Polynomial size, plaintext and ciphertext modulo |
| $\zeta$ | Precision parameter |

## APPENDIX A
## KANGAROO EXTENSIONS

We propose a feature packing algorithm `FeatureSelPack` in Algorithm 2 to select and merge the feature for multiple trees. Moreover, we propose `II-PackFeatureSel` protocol Fig. 10 for the LAN setting to perform feature selection, which avoids the rotation overhead introduced in Algorithm 1. First, the server selects a random vector $E$ and calculates $[\![X']\!] = [\![X]\!] \circ \mathbf{M} + \mathbf{E}$. Then, the server sends $[\![X']\!]$ to the client. After receiving $[\![X']\!]$, the client decrypts and then divides it into $\tau^*$ blocks, each containing $M$ values. Next, the client sums the elements within each block to obtain the vector $X''$, encrypts it, and sends it to the server, where each summation result is placed in the corresponding position. Similarly, the server utilizes $E$ to perform summation and then negates each result to obtain the vector $E'$. At last, the server calculates $[\![X'']\!] + \mathbf{E}'$ to get the selected encrypted vector $[\![X']\!]$. The protocol can also be easily extended to support multiple trees to achieve the fully amortized feature selection. We also provide an outsourcing scheme for Kangaroo, and the interested readers can refer to the full version [83] and our github repository for more details.

`PackPathEva` **Protocol**
**SInput:** The encrypted vector $[\![C]\!]$, the obfuscated structure index $\mathcal{T}^*_s$, and the encoded weight $\mathbf{W}$.
**CInput:** The obfuscated structure index $\mathcal{T}^*_s$.
**SOutput:** The encrypted evaluation result vector $[\![T]\!]$.
▷ *The server executes:*
1: $R' \leftarrow \{r'_1, *, \cdots, *, \cdots, r'_{(\tau^*-1)M+1}, *, \cdots, *\}$, $[\![I']\!] \leftarrow [\![C]\!] + \mathbf{R}' \Rightarrow$ the client.
▷ *The client constructs a tree by $\mathcal{T}^*_s$ and executes:*
2: $I' \leftarrow \mathtt{Dec}([\![I']\!], \mathtt{s})$, the $I'[(n-1)M+1] \leftarrow n$-th `node->left.cost` and the $1 - I'[(n-1)M+1] \leftarrow n$-th `node->right.cost` for $1 \leq n \leq \tau^*$.
3: $I'' \leftarrow \{i''_1, 0, \cdots, 0, \cdots, i''_{\tau^*M+1}, 0, \cdots, 0\}$, where $I''[(n-1)M+1]$ is the sum of cost along the $n$-th path and $1 \leq n \leq \tau^* + 1$.
4: $[\![I'']\!] \leftarrow \mathtt{Enc}(I'', \mathtt{pk}) \Rightarrow$ the server.
▷ *The server constructs a tree by $\mathcal{T}^*_s$ and executes:*
5: The $-R'[(n-1)M+1] \leftarrow n$-th `node->left.cost` and the $R'[(n-1)M+1] \leftarrow n$-th `node->right.cost` for $1 \leq n \leq \tau^*$.
6: $R'' \leftarrow \{r''_1, 0, \cdots, 0, \cdots, r''_{\tau^*M+1}, 0, \cdots, 0\}$, where $R''[(n-1)M+1]$ is the sum of cost along the $n$-th path and $1 \leq n \leq \tau^* + 1$.
7: The server gets the encrypted results of path cost-based evaluation $[\![I]\!] \leftarrow [\![I'']\!] + \mathbf{R}''$.
▷ *The server and client jointly execute:*
8: The server gets $[\![C']\!] \leftarrow \mathtt{PackObliviousCom}(-[\![I]\!], \mathbf{0})$. ▷ *Transform path cost-based into polynomial-based.*
▷ *The server executes:*
9: The server gets $[\![T]\!] \leftarrow [\![C']\!] \circ \mathbf{W}$.

Fig. 8: Packed path evaluation protocol for single tree.

## APPENDIX B
## CORRECTNESS ANALYSIS

The correctness of the Kangaroo framework depends on three core building blocks. Hence, it suffices to prove the correctness of these components.

