A New Paradigm for Privacy-Preserving Decision Tree Evaluation

Tianpei Lu*†, Bingsheng Zhang*†, Hao Li* and Kui Ren*
*The State Key Laboratory of Blockchain and Data Security, Zhejiang University,
Email: {lutianpei, bingsheng, kuiren}@zju.edu.cn.
†Hangzhou High-Tech Zone (Binjiang) Institute of Blockchain and Data Security

Abstract—**Privacy-preserving decision tree inference is a fundamental primitive in privacy-critical applications such as healthcare and finance, yet existing protocols rely heavily on secure selection, which accounts for more than half of the total cost. We introduce a new paradigm that eliminates this limitation by replacing multiple secure selections with a single permutation, whose cost is comparable to that of a single secure selection. Our scheme significantly reduces both computation and communication overhead compared to SOTA. Comprehensive benchmarks show an 86 % reduction in model evaluation versus the state-of-the-art FSS protocol by Ji et al., and a 99.9 % reduction versus the OT-based protocol of Ma et al. Overall, our benchmark shows that our protocol achieves a performance improvement of $20\times$ over Ma *et al.*'s scheme and $4.5\times$ over Ji *et al.*'s scheme.**

1. Introduction

Decision trees are widely used in machine learning for their interpretability and efficiency in classification and regression tasks. However, in privacy-sensitive applications such as healthcare, finance, or personalized recommendations, the input features and the decision tree model may both contain sensitive information that must be kept confidential. This raises the need for a privacy-preserving mechanism [5], [15], [28], [30], which allows the user to obtain the prediction result without revealing their private input, and without learning unnecessary information about the model.

Without considering security, a decision tree evaluation process roughly works as follows. It proceeds by starting at the root node and following a path to a leaf node based on the features involved in each node along the path. Each node is a decision point, where a feature is fetched and been evaluated by a predicate; its outcome determines whether shall go to the left child or the right child. This process continues until a leaf is reached, and its label is returned as the output. In typical privacy-preserving decision tree evaluation protocols, it is essential to protect the confidentiality of the model parameters (e.g., thresholds and feature indices), the structure of the tree (e.g., branching logic), and the traversal path, which may otherwise leak information about the input feature(s) and/or the model.

Privacy-preserving decision tree evaluation mainly relies on secure comparison [21], [22], [32] and oblivious selection [11], [24]. Secure comparison is used to evaluate whether a feature value exceeds a threshold privately. Oblivious selection is used not only to choose the correct branch based on the comparison result but also to select the feature specified at each decision node privately. Based on cost characteristics, privacy-preserving decision model evaluation can be divided into two categories. The first is depth-linear round schemes [3], [4], [10], [13], [18], [25], [27], which closely follow the plaintext evaluation logic by processing the tree level by level. At each node, the feature index is used to select the corresponding feature, which is then compared with the threshold to determine the next node to visit. The second approach is constant-round schemes [6], [7], [8], [9], [16], [26], [29], which typically pad the tree to a full binary structure and perform comparisons at all nodes in parallel. They then evaluate, for each branch, whether it is the correct one to follow. Although constant-round protocols reduce interaction latency, they incur significantly higher communication overhead compared to depth-linear round protocols.

The related techniques of mainstream privacy-preserving decision tree evaluation can be categorized into the following types:

Oblivious Transfer. Some works [25], [27] leverage Oblivious Transfer (OT) to implement oblivious selection, and employ techniques such as garbled circuits to evaluate secure comparisons. These primitives are then combined to build the secure decision tree evaluation scheme. However, OT-based oblivious selection typically incurs per-node costs that are linear in the number of features or nodes. As a result, for a tree of depth d and n features, the overall complexity is approximately $O(dn\lambda)$, where λ is the security parameter, typically set to 128 or higher.

Function Secret Sharing. A large body of work [4], [14], [17] has adopted Function Secret Sharing (FSS) to implement oblivious selection and secure comparison. The main advantage of FSS-based approaches is their sublinear communication complexity. For selecting among n secret-shared elements of a tree of depth d, the communication cost is only $O(d \log n\lambda)$. However, due to the dependence on the security parameter λ (typically, $\lambda \geq 128$), it still incurs a significant overhead in practice.

Fully Homomorphic Encryption. Fully homomorphic encryption (FHE)–based approaches [19], [26] offer better asymptotic complexity, as they theoretically require sending only encrypted feature values, resulting in a communication cost of $O(n\lambda)$, independent of the tree depth d. However,

TABLE 1: Performance comparison: m is the number of decision nodes, n is the number of features, ℓ is the bit-length of feature and classification value, λ_1 is the size of symmetric ciphertext (= 128), λ_2 is the size of Paillier ciphertext (= 4096), λ_3 is the size of AES key (= 128).

Scheme	Communication		Rounds
	offline	online	
Cock [10] *et al.*	$6(2^d n\ell + d(3\ell - \log l - 2) + 2^d - 1)$	$4(2^d n\ell + d(3\ell - \log l - 2) + 2^d - 1)$	$\log \ell + d + 1$
Zheng [31] *et al.*	$6((2^d n + 4)\ell - 5)$	$4((2^d n + 4)\ell - 5)$	$2\ell - 1$
Liu [20] *et al.***	$6(2^d - 1)\ell$	$3 \cdot 2^{d-1}\lambda_2 + 4(2^d - 1)\ell$	$d + 1$
Tueno [27] *et al.* (OT)	$6d\ell\lambda_1$	$d((m + n)\ell + 2(\log m + \log n)\lambda_1)$	$4d$
Ma [25] *et al.* (complete)	$2^d(\ell + \log n)$	$d(4\lambda_1 + n\ell + (7\ell + 8)\lambda_1)$	$2d - 1$
Ma [25] *et al.* (sparse)	$m(\ell + \log n + \lambda_1 + 3d)$	$d((4\lambda_1 + n\ell) + (7\ell + 8)\lambda_1 + 8)$	$2d - 1$
Ji [17] *et al.*	$12d(\log n + \log m + \ell)\lambda_3$	$12d(3(\log m + \log n) + 2\ell)$	$2d - 1$
Ours Π_{strawman}	$2m\ell + (4n + 3)(\log n + \log m)$	$(d - 1)(14 \log m + 14 \log n + 4\ell \log \ell) + 2n\ell$	$3d - 2$
Ours Π_{strawman} (DCF)	$2m(\ell + 2(\log m + \log n)) + 6\lambda\ell$	$(d - 1)(12\ell + 9(\log n + \log m)) + 2n\ell$	$2d - 1$
Ours Π_{tree}	$66m \log m + 6n \log n + 60m \log n$ $+3(\log n + \log m) + 2m\ell$	$(d - 1)(14 \log m + 14 \log n + 4\ell \log \ell)$ $+15n \log n$	$3d + 2$
Ours Π_{tree} (DCF)	$66m \log m + 6n \log n + 60m \log n$ $+3(\log n + \log m) + 6\lambda\ell$	$(d - 1)(12\ell + 9(\log n + \log m)) + n(2\ell + 15 \log n)$	$2d + 3$

** Those protocols do not hide the feature index from the servers.

due to the high computational overhead and the significant security parameter λ, FHE-based protocols tend to be inefficient in practice. In particular, experimental results by Ji [17] *et al.* show that such methods perform substantially worse than FSS-based schemes in real-world settings.

An analysis of the above approaches reveals that oblivious selection at each layer constitutes the primary performance bottleneck. In this work, we aim to eliminate the costly overhead of oblivious selection in decision processes.

A new PPDT paradigm. We observe that several oblivious selections can be replaced with a single oblivious permutation if the selection choices are distinct. To hide the decision tree structure, we shuffle the tree nodes and all the features randomly such that revealing node indices does not compromise privacy. Therefore, the oblivious selections at each decision point can be eliminated. The reason for choosing permutation over selection lies in the fact that, typically, the cost of a secure permutation is similar to that of an oblivious selection. For example, in the three-party setting, both secure permutation [2] and oblivious selection [23] on n elements of ℓ bits incur the communication cost of $O(n\ell)$. As noted earlier, FSS can reduce complexity to $\log n$ but requires a large security parameter λ, causing even higher communication overhead. Overall, reducing tree depth d rounds of oblivious selection to a single secure permutation can be regarded as eliminating the cost of $(d-1)$ oblivious selections.

More specifically, in a shuffled decision tree, both the threshold and the child indices of each node are permuted. After evaluating each node and determining whether to proceed to the left or right child, one can directly reveal the branching result to select the corresponding node or feature in plaintext. The node shuffling itself is straightforward and can be implemented using a standard permutation. The challenge lies in replacing the original indices of the child nodes with their permuted counterparts within each node.