### A. The Correctness of `PackFeatureSel`

*Theorem 1:* After executing `PackFeatureSel`, the selected encrypted result $[\![X']\!] = [[x_{m[1]}], [*], \cdots, [*], \cdots, [x_{m[(\tau^*-1)M+1]}], [*], \cdots, [*]]$ can be obtained.

*Proof.* To make it easier to understand, we focus only on the first $M$ values in the packed ciphertext. Since the operation is parallel, it is correct for the other positions. First, in `PackFeatureSel`, $[x_{m[1]}]$ is selected by a one-hot vector $\mathbf{m}^1_k$, and other elements are set to 0. Therefore, the feature selection can be treated as the summation operation. In `I-PackFeatureSel`, we utilize rotate-and-sum technique to finish the summation of $M$ values. To improve the efficiency,

**The Inference Protocol for Random Forests**

**SInput:** $\{M, \mathbf{Y}_\gamma^{pack}, \mathbf{W}_\gamma^{pack}, \Upsilon_\gamma^{pack}, \Psi_\gamma^{pack}\}$, $\{\mathbf{M}_k\}_{k=1}^K$, and $\{\mathcal{T}_{k,s}^*\}_{k=1}^K$.

**CInput:** $[\![X]\!]$ and $\{\mathcal{T}_{k,s}^*\}_{k=1}^K$.

**COutput:** The aggregated evaluation result $\pi$.

**1) Secure Feature Selection**

▷ *The server executes:*

1: $\{[\![X_\gamma^{'pack}]\!]\}_\gamma^{pack} \leftarrow$ `FeatureSelPack`$([\![X]\!], M, \{\mathbf{M}_k\}_{k=1}^K)$.

**2) Oblivious Comparison**

▷ *The server and client jointly execute:*

2: $[\![V'_\gamma]\!] \leftarrow$ the steps $1-5$ of `PackObliviousCom` $([\![X_\gamma^{'pack}]\!], \mathbf{Y}_\gamma^{pack})$ for $1 \le \gamma \le \Gamma$.

▷ *The server executes:*

3: $R_\gamma \leftarrow R_\gamma * \Upsilon_\gamma^{pack}$, $R_\gamma[i] \leftarrow 0$ if $\Psi_\gamma^{pack}[i] = 0$ for $1 \le \gamma \le \Gamma$ and $1 \le i \le \tau^* M$.

4: In $C'_\gamma$, $C'_\gamma[i] \leftarrow 1$ if $R_\gamma[i] = -1$ or ($\Upsilon_\gamma^{pack}[i] = -1$ and $\Psi_\gamma^{pack}[i] = 0$), 0 otherwise for $1 \le i \le \tau^* M$.

5: The server gets $[\![C_\gamma]\!] \leftarrow \mathbf{C}'_\gamma + \mathbf{R}_\gamma \circ [\![V'_\gamma]\!]$.

**3) Secure Path Evaluation**

▷ *The server and client jointly execute:*

6: The server gets $[\![T_\gamma]\!] \leftarrow$ `PackPathEva`$([\![C_\gamma]\!], \{\mathcal{T}_{k,s}^*\}_{k=(\gamma-1)M+1}^{\gamma M}, \mathbf{W}_\gamma^{pack})$ for $1 \le \gamma \le \Gamma$.

**4) Result Response**

▷ *The server executes:*

7: $T' \leftarrow \mathcal{Z}_q^{(\tau^*+1)M}$, $\sum_{\gamma=1}^{\Gamma}[\![T_\gamma]\!] + \mathbf{T}'$, $\sum_{i=1}^{(\tau^*+1)M} T'[i] \Rightarrow$ the client.

▷ *The client executes:*

8: $T'' \leftarrow$ `Dec`$(\sum_{\gamma=1}^{\Gamma}[\![T_\gamma]\!] + \mathbf{T}', \mathtt{s})$.

9: The client gets $\pi \leftarrow \sum_{i=1}^{(\tau^*+1)M}(T''[i] - T'[i])$.