First attempt. A straightforward idea is to treat the permutation applied to the permuted nodes as a list and use each node's original child indices to perform oblivious selection. That is, generate a permutation $\pi : [n] \mapsto [n]$, and for each child index $j \in [n]$ in a node, substitute it with $\pi(j)$. However, this brings us back to the initial problem, as it requires oblivious selection operations; namely, oblivious selection of $\pi(j)$ for each secret-shared j, performing one oblivious selection per node. A natural question arises: can we use a single permutation to update all child indices at once? Intuitively, directly applying a single permutation is problematic. To ensure that an adversary cannot distinguish whether the node is a leaf node or a decision node, every node, including leaf nodes, is assigned two child indices that point to a dummy node. As a result, a tree with n nodes yields $2n$ child indices, and many of these indices are duplicated. These repetitions prevent the list of child-node indices from appearing as the output of a permutation.

We observe that the above issue can be reformulated as follows: Given a vector $\pi(1), \ldots, \pi(n)$ corresponding to permuted indices, we replicate them according to the required duplicates to obtain $2n$ indices. These $2n$ items then need to be "placed" into the $2n$ child-index positions of the n nodes. Placing $2n$ items into $2n$ positions can be achieved with a single permutation ζ. To realize this repetition process, we introduce a new primitive called Oblivious Vector Expansion (OVE). OVE takes a list and, following a predefined rule, duplicates certain elements to produce an expanded list. The model owner knows which nodes (or feature indices) need to be reused as child indices, which defines the duplication rule of OVE. We further find that the duplication rule can also be represented by another permutation σ (see Section 4). Thus, the relationship between parent and child nodes can be expressed using two permutations, ζ and σ. In our new framework, we abandon the traditional node-based representation of decision trees. Instead, the tree owner represents a decision tree using two permutations (ζ, σ) and a threshold list $(t_i)_{i \in \mathbb{Z}_n}$.

With this representation, a shuffled decision tree can be efficiently generated as follows: first, generate a random permutation π of length n; next, input σ to OVE to expand $(\pi(1), \ldots, \pi(n))$ into $2n$ length; then, use ζ to permute each item to the correct child index positions; finally, insert the thresholds t_i into the nodes and apply π to shuffle the n nodes, producing the final shuffled decision tree. By using different π, we can reuse σ, ζ, and t_i to generate different shuffled decision trees.

In summary, this paper makes the following contributions:

- **A new primitive—Oblivious Vector Expansion (OVE)**. We propose a new primitive–Oblivious Vector Expansion. This primitive enables repeating the item at some given indices in a list with arbitrary times, without revealing either the index or the replicated times. We design a constant round protocol to realize such functionality.
- **A novel privacy-preserving decision tree evaluation paradigm.** We design a new PPDT evaluation paradigm, where the decision tree is represented using permutations. Under this representation, we can generate a shuffled decision tree efficiently, which significantly reduces the communication cost and running time compared to the state-of-the-art.

PPDT-as-a-service. We focus on the privacy-preserving decision-tree evaluation-as-a-service setting, where a deployed model serves a large number of users. In this scenario, we assume the model is known in advance and deployed on the computation servers in secret-shared form, while in the online phase, users provide secret-shared features for inference.

Performance. Our protocol significantly reduces both the communication and computation overhead required for decision tree evaluation. Table 1 presents a comparison between our approach and prior works. Unlike existing schemes, our protocol eliminates the need for oblivious selection mechanisms. For example, with a tree of depth 7, 128 nodes, and 24 features, our online phase reduces communication cost by 50% compared to Ji [17] *et al.*, and achieves only 0.1% of the communication cost required by Ma [25] *et al.* Moreover, while Ji et al.'s protocol relies heavily on costly DCF and DPF computations, our protocol requires only a small number of memory accesses—equal to the depth of the tree. If we further substitute our COT protocol with DCF, the online communication cost of our protocol drops to just 14% of that in Ji *et al.*. Overall, our benchmark shows that our protocol achieves a performance improvement of $20\times$ over Ma *et al.*'s scheme and $4.5\times$ over Ji *et al.*'s scheme.

Paper organization. Section 2 introduces the preliminaries, including notations and the cryptographic primitives used to construct our framework. In Section 3, we present our strawman protocol and highlight its main limitation—namely, the requirement of three computing parties in addition to the model and data holders. To overcome this, we propose a new PPDT paradigm in Section 5. We also provide a detailed analysis of the overhead and security of our scheme in the same section. Finally, Section 6 presents experimental results demonstrating the efficiency of our proposed paradigm.

2. Preliminaries

Notation. The frequently used notations are shown in Table 2. We denote the integer set $\{0, \ldots, k-1\}$ as $\mathbb{Z}_k$. We denote the feature holder as $\mathcal{C}$, the tree model holder as $\mathcal{S}$, the computing parties as $\mathcal{P} := (P_0, P_1, P_2)$. We denote the feature list as $\mathcal{X} := (x_0, \ldots, x_{n-1})$, and n for feature list length. We denote the decision tree model as $\mathcal{T}$, which is constructed by the set of nodes $(N_0, \ldots, N_{m-1})$, and the corresponding feature index of the root node as α. For bijective functions representing random permutations, we use symbols such as π, ζ, or σ. We use $[n, n+k]$ to denote the set $\{n, \ldots, n+k\}$. A random permutation $\pi \in S_n \mapsto S_n$ is denoted as the vector of destinations:

$$\pi = (\pi(0), \ldots, \pi(n-1)).$$

When π is applied to a list $\mathcal{X} := x_0, \ldots, x_{n-1}$, each element x_i is moved to position $\pi(i)$. We use the notation $\pi(\mathcal{X})$ to represent the list obtained by applying π to $\mathcal{X}$. We denote the inverse of a permutation π as π^-. Obviously, given $\mathcal{X}' = \pi(\mathcal{X})$, it follows that each element x'_i in $\mathcal{X}'$ satisfies $x'_i = x_{\pi^-(i)}$. We denote the permutation composition for π and π' as $\pi \circ \pi'$, and it holds that $\pi \circ x_{\pi^-} = id$, while id is the identity permutation $id = (0, 2, \ldots, n-1)$.

TABLE 2: Notations

Notations	Descriptions
$\mathcal{X}$	The feature list $\mathcal{X} := (x_0, \ldots, x_{n-1})$.
$\mathcal{T}$	The representation of a decision tree.
α	The root feature's index of the decision tree.
$\mathcal{C}$	The client who input the feature list.
$\mathcal{S}$	The server that inputs the decision tree.
$\mathcal{P} := (P_0, P_1, P_2)$	The computing parties that evaluate the tree.
$\mathbb{Z}_k$	The integer set $\{0, 1, \ldots, k-1\}$.
$\langle x \rangle := (\langle x \rangle_0, \langle x \rangle_1, \langle x \rangle_2)$	The 3PC secret shares of x over $\mathbb{Z}_{2^\ell}$ where $x = \langle x \rangle_0 + \langle x \rangle_1 + \langle x \rangle_2 \pmod{2^\ell}$.
π, ζ, σ	Some permutation list.
$\eta_{i,j}$	The random seed hold by P_i and P_j.

Our Secure Sharing Scheme. We consider typical 2-out-of-3 secret shares (3PC replicated secret sharing scheme) and define the secret share $\langle \cdot \rangle^\ell$ over ring $\mathbb{Z}_{2^\ell}$ as $\langle \cdot \rangle^\ell := (\langle \cdot \rangle^\ell_0, \langle \cdot \rangle^\ell_1, \langle \cdot \rangle^\ell_2)$ where $x = \langle x \rangle^\ell_0 + \langle x \rangle^\ell_1 + \langle x \rangle^\ell_2 \pmod{2^\ell}$. When the meaning is clear from context, we omit the superscript and write $\langle x \rangle$. Each party P_i among three parties holds two shares $\langle x \rangle_{i-1}$ and $\langle x \rangle_{i+1}$. For $\langle \mathcal{X} \rangle$ where $\mathcal{X}$ is a vector or a set, we represent the vector or set of each secret-share of its elements, namely, $\langle \mathcal{X} \rangle := (\langle x_0 \rangle, \ldots, \langle x_{n-1} \rangle)$. We define the addition on the secret share as $\langle z \rangle = \langle x \rangle + \langle y \rangle$ and it holds that $z = x + y$ in secret shared form. P_i locally executes $\langle z \rangle_{i-1} = \langle x \rangle_{i-1} + \langle y \rangle_{i-1}$ and $\langle z \rangle_{i+1} = \langle x \rangle_{i+1} + \langle y \rangle_{i+1}$ to obtain the shared result. We define the secret share protocol and the reconstruct protocol as follows.