Fig. 9: The inference protocol for random forests under client-server model.

we utilize the division-based approach to finish the summation and the core idea is to sum two adjacent results and store the sum in the previous result. Finally, by iterating, the sum of $M$ values is accumulated into the first position. If a value does not have an adjacent result, it waits to enter the next round. We also show a toy example for $M = 8$ and $M = 12$ in Fig. 11 to better understand our `I-PackFeatureSel`. In `II-PackFeatureSel`, the server calculates $-\sum_{m=1}^{M}(E[m])$, and the client calculates $\sum_{m=1}^{M}(X'[m] + E[m])$. When the server calculates $[\![X'']\!] + \mathbf{E}'$, the sum of the $M$ value $[\![\sum_{m=1}^{M}(X'[m])]\!] = [\![\sum_{m=1}^{M}(X'[m]+E[m])]\!] - \sum_{m=1}^{M}(\mathbf{E}[m])$ is placed in the first position. Thus, Theorem 1 is correct. □

### B. The Correctness of `PackObliviousCom`

*Theorem 2:* After executing `PackObliviousCom`, the comparison result $[\![C]\!]$ of $[\![X']\!]$ and $\mathbf{Y}$ can be obtained, where $C[i] = 0$ if $X'[i] - Y[i] < 0$, $C[i] = 1$ otherwise.

**Algorithm 2** `FeatureSelPack`

**SInput:** $[\![X]\!]$, $M$, and $\{\mathbf{M}_k\}_{k=1}^K$.

**SOutput:** $\{[\![X_\gamma^{'pack}]\!]\}_{\gamma=1}^{\Gamma}$.

1: ▷ *The server executes:* ◁

2: $[\![X'_k]\!] \leftarrow$ `I-PackFeatureSel`$([\![X]\!], M, \mathbf{M}_k)$ for $1 \le k \le K$.

3: $E \leftarrow \{1, 0, \cdots, 0, \cdots, 1, 0, \cdots, 0\}$.

4: The server gets $[\![X_\gamma^{'pack}]\!] \leftarrow \sum_{m=1}^{M}$ `Rot`$([\![X'_{(\gamma-1)M+m}]\!] \circ \mathbf{E}, m-1)$ for $1 \le \gamma \le \Gamma$.

`II-PackFeatureSel` **Protocol**

**SInput:** $[\![X]\!]$, $M$, and $\mathbf{M}$.

**SOutput:** The selected encrypted vector $[\![X']\!]$.

▷ *The server executes:*

1: $E \leftarrow \mathcal{Z}_q^{\mathtt{N}}$, $[\![X']\!] \leftarrow [\![X]\!] \circ \mathbf{M} + \mathbf{E} \Rightarrow$ the client.

▷ *The client executes:*

2: $X' \leftarrow$ `Dec`$([\![X']\!], \mathtt{s})$, $X'' \leftarrow \{\sum_{i=1}^{M} X'[i], 0, \cdots, 0, \cdots, \sum_{i=1+M(\tau^*-1)}^{M\tau^*} X'[i], 0, \cdots, 0\}$.

3: $[\![X'']\!] \leftarrow$ `Enc`$(X'', \mathtt{pk}) \Rightarrow$ the server.

▷ *The server executes:*

4: $E' \leftarrow \{-\sum_{i=1}^{M} E[i], 0, \cdots, 0, \cdots, -\sum_{i=1+M(\tau^*-1)}^{M\tau^*} E[i], 0, \cdots, 0\}$.

5: The server gets $[\![X']\!] \leftarrow [\![X'']\!] + \mathbf{E}'$.

Fig. 10: Packed feature selection protocol for single tree.

*Proof.* To make it easier to understand, we focus only on the one value in the packed ciphertext. It is important to note that both $[\![X']\!]$ and $\mathbf{Y}$ in `PackObliviousCom` are quantized, ensuring that $X'[i]$ and $Y[i]$ lie within the range $[0, \zeta]$, where $\zeta = 2^{\frac{\log q}{2}-1}$. It follows that $X'[i] - Y[i] \in [-\zeta, \zeta]$. For $A$, $B$, and $R$, we have $\zeta > A[i] > B[i] > 0$ and $R[i] \in \{-1, 1\}$. Thus, $A[i] \cdot (X'[i] - Y[i]) + B[i] \in (-\zeta^2 + \zeta, 0) \cup (0, \zeta^2 + \zeta) = (-2^{\log q-2} + 2^{\frac{\log q}{2}-1}, 0) \cup (0, 2^{\log q-2} + 2^{\frac{\log q}{2}-1}) \in (-2^{\log q-2} - 2^{\log q-2}, 0) \cup (0, 2^{\log q-1} + 2^{\log q-1}) = (-\frac{q}{2}, 0) \cup$

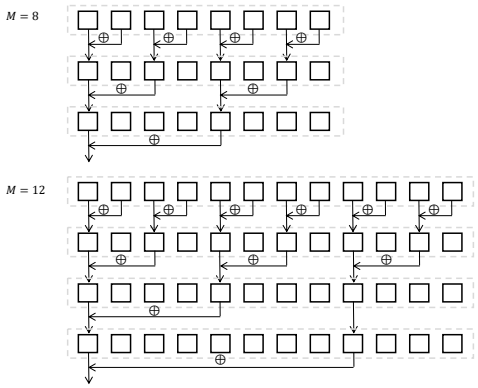



Fig. 11: A toy example of `I-PackFeatureSel` for $M = 8$ and $M = 12$.

TABLE IX: The Truth Table for Oblivious Comparison.

| $X'[i]-Y[i]$ | $R[i]$ | $V[i]$ | $V'[i]$ | $C'[i]$ | $C[i]$ |
|---|---|---|---|---|---|
| $<0$ | $-1$ | $>0$ | 1 | 1 | 0 |
| $<0$ | 1 | $<0$ | 0 | 0 | 0 |
| $\geq 0$ | $-1$ | $<0$ | 0 | 1 | 1 |
| $\geq 0$ | 1 | $>0$ | 1 | 0 | 1 |

$(0, \frac{q}{2})$. Clearly, when the sign of $A[i] \cdot (X'[i] - Y[i]) + B[i]$ changes, its value still remains within the range $(-\frac{q}{2}, 0) \cup (0, \frac{q}{2})$. This ensures that the sign of $A[i] \cdot (X'[i] - Y[i]) + B[i] \mod p$ is only affected by $X'[i] - Y[i]$ and $R[i]$. Because $X'[i] - Y[i]$ and $R[i]$ are random, we have them in four cases: 1) $X'[i] - Y[i] < 0$, 2) $X'[i] - Y[i] \geq 0$, 3) $R[i] = -1$, and 4) $R[i] = 1$. We show the true table in TABLE IX. It is evident that when $X'[i] - Y[i] < 0$, $C[i] = 0$, and when $X'[i] - Y[i] \geq 0$, $C[i] = 1$. By leveraging SIMD technique, all comparison results satisfy this condition. Therefore, Theorem 2 is correct. □

### C. The Correctness of PackPathEva

*Theorem 3:* After executing PackPathEva, the encrypted evaluation result vector $[\![T]\!]$, where one element in $T$ is the actual weight value, and the remaining elements are 0.

*Proof.* Similarly, we take the case of one node as an example. In the original setup, the left cost of the $i$-th node is set to $C[i]$, while the right cost of $i$-th node is set to $1 - C[i]$. Now, on the client side, the left cost of the $i$-th node is set to $C[i] + R'[i]$ and the right cost of $i$-th node is set to $1 - C[i] - R'[i]$. On the server side, the left cost of the $i$-th node is set to $-R'[i]$ and the right cost of $i$-th node is set to $R'[i]$. When the costs corresponding to the client and server are added together, the left and right costs of the $i$-th node are recovered. Therefore, when summing along each path, only one path will yield a sum of 0, while the sums of the other paths will be greater than 0. Next, we utilize PackObliviousCom to convert the path cost-based results into polynomial-based results. It is obvious that taking $(-[\![I]\!], 0)$ as the input of PackObliviousCom can finish the operation. This ensures that the result at the position of the real weight is 1, while the results at all other weight positions are 0, which means that the encrypted evaluation result vector $[\![T]\!]$ can be obtained. Therefore, Theorem 3 is correct. □

## APPENDIX C
## SECURITY ANALYSIS

In model inference, it is generally unnecessary to assume an adversary that actively tampers with model correctness. If a client behaves maliciously, any resulting inaccuracies are their own responsibility. Conversely, if a service provider intentionally modifies inference results, it may degrade the user experience, erode user trust, and ultimately damage the provider's reputation and commercial value. Therefore, adopting the semi-honest assumption is reasonable in the scenario. Based on this assumption, we give the security definition and the corresponding security analysis of Kangaroo.