- $\langle x \rangle \leftarrow \Pi_{\mathcal{P}}(\mathbb{P}, x)$: We define the secret share to the computing parties $\mathcal{P}$ as $\Pi_{\mathcal{P}}(\mathbb{P}, x)$, where $\mathbb{P} \in \{\mathcal{C}, \mathcal{S}\}$ holds x and secret shares x to $\mathcal{P}$. Before execution, $\mathbb{P}$, P_0 and P_1 hold the same correlated seed η_2; $\mathbb{P}$, P_0 and P_2 hold the same correlated seed η_1. In the offline phase, $\mathbb{P}$, P_0 and P_1 locally generate $\langle x\rangle_2$ via PRG using seed η_2; $\mathbb{P}$, P_0 and P_2 locally generate $\langle x\rangle_1$ via PRG using seed η_2. $\mathbb{P}$ locally sets $\langle x\rangle_0 = x - \langle x\rangle_2 - \langle x\rangle_2$ and sends it to both P_1 and P_2.

- $x \leftarrow \Pi_{\mathsf{rec}}(P_i, \langle x\rangle)$: We define the reconstruction of share $\langle x\rangle$ to the P_i as $\Pi_{\mathsf{rec}}(P_i, \langle x\rangle)$. P_{i-1} sends $\langle x\rangle_i$ to P_i. P_i reconstruct $x = \sum_{j=0}^{2}\langle x\rangle_j$

System Architecture and Threat Model. This work aims to protect the privacy of decision tree evaluation. Specifically, this work aims to 1). Protect the confidentiality of the decision tree model $\mathcal{T}$ and the input feature vector $\mathcal{X}$; 2). Protect the decision path length. Since the number of evaluation rounds corresponds to the depth of the path taken in the tree, exposing this information may allow an adversary to infer which branch was followed; 3). Protect access pattern over both $\mathcal{T}$ and $\mathcal{X}$. In particular, no party should learn which node or which feature is selected at each evaluation step. This work does not protect the maximum depth d of the decision tree, the number of nodes m, or the number of features n. We assume all the participants are semi-honest without collusion, where the adversary may attempt to extract private information from her view, but she must follow the protocol. In particular, our framework contains three roles, $\mathcal{C}$ is the feature holder, $\mathcal{S}$ is the decision tree model holder, and $\mathcal{P}$ are the computing parties, containing three parties $\mathcal{P} := \{P_0, P_1, P_2\}$. In our strawman protocol, $\mathcal{S}$ and $\mathcal{C}$ are separated from $\mathcal{P}$. Then, in our new paradigm, $\mathcal{S}$ and $\mathcal{C}$ can both be one party of $\mathcal{P}$ or separated from $\mathcal{P}$.

Secure Shuffle. Secure shuffle refers to a cryptographic protocol that randomly permutes a list of (securely shared) elements in a way that conceals the original order from all participating parties, and no party knows the permutation. It plays a critical role in privacy-preserving computations. In this work, we define secure shuffle as the functionality $\mathcal{F}_{\mathsf{shuffle}}$, where $\mathcal{F}_{\mathsf{shuffle}}$ accepts a shared list $\langle x_0\rangle, \ldots, \langle x_{n-1}\rangle$, selects a random permutation π, and outputs the new shared list $(\langle x_{\pi^-(0)}\rangle, \ldots, \langle x_{\pi^-(n-1)}\rangle)$. Here, the permutation π is randomly selected and kept secret from all computing parties. None of the parties has any knowledge about π. In this work, we adopt the shuffle protocol proposed by Asharov [2] *et al.* to handle 3-party replicated secret sharing. Briefly, for shuffling $\langle \mathcal{X}\rangle$, their protocol can be described as follows: (i) P_0 and P_1 generate same random permutation π_0. P_0 perform π_0 on list $(\langle x_0\rangle_1 + \langle x_0\rangle_2, \ldots, \langle x_{n-1}\rangle_1 + \langle x_{n-1}\rangle_2)$, P_1 perform π_0 on $(\langle x_0\rangle_0, \ldots, \langle x_{n-1}\rangle_0)$. As a result, parties P_0 and P_1 hold the additively secret shares of the permuted values $\pi_0(\mathcal{X})$ after the shuffle. (ii) P_0 and P_1 convert the additive secret shares into replicated secret shares. For a single additive secret share $y_0 + y_1 = y$, where P_0 holds y_0 and P_1 holds y_1, to obtain the replicated secret sharing $\langle y\rangle$, the parties proceed as follows: P_0 and P_2 jointly generate share $\langle y\rangle_1$ with seed $\eta_{0,2}$; P_1 and P_2 generate $\langle y\rangle_0$; P_0 computes and sends $\Delta_0 = y_0 - \langle y\rangle_1$ to P_1; P_1 computes and sends $\Delta_1 = y_1 - \langle y\rangle_0$ to P_0; Finally, both P_0 and P_1 set $\langle y\rangle_2 = \Delta_0 + \Delta_1$. (iii) In the above procedure, P_2 has no knowledge of the permutation π_0. Similarly, P_1 and P_2 jointly generate a second permutation π_1 and repeat the same process, followed by P_2 and P_0 generating a third permutation π_2 and performing the procedure again. As a result, each computing party remains unaware of at least one permutation. To permute a list of length n over $\mathbb{Z}_{2^\ell}$, the above protocol requires 3 rounds of communication and incurs a total communication cost of $6n\ell$.

Secure Permutation. In our work, we make use of a secure permutation, which differs from a shuffle protocol in that the permutation is explicitly specified rather than randomly sampled. The functionality of a secure permutation, $\mathcal{F}_{\mathsf{per}}$, takes as input a secret-shared list $\langle \mathcal{Y}\rangle$ and a secret-shared permutation $\langle \pi\rangle$, and securely outputs the permuted result $\langle \pi(\mathcal{Y})\rangle$. Asharov [2] *et al.* implement an efficient secure permutation protocol from a secure shuffle protocol. For $\langle \mathcal{Y}\rangle$ and $\langle \pi\rangle$, its core logic is as follows:

- By applying the same secure shuffle simultaneously to both $\langle \pi\rangle$ and $\langle \mathcal{Y}\rangle$, we obtain $\langle \pi'(\pi)\rangle$ and $\langle \pi'(\mathcal{Y})\rangle$. It can be realized by apply same π_0, π_1 and π_2 on secure shuffle protocol, where $\pi' = \pi_2 \circ \pi_1 \circ \pi_0$.
- By revealing $\langle \pi'(\pi)\rangle$, we obtain the cleartext permutation $\pi'' = \pi'(\pi) = \pi \circ \pi'^{-}$(Observation 2.4 in Asharov [2] *et al.*). Then, each party can locally apply π'' to $\langle \pi'(\mathcal{Y})\rangle$ to obtain the final result $\langle \pi(\mathcal{Y})\rangle$.

The protocol requires two shuffle operations and one opening step, resulting in a total of 4 communication rounds (3 rounds for shuffle, 1 round for reconstruction) and a communication cost of $15n\ell$ bits. In this work, we use the notation $\langle \pi(\mathcal{Y})\rangle \leftarrow \Pi_{\mathsf{per}}(\langle \mathcal{Y}\rangle, \langle \pi\rangle)$ to denote the secure permutation protocol described above.

Secure Comparison. Secure Comparison is a cryptographic protocol that enables two or more parties to compare private values without revealing the actual values to each other. Lu [22] *et al.* implemented a one-round comparison protocol in the three-party replicated secret sharing setting. Fig. 10 (Cf. Appendix. A) depicts the functionality for their comparison protocol. $\mathcal{F}_{\mathsf{cmp}}[i]$ (i corresponds to that P_{i-1} and P_{i+1} hold same share) takes two secret-shared inputs $\langle a\rangle$, $\langle b\rangle$, and sends a bit $z_0 \in \{0, 1\}$ to party P_i, and a bit $z_1 \in \{0, 1\}$ to parties P_{i-1} and P_{i+1}. If $a \geq b$, then $z_0 \oplus z_1 = 1$; otherwise, $z_0 \oplus z_1 = 0$.