### A. Security Definition

We use the security definition of simulation-based real/ideal worlds model [81] for two-party computation (2PC). In the real/ideal worlds, there are two probabilistic polynomial time (PPT) adversaries $\mathcal{A}_1$ and $\mathcal{A}_2$ that corrupt party A and party B. The real world is consistent with our system model. All messages that $\mathcal{A}_1$ and $\mathcal{A}_2$ can view are the same as that they can view in our scenario. In the ideal world, there is a simulator $\mathcal{S}_1$ with $\{\mathcal{L}_{1,2}, \mathcal{L}_1\}$ and a simulator $\mathcal{S}_2$ with $\{\mathcal{L}_{1,2}, \mathcal{L}_2\}$, where $\mathcal{L}_{1,2}$ is leakage of our scheme to party A and party B, $\mathcal{L}_1$ leakage of our scheme to party A, and $\mathcal{L}_2$ leakage of our scheme to party B. The simulator $\mathcal{S}_1$ (resp. $\mathcal{S}_2$) will simulate the messages that $\mathcal{A}_1$ (resp. $\mathcal{A}_2$) views in the ideal world. If $\mathcal{A}_1$ and $\mathcal{A}_2$ can only distinguish between the real and ideal worlds with negligible probability, then the private data ① - ⑨ remains protected, and our scheme is secure under the leakages $\mathcal{L}_{1,2}, \mathcal{L}_1, \mathcal{L}_2$.

### B. The Security of Kangaroo for Client-Server Model

In client-server model, party A is the client and party B is the server. We give the leakages to the client and server.

- Leakages to both client and server. $\mathcal{L}_{1,2}$ includes 1) the public parameters of cryptosystem: pp; 2) the public key: $\{(\mathsf{b}, \mathsf{a})\}$; 3) the feature dimension: $M$; 4) the obfuscated model structure indices: $\{\mathcal{T}^*_{k,s}\}_{k=1}^K$; 5) the precision parameter $\zeta$.
- Leakages to client only. $\mathcal{L}_1$ includes the private key $(1, \mathsf{s})$.
- Leakages to server only. $\mathcal{L}_2$ includes 1) the packed model $\{\mathbf{Y}^{pack}_\gamma, \mathbf{W}^{pack}_\gamma, \Upsilon^{pack}_\gamma, \Psi^{pack}_\gamma\}$; 2) the feature indices $\{\mathbf{M}_k\}_{k=1}^K$.

Based on the above leakages, we construct an ideal world model with two adversaries $\{\mathcal{A}_1, \mathcal{A}_2\}$, a simulator $\mathcal{S}_1$ with $\{\mathcal{L}_{1,2}, \mathcal{L}_1\}$ and a simulator $\mathcal{S}_2$ with $\{\mathcal{L}_{1,2}, \mathcal{L}_2\}$. Then, we formally define the security of Kangaroo.

> *Definition 1 (Security of Kangaroo):* The Kangaroo is selectively secure iff for any two PPT adversaries $\{\mathcal{A}_1, \mathcal{A}_2\}$, there exist two simulators $\{\mathcal{S}_1, \mathcal{S}_2\}$ with $\{\mathcal{L}_{1,2}, \mathcal{L}_1, \mathcal{L}_2\}$ to simulate an ideal world such that $\{\mathcal{A}_1, \mathcal{A}_2\}$ distinguish the views from the real world or the ideal world with negligible probability.

*Theorem 4:* Kangaroo is selectively secure with the $\{\mathcal{L}_{1,2}, \mathcal{L}_1, \mathcal{L}_2\}$.

*Proof.* Due to space limitation, the detailed security analysis can be found in the full version [83]. Our analysis shows that in all phases of Kangaroo, both $\mathcal{A}_1$ and $\mathcal{A}_2$ can only distinguish the real and ideal world with negligible probability, and the private data ① - ⑨ are protected. Therefore, our Kangaroo is selectively secure with $\{\mathcal{L}_{1,2}, \mathcal{L}_1, \mathcal{L}_2\}$ and provides the same security as these schemes [9]–[11], [14]–[16], [18]. In addition, to mitigate risks posed by arbitrary or abusive queries, we recommend integrating query-based payment and rate-limiting mechanisms into Kangaroo. □

## APPENDIX D
## ARTIFACT APPENDIX

Kangaroo is a high-performance framework designed for secure decision tree inference in two key scenarios: 1) Two-Party Secure Inference: Enables privacy-preserving decision tree inference between two parties; 2) Single-Cloud Outsourced Model Inference: Supports secure and efficient decision tree inference for outsourced models in a single-cloud environment. This artifact allows users to evaluate the availability and runtime latency of Kangaroo.

### A. Description & Requirements

*1) How to access:* Our implementation is available on Zenodo with DOI: https://doi.org/10.5281/zenodo.17055770. Alternatively, we also make the artifact available on GitHub, as stated in the manuscript.

• Kangaroo is a high-performance framework designed for secure decision tree inference in two scenarios:

**i) Two-Party Secure Inference:** Enables privacy-preserving decision tree inference between two parties. (Key scenario, provided for correctness and efficiency.)

**ii) Single-Cloud Outsourced Model Inference:** Supports secure and efficient decision tree inference for outsourced models in a single-cloud environment. (Not key scenario, provided for efficiency.)

• Repository Structure

**i) Kangaroo for client-server model-correct** - Verify the correctness of the scheme.

**ii) Kangaroo for client-server model-time** - Measure the time overhead of a two-way security reasoning scheme.

**iii) Kangaroo for client-server model-communication** - Measurement communication overhead of a two-way security reasoning scheme.

**iv) Kangaroo for outsourcing model-time** - Measure the time overhead of a Single-Cloud Outsourced security reasoning scheme.

**v) Kangaroo for outsourcing model-communication** - Measurement communication overhead of a Single-Cloud Outsourced.

**vi) Kangaroo for WAN** - Deploying Kangaroo in Real-World Applications.

*2) Hardware dependencies:* None

*3) Software dependencies:* **Microsoft SEAL**, Github: https://github.com/microsoft/SEAL

*4) Benchmarks:* We place our datasets under the UCI_dectrees directory and provide a test.py script to visualize the structure of each decision tree model. The models include boston, breast, decision_tree, diabetes, digits, iris, linnerud, and wine. By executing the test.py script, a png image showing the structure of the corresponding model will be generated. In the main paper, we focus on three larger datasets, boston, digits, and diabetes, to align with our system's performance and scalability goals. However, in this artifact, we include experiments on all datasets to comprehensively validate the correctness and robustness of the Kangaroo system.

To demonstrate Kangaroo's capability under real-world WAN conditions, we deploy servers across different cloud platforms. One server is hosted on JD Cloud, equipped with single thread, 3.8 GB RAM, and an Intel(R) Xeon(R) Gold 6148 @ 2.40 GHz processor, running Ubuntu 22.04.5. The other server is deployed on Alibaba Cloud, featuring a single CPU thread, 4 GB RAM, and an Intel(R) Xeon(R) Platinum @ 2.50 GHz processor, also running Ubuntu 22.04.5. The client is deployed on a ThinkPad-P53 machine with single thread, 23.1 GB RAM, and an Intel Core i5-9400H 2.50GHz processor running on Ubuntu 18.04.6.

As mentioned above, our code is designed to run on any Linux system. Therefore, for correctness testing, users can follow the provided README to run the Kangaroo for client-server model-correct configuration on a single Linux machine.

To evaluate network-level availability and functionality, the Kangaroo for WAN configuration needs to be executed on two separate machines, which can be any two Linux-based systems. For this purpose, and to protect sensitive information stored on our personal laptops, we provide access to two cloud servers for testing under realistic network conditions. These servers allow reviewers to reproduce our WAN-based experiments and verify that our system is functional and executable in real-world deployment settings.

• Alibaba Cloud, SSH + vscode, root, ip: 8.139.254.165. Password: P@ssw0rd

• JD Cloud, SSH + vscode, root, ip: 117.72.114.221. Password: P@ssw0rd

Due to space limitation, we have provided the step-by-step instructions for local IP configuration in the readme.md on GitHub and the archived artifact on Zenodo.

### B. Artifact Installation & Configuration

• Required packages for Kangaroo

**i) g++**

**ii) make**

**iii) seal** (https://github.com/microsoft/SEAL)

We use g++(11.4.0), make(4.3), seal(4.1).

### C. Experiment Workflow

To ensure reproducibility, we include the entire experiment workflow directly in this Evaluation section.