Conditional Oblivious Transfer. Conditional Oblivious Transfer (COT) is a variant of the classic Oblivious Transfer (OT) protocol in which the sender sends several messages and the receiver obtains one of the possible messages based on a private condition, without learning the other messages and without revealing the condition to the sender and receiver. In this work, we consider a 1-out-of-2 conditional oblivious transfer with public opening (COTP), where the condition is a comparison operation. That is, the condition and the messages are shared, and the output is publicly opened. Specifically, the COTP functionality $\mathcal{F}_{\mathsf{cotp}}$ takes as input two secret-shared values $\langle a\rangle$, $\langle b\rangle$, and two secret-shared selection vectors $\langle \mathcal{L}\rangle$, $\langle \mathcal{R}\rangle$. It outputs the plain vector $\mathcal{I}$ to all the parties, such that if $a \geq b$, then $\mathcal{I} = \mathcal{L}$; otherwise, $\mathcal{I} = \mathcal{R}$.

Strawman scheme $\Pi_{\mathsf{strawman}}(\mathcal{X}, \mathcal{T})$

$\mathcal{C}$ and $\mathcal{S}$ holds correlated seed η. The depth d, the number of nodes m of the decision tree, and the number of features n are publicly known.
Input : Client $\mathcal{C}$ inputs feature list $\mathcal{X} := (x_0, x_1, \ldots, x_{n-1})$; Server $\mathcal{S}$ inputs decision tree $\mathcal{T} := ((N_0, \ldots, N_{m-1}), \alpha)$.
Output : $\langle\cdot\rangle$-shared model evaluation result $\mathcal{R}$.

Model Shuffle:

1) $\mathcal{S}$ generates dummy node $N'_i := (0, i+m+1, i+m+1, i+n+1, i+n+1)$ for $i \in \mathbb{Z}_{d-2}$;
2) $\mathcal{S}$ appends dummy node to original nodes, namely, $\mathcal{T} := ((N_0, \ldots, N_{m-1}, N'_0, \ldots N'_{d-3}), \alpha)$;
3) For every leaf node N_e whose path to the root indicies $I_0, \ldots, I_{k-1}$, set $N_e := (t_e - \sum_{i=0}^{k-1} t_{I_i}, m, m, n, n)$.
4) $\mathcal{S}$ generates random permutation $\pi \leftarrow S_{m+d-2} \mapsto S_{m+d-2}$;
5) $\mathcal{S}$ generates random permutation $\pi' \leftarrow S_{n+d-2} \mapsto S_{n+d-2}$ using seed η;
6) For each node $N_i := (t_i, L_i, R_i, L'_i, R'_i)$, $\mathcal{S}$ updates it as $N_i := (t_i, \pi(L_i), \pi(R_i), \pi'(L'_i), \pi'(R'_i))$;
7) $\mathcal{S}$ sets $(N'_0, \ldots, N'_{m+d-3}) := \pi((N_0, \ldots, N_{m+d-3}))$
8) $\mathcal{S}$ invokes $\langle N'_i \rangle \leftarrow \Pi_{\mathcal{P}}(\mathcal{S}, N'_i)$ for $i \in \mathbb{Z}_{m+d-2}$ to secret-shares nodes to $\mathcal{P}$;
9) $\mathcal{S}$ sends $\mathcal{I}_0 \leftarrow \pi(0)$ and $\mathcal{I}_1 \leftarrow \pi'(\alpha)$ to $\mathcal{P}$;

Model Evaluation:

1) $\mathcal{C}$ generates random permutation $\pi' \leftarrow S_{n+d-2} \mapsto S_{n+d-2}$ using seed η, which is same as $\mathcal{S}$ holds.
2) $\mathcal{C}$ appends $d-2$ zero after $\mathcal{X}$, namely, set $x_i = 0$ for $i \in [m, m+d-3]$.
3) $\mathcal{C}$ sets $\mathcal{X} := \pi'(\mathcal{X})$, and invokes $\langle \mathcal{X} \rangle \leftarrow \Pi_{\mathcal{P}}(\mathcal{C}, \mathcal{X})$ to secret-shares features to $\mathcal{P}$
4) $\mathcal{P}$ sets $\langle \mathcal{R} \rangle = \langle 0 \rangle^{\ell}$
5) For $i \in \mathbb{Z}_{d-1}$, $\mathcal{P}$ do
    - set $\langle N_o \rangle \leftarrow \langle N_{\mathcal{I}_0} \rangle$; $\langle x_o \rangle^{\ell} \leftarrow \langle x_{\mathcal{I}_1} \rangle^{\ell}$;
    - parse $\langle N_o \rangle$ as tuple $\langle N_o \rangle := (\langle t_o \rangle^{\ell}, \langle L_o \rangle^{\log m}, \langle R_o \rangle^{\log m}, \langle L'_o \rangle^{\log n}, \langle R'_o \rangle^{\log n})$
    - update $\langle \mathcal{R} \rangle^{\ell} \leftarrow \langle \mathcal{R} \rangle^{\ell} + \langle t_o \rangle^{\ell}$
    - invoke COTP $(\mathcal{I}_0, \mathcal{I}_1) \leftarrow \Pi_{\mathsf{cotp}}(\langle x_o \rangle^{\ell}, \langle t_o \rangle^{\ell}, (\langle L_o \rangle^{\log m}, \langle L'_o \rangle^{\log n}), (\langle R_o \rangle^{\log m}, \langle R'_o \rangle^{\log n}))$
6) $\mathcal{P}$ calculate $\langle \mathcal{R} \rangle^{\ell} \leftarrow \langle \mathcal{R} \rangle^{\ell} + \langle t_{\mathcal{I}_0} \rangle^{\ell}$.

Figure 1: Strawman Protocol for Privacy-Preserving Decision Tree Evaluation.

3. Strawman Scheme

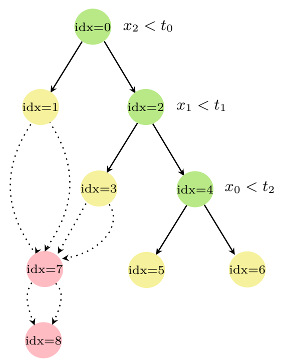
In this section, we first propose a strawman scheme. In our strawman scheme, the client holds the input features, while the model owner holds the decision tree model. They jointly engage a set of external computing parties to perform the secure evaluation of the tree model. Thanks to the separate roles of feature/model holders and computing parties, this scheme achieves high efficiency.
Remark: In most cases, a single path in a decision tree does not reuse the same feature more than once. During tree construction, the algorithm typically selects the most informative feature at each node and does not reconsider the same feature further down the same path. Therefore, in the scheme described in this section, we assume that each path does not contain repeated feature usage. We address the case where a feature may appear multiple times along the same path at the end of the paper (Cf. 4.3).

3.1. Shuffle-based Scheme.

Tree model. We first use the classic decision tree representation method. A decision tree $\mathcal{T}$ is construct by a collection of nodes $(N_0, N_1, \ldots, N_{m-1})$, where each node N_i encapsulates the structural and decision-making logic required for tree traversal. We denote the root node by N_0 and its corresponding feature index by α, so that the entire tree can be represented as $\mathcal{T} := ((N_i)_{i \in \mathbb{Z}_m}, \alpha)$. We define the input features as a list $\mathcal{X} = [x_0, x_1, \ldots, x_{n-1}]$, where each x_j represents a single feature value encoded in a finite ring $\mathbb{Z}_{2^{\ell}}$. Specifically, each node $N_i := (t_i, L_i, R_i, L'_i, R'_i)$ is defined by the following components:

- A threshold value $t_i \in \mathbb{Z}_{2^{\ell}}$, which is encoded in the finite ring $\mathbb{Z}_{2^{\ell}}$ and used to compare against a feature value in the input;
- A left child node index $L_i \in \mathbb{Z}_m$, indicating the next node to proceed to if the comparison result is true (e.g., feature value $< t_i$);
- A right child node index $R_i \in \mathbb{Z}_m$, indicating the next node if the comparison is false;
- A feature index for the left path L'_i, denoting which input feature is used in the comparison when traversing to the left;
- A feature index for the right path R'_i, used when traversing to the right.

Typical Private Decision Tree Evaluation. In typical private decision tree evaluation schemes, both the decision tree nodes and feature values are secret-shared among the computing parties. The evaluation proceeds in a layer-by-layer manner. At each layer, an oblivious selection protocol—such as a 1-out-of-n oblivious transfer—is employed to retrieve the current node and its associated feature based on the evaluation path. The parties then engage in a secure comparison between the selected feature value and the threshold embedded in the node. Depending on the



idx	t	L	R	L'	R'
0	t_0	1	2	1	2
1	$c_1 - t_0$	3	7	·	·
2	t_1	3	4	·	3
3	$c_2 - t_0 - t_1$	7	7	·	·
4	t_2	5	6	·	·
5	$c_3 - t_0 - t_1 - t_2$	/	/	·	·
6	$c_4 - t_0 - t_1 - t_2$	/	/	·	·
7	0	8	8	·	·
8	0	/	/	·	·

Figure 2: Decision tree structure and representation, c refers to the label of the leaf.

comparison outcome, the protocol securely determines the indices of the next node and feature to be evaluated in the following layer. This process is repeated iteratively until a leaf node is reached.

Private Decision Tree Evaluation with Shuffle. We observe that secure decision tree evaluation can be significantly simplified by leveraging random shuffling, effectively eliminating the need for 1-out-of-n selection of features or nodes. Specifically, considering a secret-shared decision tree node represented as $N_i := (t_i, L_i, R_i, L'_i, R'_i)$, if all nodes and features are randomly shuffled—using a permutation π for nodes and π' for features—the resulting node set $(N'_i)_{i \in \mathbb{Z}_m}$ becomes: $N'_{\pi(i)} = N_i := (t_i, \pi(L_i), \pi(R_i), \pi'(L'_i), \pi'(R'_i))$. In this setting, revealing $\pi(L_i)$ or $\pi(R_i)$ is secure, since without knowledge of the permutation π, the revealed value appears as a uniformly random index in the range $[0, m-1]$. The computing parties can then locally select the secret-shared node $N'_{\pi(L_i)}$ or $N'_{\pi(R_i)}$ using plaintext $\pi(L_i)$ or $\pi(R_i)$ for further evaluation, rather than expensive 1-out-of-m oblivious selection. A similar argument applies to feature indices. Then we employ the aforementioned $\mathcal{F}_{\mathsf{cotp}}$ to perform the secure comparison and directly open the node/feature index of the next layer. Since revealing the same index would leak information, the shuffled decision tree can only be used once. We employ a one-time-pad-like approach by using fresh random permutations π and π' for each evaluation.

Dummy Nodes. To preserve privacy, the evaluation of any branch in a sparse decision tree should appear indistinguishable in depth from that of a full-depth path. To achieve this, dummy nodes [17] are introduced to pad the evaluation path after a label is reached, ensuring that every evaluation proceeds to depth d. As illustrated in Fig. 2, for each shallow leaf node (i.e., one at depth $d' < d$), we append a chain of $d-2$ dummy nodes to extend its path to depth d. Since our protocol relies on a shuffled structure, revealing the same index during evaluation could lead to privacy leakage; therefore, $d-2$ dummy nodes are required (The worst case is that the child node of the root node is a leaf node). In particular, each dummy node is constructed with both its left and right child pointers referencing the next dummy node in the chain. Moreover, for all leaf nodes at depths less than d, their left and right child indices are redirected to the first dummy node in the appended sequence. Similarly, we need to assign a dummy feature to each dummy node.

In addition, we modified the way to obtain the final label. Considering that the final node obtained by evaluation on some branches will be a dummy node, we get the evaluation result by accumulating all weight values on the entire path. We only need to subtract the sum of the weights on the path from the label of each leaf node, and then set the weights of all dummy nodes to 0. Then the result accumulated on the entire path will be the label of the original leaf node. Fig. 2 illustrates the corresponding procedure.

Our Strawman Protocol. Fig. 1 illustrates our strawman protocol, which comprises two execution phases. The first is the model shuffling phase, in which the server $\mathcal{S}$ generates a shuffled tree model and secret-shares it to the computing parties $\mathcal{P}$. The second is the evaluation phase, where the client $\mathcal{C}$ provides the feature vector, and the computing parties $\mathcal{P}$ collaboratively evaluate the decision tree layer by layer. We now present a detailed step-by-step explanation of the strawman protocol. In the model shuffling phase, it performs the following steps.

- Steps 1-2: $\mathcal{S}$ generates $d-2$ dummy nodes and inserts them into the original set of nodes with indices ranging from m to $m+d-2$.
- Step 3: We modify the original leaf nodes by redirecting their child pointers to the first dummy node, and we update their label values to be the difference between the true label and the accumulated weight along the evaluation path. This ensures that the final label, obtained by summing all node values along the path, is correct.
- Steps 4-7: $\mathcal{S}$ shuffles the tree nodes using a random permutation and independently generates a permutation over the feature list. The feature indices in the tree nodes are updated according to the feature permutation.
- Steps 8-9: $\mathcal{S}$ secret-shares the shuffled decision tree model to $\mathcal{P}$ and sends $\mathcal{P}$ the starting index of the root node and feature.

In the model evaluation phase, the following steps are performed:

- Steps 1-3: The client $\mathcal{C}$ generates the same permutation π' as the server by using an identical seed. It then appends $d-2$ dummy features to the feature list and applies π' to shuffle the entire feature list.
- Steps 4-6: $\mathcal{P}$ perform the decision tree evaluation layer by layer. In each round, $\mathcal{F}_{\mathsf{cotp}}$ is used to obtain the next node index and corresponding feature index. These indices are then used to select the appropriate node and feature, which are input into $\mathcal{F}_{\mathsf{cotp}}$ in the subsequent round. This process is repeated for $d-1$ iterations to obtain the final node index, ultimately yielding the final label.

3.2. Conditional Oblivious Transfer with Public Opening.

We can directly use the one-round DCF-based SCOT protocol of Ji et al. [17] to instantiate our COTP functionality. Their SCOT takes two replicated secret shares x and y, along with secret-shared indices a and b, and outputs a secret sharing of either a or b depending on the comparison of x and y. By simply opening the SCOT output, we obtain a COTP protocol. However, their construction relies on DCF and therefore requires a large security parameter $\lambda = 128$. In contrast, we design a secret sharing-based COTP protocol purely based on secret sharing, achieving information-theoretic security. Unfortunately, under this stronger security requirement, we are only able to obtain a three-round protocol. Cf. Appendix. A for details.

4. Our New Paradigm for PPDT

The strawman protocol we proposed can evaluate the decision tree very quickly; however, there still exist some challenges. From one perspective, the strawman protocol requires the server $\mathcal{S}$ participants to perform the model shuffling phase. Considering the model requires providing service for multiple distinct users, $\mathcal{S}$ needs to stay online and perform intensive computations. From another perspective, $\mathcal{S}$ requires an external third party and cannot overlap with the computing parties in the $\mathcal{P}$. This is because if the $\mathcal{S}$ assumes the role of a computing party, the permutation becomes visible to it. As a result, the index information revealed during the online phase could allow the server to infer the evaluation path of the decision tree. In this section, we design a novel decision tree evaluation scheme that allows $\mathcal{S}$ to input the tree model in the setup phase and enables both $\mathcal{S}$ and $\mathcal{C}$ to participate as part of $\mathcal{P}$. In this section, we assume that $\mathcal{S}$ has already appended dummy nodes to the tree, and $\mathcal{C}$ has added dummy features to the feature list. **We abuse m and n to denote the total number of nodes and features, respectively, including the appended dummy entries.**

4.1. First Try: Secure Permutation

The most straightforward solution is to execute the model shuffling phase directly within the multi-party setting, i.e., by employing a secure permutation protocol rather than having the server $\mathcal{S}$ perform the shuffling locally. In this approach, the permutation is jointly generated in a distributed manner by $\mathcal{P}$, and remains hidden from all individual computing parties.

Naively, we let $\mathcal{P}$ generate the secret-shared random permutation $\langle\pi\rangle$ using $\Pi_{\mathsf{per-gen}}$, while the permutation π is unknown to any computing parties. Next, we need to replace the node or feature index, e.g., L_i, with the permuted index, e.g., $\pi(L_i)$. In this case, what we need is to use L_i to obliviously select $\pi(L_i)$ from the shared list $\langle\pi\rangle$. Informally, the brief procedure to generate the shuffled decision tree tuple is as follows.