### D. Major Claims

We summarize our major claims as follows:

- (**Completeness and Correctness**): SYSTEM implements a two-party secure inference protocol for privacy-preserving decision tree evaluation. The protocol comprises three main phases: secure feature selection (including non-interactive selection and interactive selection, as shown in Algorithm 1 and Fig. 10), oblivious comparison (Fig. 2), and secure path evaluation (Fig. 8). By combining these components, our ciphertext-based inference can be correctly and securely executed.

  To demonstrate the correctness of our protocol, we provide a full implementation under the **Kangaroo for**


Fig. 12: Correctness Test.

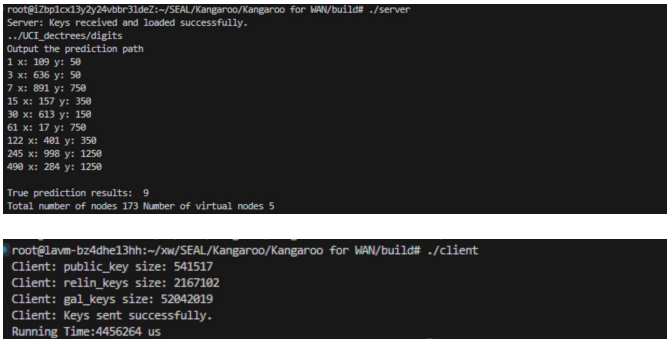



Fig. 13: Exercisability Test.

**client-server model-correct** directory. The main file contains annotated code corresponding to each protocol step:

**i) Comment 2.1** corresponds to secure feature selection (Comment 2.1.1 → Algorithm 1, non-interactive selection; Comment 2.1.2 → Fig. 10 interactive selection)

**ii) Comment 2.2** → oblivious comparison (Fig. 2)

**iii) Comment 2.3** → secure path evaluation (Fig. 8)

To verify the correctness of our protocol, we compare the plaintext and ciphertext prediction results. The plaintext inference logic is implemented starting from line 938, which uses the decision tree model to predict on a given plaintext input vector x. The output of ciphertext-based inference is decrypted at line 1062 and compared to the plaintext result. Matching outputs confirm the functional correctness and integrity of our secure inference protocol.

- (**Exercisability**): SYSTEM can be executed either locally or over a real WAN. For local testing, all programs can be directly executed on a single Linux machine. To further demonstrate the deployability and practicality of our system in real network environments, we provide a WAN-based deployment example using our core client-server inference protocol. This setup illustrates that our system is functional and executable across two physically separated machines communicating over the internet.

### E. Evaluation

*1) Experiment for Completeness and Correctness:* We provide the details for completeness and correctness.

*[How to]* To facilitate the evaluation process, we have deployed our system on an Alibaba Cloud server. For each subdirectory, we provide brief usage instructions to help evaluators execute the corresponding code smoothly.

*[Execution]* As an example, for functional correctness testing, evaluators can:

Log in to our cloud server via SSH.

Navigate to the directory: **cd /SEAL/Kangaroo/Kangaroo for client-server model-correct/**

Execute the script: **./1.sh** This will run both plaintext and ciphertext inference on all supported models, and print the corresponding outputs to allow direct comparison and verification.

*[Results]* We show a running result as Fig. 12.

Similarly, by following the same procedure in the communication and runtime directories as described in Experiment for Completeness and Correctness, evaluators can reproduce the experiments for measuring the communication overhead and runtime performance of our scheme.

*2) Experiment for Exercisability:* We provide the details for exercisability.

*[How to]* To facilitate the evaluation process under WAN network, we have deployed the client on an JD Cloud server and have deployed the server on an Alibaba Cloud server. We provide brief usage instructions to help evaluators execute the corresponding code smoothly.

*[Execution]* Log in to our cloud servers via SSH.

Navigate to the directory for Alibaba Cloud: **cd /SEAL/Kangaroo/Kangaroo for WAN** Execute the script: **./server**

Navigate to the directory for JD Cloud: **cd /xw/SEAL/Kangaroo/Kangaroo for WAN** Execute the script: **./client**

The above execution runs the Kangaroo system over a real wide-area network. Due to the use of JD Cloud and its bandwidth constraints, the execution time may be slightly slower than local testing.

*[Results]* We show a running result as Fig. 13.

Please ensure that **./server** is executed before running **./client**. If you encounter any issues during the evaluation process, feel free to contact us for assistance.