- $\mathcal{S}$ secret shares the tree nodes, which are appended with the dummy nodes, to $\mathcal{P}$ with the start feature index α in the setup phase.
- $\mathcal{P}$ invoke $\langle\pi\rangle \leftarrow \Pi_{\mathsf{per-gen}}(m)$ and $\langle\pi'\rangle \leftarrow \Pi_{\mathsf{per-gen}}(n)$ to generate secret-shared random permutation lists for nodes and features.
- $\mathcal{P}$ invoke oblivious selection for $\langle L_i\rangle$, $\langle L'_i\rangle$, $\langle R_i\rangle$ and $\langle R'_i\rangle$ to select $\langle\pi(L_i)\rangle$, $\langle\pi'(L'_i)\rangle$, $\langle\pi(R_i)\rangle$ and $\langle\pi'(R'_i)\rangle$ of each shared node $\langle N_i\rangle := (\langle t_i\rangle, \langle L_i\rangle, \langle R_i\rangle, \langle L'_i\rangle, \langle R'_i\rangle)$.
- $\mathcal{P}$ invoke secure permutation for node list with π, namely,

$$(\langle\hat{N}_0\rangle, \ldots, \langle\hat{N}_{m-1}\rangle) \leftarrow \Pi_{\mathsf{per}}((\langle N_0\rangle, \ldots, \langle N_{m-1}\rangle), \langle\pi\rangle)$$

- $\mathcal{P}$ invoke oblivious selection for $\langle\alpha\rangle$ to select $\langle\mathcal{I}_1\rangle := \langle\pi'(\alpha)\rangle$. Then $\mathcal{P}$ open $\mathcal{I}_1$ and $\mathcal{I}_0 \leftarrow \pi(0)$.
- After $\mathcal{C}$ inputs $\mathcal{X}$, $\mathcal{P}$ invoke $\Pi_{\mathsf{per}}(\langle\mathcal{X}\rangle, \langle\pi'\rangle)$ to permute $\langle\mathcal{X}\rangle$.
- $\mathcal{P}$ follows the online evaluation procedure in Fig. 2 to complete the decision tree evaluation.

We describe how to generate a secret-shared random permutation, $\langle\pi\rangle \leftarrow \Pi_{\mathsf{per-gen}}(m)$. The process begins with one party secret-sharing a randomly chosen permutation. Then, the other two parties each apply their own independent random permutation to the shared values. This ensures that no single party learns the complete permutation. A simplified description is as follows:

1) P_0 generate random permutation $\pi_0 \in S_m \mapsto S_m$ and secret share π_0 to obtain $\langle\pi_0\rangle$;
2) P_1, P_2 generate same random permutation π_1 and apply π_1 on list $\langle\pi_0\rangle_0 + \langle\pi_0\rangle_2$ and $\langle\pi_0\rangle_1$ respectively; As a result, parties P_1 and P_2 hold the additively secret shares of the permuted values.
3) Applying the technique described in the secure shuffle, P_0 and P_1 convert the additive secret shares into replicated secret shares.

It can be seen that after $\mathcal{S}$ secret-shares the tree model, it does not need to participate in subsequent executions. Moreover, both $\mathcal{S}$ and $\mathcal{C}$ can be part of $\mathcal{P}$. However, the issue is that the aforementioned scheme requires the invocation of oblivious selection on each node, which incurs significant overhead. In the three-party setting, oblivious selecting among n elements of bit-length ℓ (i.e., from a domain of size 2^ℓ) requires at least $O(n\ell)$ communication or computational overhead. In the following sections, we aim to address the inefficiency caused by repeated oblivious selection calls.

4.2. The Solution: Our New PPDT Paradigm

In this section, we aim to eliminate the requirement for multiple oblivious selections. We first analyze how to handle node indices, and then apply a similar technique to feature indices.

Considering that m nodes have $2m$ child node indices, our intuition is whether we can directly place all permuted indices into the corresponding $2m$ positions, rather than

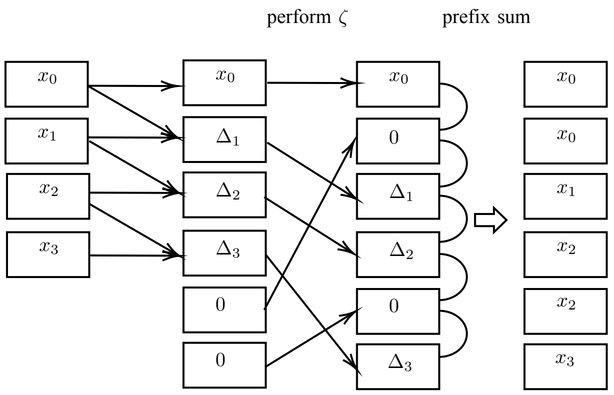


Figure 3: Oblivious Vector Expansion.

Oblivious Vector Expansion $\Pi_{\mathsf{ove}}(\mathsf{aux}, \mathcal{X})$

Input : $\mathsf{aux} := ((\mathcal{I}_0, a_0), \ldots, (\mathcal{I}_{n-1}, a_{n-1}))$ hold by $\mathcal{S}$, shared list $\langle x_0 \rangle, \ldots, \langle x_{n-1} \rangle$

Output : aligned secret-shared vector $\langle x'_0 \rangle, \ldots, \langle x'_{m-1} \rangle$.

Execution:

- calculate $\mathcal{X}' = (\langle x_0 \rangle, \langle x_1 \rangle - \langle x_0 \rangle, \langle x_2 \rangle - x_1, \ldots, \langle x_{n-1} \rangle - \langle x_{n-2} \rangle)$
- Let $\mathsf{point} = 0$, $\mathsf{end} = m$;
- For $i \in \mathbb{Z}_n$, $\mathcal{S}$ sets
  - $\pi(i) = \mathsf{point}$,
  - $\pi(\mathsf{end} - 1) = \mathsf{point} + 1, \ldots, \pi(\mathsf{end} - a_i) = \mathsf{point} + a_i$
  - $\mathsf{point} \leftarrow \mathsf{point} + a_i + 1$;
  - $\mathsf{end} \leftarrow \mathsf{end} - a_i$
- $\mathcal{S}$ secret-share π to $\mathcal{P}$
- $\mathcal{P}$ does:
  - append $m - n$ numbers of $\langle 0 \rangle$ to $\langle \mathcal{X} \rangle$.
  - perform $\Pi_{\mathsf{per}}(\langle \mathcal{X} \rangle, \langle \pi \rangle)$.
  - calculate $\langle x'_i \rangle \leftarrow \sum_{j=0}^{i} \langle x_j \rangle$.

Figure 4: Oblivious Expansion

being achieved through multiple selection. We observe that if each index (L_i or R_i) is unique and drawn from $\mathbb{Z}_{2m}$, then the transformation from $(L_0, R_0, \ldots, L_{m-1}, R_{m-1})$ to $(\pi(L_0), \pi(R_0), \ldots, \pi(L_{m-1}), \pi(R_{m-1}))$ can be viewed as a single permutation rather than multiple individual selections. For convenience, we denote $(L_0, R_0, \ldots, L_{m-1}, R_{m-1})$ as $(I_0, \ldots, I_{2m-1})$. In particular, there exists a permutation $\sigma : i \mapsto I_i$ which can perform on the permutation $\pi := (\pi(0), \ldots, \pi(2m-1))$ to obtain $(\pi(I_0), \ldots, \pi(I_{2m-1}))$. Checking its correctness, the i^{th} item of the permutation $\pi(i)$ will permute to I_i^{th} position, resulting $\pi(I_i)$.

Furthermore, we observe that in the above-described scheme, the list π does not need to be a permutation; it can be repeated. In light of this, we assign indices from 1 to $2m$ to all child node indices of the nodes, resulting in the index I_i. Then, for those that originally share the same index — for example, if I_i and I_j are equal — we enforce that $\pi(i) = \pi(j)$. We can just use the original $\pi \in S_m \mapsto S_m$ and duplicate $\pi(i)$ to obtain $\pi(j)$, finally resulting in $2m$-dimensional list, namely, we still use the π mentioned in 4.1 to permute the node list. However, we expand π to $\hat{\pi}$, by copying the items that refer to the same index, and use $\hat{\pi}$ to set the child node index for each node. Then, we invoke Π_{per} with σ to directly move $\hat{\pi}$ to the correct position corresponding to $(1, \ldots, 2m)$. Fig. 5a illustrates the corresponding process.

Note that with the above approach, the new index list $\hat{\pi}(I_i)$ can be only reconstructed using the permutations $\hat{\pi}$ and σ. The original index information L_i and R_i, which identify either the child node or its corresponding feature, is now embedded in the permutation σ, eliminating the need to store it explicitly.

Oblivious Vector Expansion. The remaining problem is how to expend π to $\hat{\pi}$ without revealing the model structure to computing parties $\mathcal{P}$. We design a protocol to solve that through a shuffling-based approach. Specifically, our starting point is the observation that the model holder can represent the repeated usage of certain nodes in the decision tree as a permutation structure. Formally, we define a replication pattern as follows,

$$((I_0, a_0), \ldots, (I_i, a_i), \ldots, (I_{m-1}, a_{m-1}))$$

where a_i denotes that I_i is repeated a_i times. Then, we define the concept of oblivious vector expansion

Definition 1. Let $((I_0, a_0), \ldots, (I_i, a_i), \ldots, (I_{m-1}, a_{m-1}))$ be a repetition pattern, $(\langle y_0 \rangle, \ldots, \langle y_n \rangle)$ be a secret-shared list. An Oblivious Vector Expansion scheme $\mathcal{F}_{\mathsf{ovd}}$ accepts input $((I_0, a_0), \ldots, (I_i, a_i), \ldots, (I_{m-1}, a_{m-1}))$ from one party and secret shared list $(\langle y_0 \rangle, \ldots, \langle y_n \rangle)$, after that $\mathcal{F}_{\mathsf{ovd}}$ output $(\langle y'_j \rangle)_{j \in \mathbb{Z}_L}$, $L = \sum_{i=0}^{m-1} a_i$, where $y'_k = y_i$ for $k \in [\sum_{j=0}^{i-1} a_j, \sum_{j=0}^{i} a_j)$.

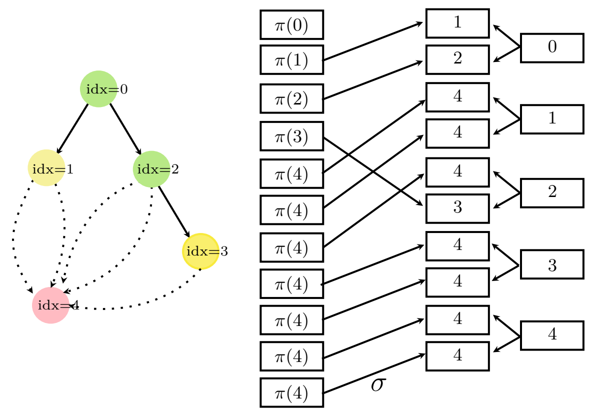
Fig. 3 depicts the procedure of our protocol to copy some elements in a vector. Given a toy example, for a shorter list $(x_0, \ldots, x_3)$, to repeat x_0 and x_2 twice, we first compute the difference sequence of the list $(x_0, \ldots, x_3)$, namely,

$$\mathcal{L}_0 := (x_0, \Delta_1 := x_1 - x_0, \Delta_2 := x_2 - x_1, \Delta_3 := x_3 - x_2)$$

Then we append several zeros at the end corresponding to the number of items to be copied, namely, $\mathcal{L}_1 := (x_0, \Delta_1, \Delta_2, \Delta_3, 0, 0)$. After that, we apply a permutation ζ to $\mathcal{L}_1$, which moves the appended zeros to the positions immediately following the items that need to be replicated, obtaining the list $\mathcal{L}_2 := (x_0, 0, \Delta_1, \Delta_2, 0, \Delta_3)$. Finally, by computing the prefix sum of $\mathcal{L}_2$, we obtain the expanded list, which is $\mathcal{L}_3 := (x_0, x_0, x_1, x_2, x_2, x_3)$.

We define $\mathsf{aux} := ((I_0, a_0), \ldots, (I_{m-1}, a_{m-1}))$ as the repetition pattern for the nodes list held by $\mathcal{S}$. Similarly, we define aux' to represent the repetition pattern for feature indices. Our goal is to extend the randomly secret-shared permutation $\pi \in S_n$ by using the auxiliary input aux provided by $\mathcal{S}$, to obtain the new shared list $\hat{\pi}$ where repeat $\pi(I_i)$ for a_i times. Using the aforementioned scheme, we generate ζ using aux as follows, where it moves the tail zero to the repetition position.

- $\mathsf{aux} := ((\mathcal{I}_0, a_0), \ldots, (\mathcal{I}_{n-1}, a_{n-1}))$
- Let $\mathsf{point} = 0$, $\mathsf{end} = m$;


(a) Permutation for nodes

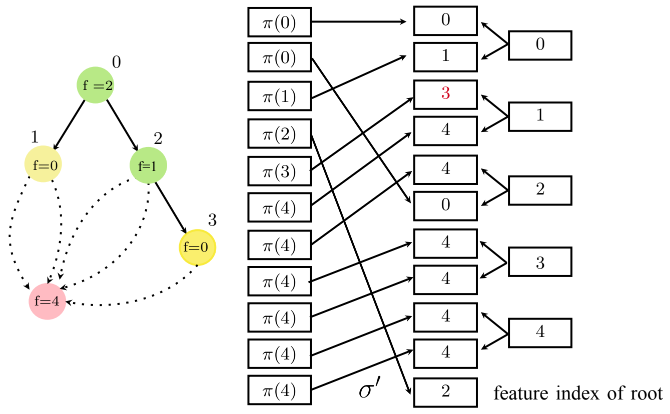

(b) Permutation for features

Figure 5: A Case of Permutation σ

- For $i \in \mathbb{Z}_m$, $\mathcal{S}$ sets the permutation $\zeta(i)$ for ζ as follows
  - $\zeta(i) = \mathsf{point}$,
  - $\zeta(\mathsf{end}-1) = \mathsf{point}+1, \ldots, \zeta(\mathsf{end}-a_i) = \mathsf{point}+a_i$
  - $\mathsf{point} \leftarrow \mathsf{point} + a_i + 1$;
  - $\mathsf{end} \leftarrow \mathsf{end} - a_i$

By applying the aforementioned expansion procedure with permutation ζ on $\pi \in S_n$, we can obtain $\hat{\pi}$. The overall oblivious vector expansion protocol is depicted in Fig. 4.

New Representation of Decision Tree. So far, we have obtained an extended list $\hat{\pi}$. $\mathcal{S}$ knows the decision tree, and it can use it to generate the permutation σ to move $\hat{\pi}$ to the correct position on the tree. Fig. 5a depicts how to construct the permutation σ for the nodes permutation $\hat{\pi}$. The left-hand side is a simple case of a 4-node decision tree with a dummy node. The right-hand side is how the permutation σ moves π to the correct position. The index $\pi(0)$, which corresponds to the root node, is directly revealed to determine the entry point of the evaluation. With the exception of $\pi(0)$, each of the remaining elements of π can be uniquely associated with the left or right child index of node N_i. This mapping allows $\mathcal{S}$ to deterministically construct a unique permutation σ. Thus, the node information of the decision tree has been encoded into two permutations, ζ and σ. Accordingly, we express the model information required by $\mathcal{S}$ as $\mathcal{T} := ((t_i)_{i\in\mathbb{Z}_m}, \zeta, \sigma, \zeta', \sigma')$, where ζ and σ encode the node-related permutations of the decision tree: ζ replicates indices with respect to their usage frequency, and σ places them into the correct positions via a random permutation. The permutations ζ' and σ' serve a similar purpose for the feature indices.

Unlike nodes, where the root node is conventionally placed at index 0 and $\mathcal{P}$ can directly reveal $\pi(0)$ to obtain the starting node index, the initial feature index is determined by the root node. Consequently, a permutation σ is applied to map the corresponding $\pi'(\alpha)$ to a predetermined position. In the online phase, $\mathcal{P}$ can then reveal the corresponding entry in π' at that fixed position to retrieve the required feature index. Fig. 5b illustrates a simple example of this process. We construct σ to uniformly move $\pi(\alpha)$ to the last position $2m+1$ ($\alpha = 2$ in Fig. 5b), rather than filling it into the child index of the decision tree.

Concrete Construction. Fig. 6 illustrates the overall process of our decision tree evaluation. At the setup phase, the server $\mathcal{S}$ secret-shares the decision tree $(t, \zeta, \sigma, \zeta', \sigma')$ to the parties $\mathcal{P}$. In the model shuffle phase, $\mathcal{P}$ uses the secret-shared values $(t, \zeta, \sigma, \zeta', \sigma')$ to generate a one-time-use permuted decision tree tuple. During the online phase, the client $\mathcal{C}$ inputs the feature vector, and P evaluates the tree layer by layer to obtain the output. We will introduce the protocol step by step below.

- In the setup phase, $\mathcal{S}$ share the tree model $\mathcal{T} := ((t_i)_{i\in\mathbb{Z}_m}, \zeta, \sigma, \zeta', \sigma')$ to $\mathcal{P}$;
- In the model shuffle phase, steps 1-5, $\mathcal{P}$ generates random permutations π and π' for tree nodes and features, and expands the permutation to $2m$ dimensions.
- In the model shuffle phase, steps 6-8, $\mathcal{P}$ moves π and $\hat{\pi}$ to the corresponding nodes, and shuffles the tree nodes.
- In the model shuffle phase step 9, $\mathcal{P}$ open the root node index and feature index.
- In the online phase steps 1-2, $\mathcal{C}$ secret shares the feature $\mathcal{X}$ to $\mathcal{P}$, and $\mathcal{P}$ invokes Π_{per} to shuffle $\mathcal{X}$ with $\hat{\pi}$ generated in the model shuffle phase.
- In the online phase steps 3-5, $\mathcal{P}$ evaluates the decision tree layer by layer to obtain the label R.

Remark: Since we consider the PPDT-as-a-service setting, model shuffling can be completed by $\mathcal{P}$ in the offline phase.

4.3. Repeated Feature on the Same Branch

Most decision trees do not reuse the same feature index along a single path, and the above scheme can effectively handle such cases. However, if multiple occurrences of the same feature index appear on the same branch, applying the above scheme would reveal the index by opening it multiple times, leading to potential information leakage.

This issue can be easily addressed using our expansion protocol. Specifically, for repeated feature indices, we can

```
Decision Tree scheme $\Pi_{\mathsf{tree}}(\mathcal{X}, \mathcal{T})$

Input : Client $\mathcal{C}$ holds feature list $\mathcal{X} := (x_0, x_1, \ldots, x_{n-1})$; Server $\mathcal{S}$ holds decision tree $\mathcal{T} := (\zeta, \sigma, \zeta', \sigma', (t_0, \ldots, t_{m-1}))$.
Output : model evaluation result $R$.

Set-up:
- $\mathcal{S}$ secret share $\zeta$, $\sigma$, $\zeta'$, $\sigma'$ and the threshold list $(t_0, \ldots, t_{m-1})$ to $\mathcal{P}$;

Model Shuffle:
1) $\mathcal{P}$ generate random permutation $\langle\pi\rangle \leftarrow S_m \mapsto S_m$ and $\langle\pi'\rangle \leftarrow S_n \mapsto S_n$;
2) $\mathcal{P}$ set $\langle\delta\rangle \leftarrow (\langle\pi(1)\rangle, \langle\pi(2)\rangle - \langle\pi(1)\rangle, \ldots, \langle\pi(m-1)\rangle - \langle\pi(m-2)\rangle)$ and
   $\langle\delta'\rangle \leftarrow (\langle\pi'(0)\rangle, \langle\pi'(1)\rangle - \langle\pi'(2)\rangle, \ldots, \langle\pi'(n-1)\rangle - \langle\pi'(n-2)\rangle)$.
3) $\mathcal{P}$ append $m+1$ items of zero to $\langle\delta\rangle$ and $2m-n+1$ items of zero to $\langle\delta'\rangle$
4) $\mathcal{P}$ invoke $\langle\delta\rangle \leftarrow \Pi_{\mathsf{per}}(\langle\delta\rangle, \zeta)$ and $\langle\delta'\rangle \leftarrow \Pi_{\mathsf{per}}(\langle\delta'\rangle, \langle\zeta'\rangle)$
5) $\mathcal{P}$ calculate the prefix sum $\langle\delta_i\rangle \leftarrow \sum_{j=0}^{i}\langle\delta_j\rangle$ and $\langle\delta'_i\rangle \leftarrow \sum_{j=0}^{i}\langle{\delta'}_j\rangle$
6) $\mathcal{P}$ invoke $\langle\hat{\pi}\rangle \leftarrow \Pi_{\mathsf{per}}(\langle\delta\rangle, \langle\sigma\rangle)$ and $\langle\hat{\pi}'\rangle \leftarrow \Pi_{\mathsf{per}}(\langle\delta'\rangle, \langle\sigma'\rangle)$
7) $\mathcal{P}$ set $\langle N_i\rangle := (\langle t_i\rangle, \langle\delta_{2i}\rangle, \langle\delta_{2i+1}\rangle, \langle\hat{\delta}_{2i}\rangle), \langle\hat{\delta}_{2i+1}\rangle)$ for $i \in \mathbb{Z}_m$
8) $\mathcal{P}$ invoke $(\langle N_0\rangle, \ldots, \langle N_{m-1}\rangle) \leftarrow \Pi_{\mathsf{per}}((\langle N_0\rangle, \ldots, \langle N_{m-1}\rangle), \langle\pi\rangle)$;
9) $\mathcal{P}$ open $\langle\mathcal{I}_0\rangle := \langle\pi_0\rangle$ and $\langle\mathcal{I}_1\rangle := \langle{\pi'}_{2m+1}\rangle$ to $\mathcal{P}$

Model Evaluation:
1) $\mathcal{C}$ secret-share $\mathcal{X}$ to $\mathcal{P}$;
2) $\mathcal{P}$ invoke $\langle\mathcal{X}\rangle \leftarrow \Pi_{\mathsf{per}}(\langle\mathcal{X}\rangle, \langle\pi'\rangle)$;
3) $\mathcal{P}$ sets $\langle R\rangle = \langle 0\rangle$
4) For $i \in \mathbb{Z}_d$, $\mathcal{P}$ do
   • set $\langle N_o\rangle \leftarrow \langle N_{\mathcal{I}_0}\rangle$; $\langle x_o\rangle \leftarrow \langle x_{\mathcal{I}_1}\rangle$;
   • parse $\langle N_o\rangle$ as tuple $\langle N_o\rangle := (\langle t_o\rangle, \langle L_o\rangle, \langle R_o\rangle, \langle L'_o\rangle, \langle R'_o\rangle)$
   • update $\langle R\rangle \leftarrow \langle R\rangle + \langle t_o\rangle$
   • invoke shared conditional oblivious transfer $(\mathcal{I}_0, \mathcal{I}_1) \leftarrow \Pi_{\mathsf{cotp}}(\langle x_o\rangle, \langle t_o\rangle, \langle L_o\rangle, \langle R_o\rangle, \langle L'_o\rangle, \langle R'_o\rangle)$
5) $\mathcal{P}$ calculate $\langle R\rangle \leftarrow \langle R\rangle + \langle t_{\mathcal{I}_0}\rangle$.
```

Figure 6: Our New Privacy-Preserving Decision Tree Evaluation Paradigm.

use the oblivious vector expansion protocol to create a copy and assign a new index to it. Then we let $\mathcal{S}$ modify the duplicated feature indices in the tree model to point to the replicated index. Note that if a feature appears multiple times across different branches, we take the maximum number of repetitions from each branch as the number of required copies. In particular, $\mathcal{S}$ inputs the list (I_i, a_i) and generate π for duplicate I_i^{th} feature, and $\mathcal{C}$ inputs features $\mathcal{X}$. Simultaneously, $\mathcal{S}$ uses the expanded feature index to generate the decision tree.

5. Cost and Security Analysis.

In this section, we systematically analyze the cost and security of each component as well as the overall decision tree protocol.

5.1. Communication Cost

Conditional Oblivious Transfer with Public Opening. Since our decision tree evaluation in the online phase is conducted by invoking COTP at each layer to reveal the output, COTP constitutes the primary overhead in our protocol. We could directly adopt the protocol by Ji *et al.* [17], who realized a one-round shared conditional OT under replicated sharing; adding one extra opening round also yields a two-round protocol. Their method incurs higher computation and offline communication than the COTP we proposed in Appendix. A, but achieves lower online communication. Using their construction, the online phase requires two rounds with $12\ell + 9(\log n + \log m)$ bits of communication, while the offline phase incurs $6\lambda\ell$ bits. If we adopt our proposed COTP protocol (which achieves information-theoretic security), the online phase requires 3 rounds with a communication cost of $14(\log m + \log n) + 4\ell\log\ell$ bits, and the offline phase requires $20\log n + 20\log m + 2\ell\log\ell$ bits.

Our Strawman Protocol. In our Strawman protocol, the offline phase only requires secret-sharing the decision tree and the corresponding input indices. Each node has a size of $\ell + 2(\log m + \log n)$ bits, and for m nodes, the communication cost of secret-sharing is $2m(\ell + 2(\log m + \log n))$ bits. Considering the COTP, the total communication cost in the offline phase of our Strawman protocol is $2m(\ell + 2(\log m + \log n)) + 6\lambda\ell$ bits for the DCF-based COTP and $2m(\ell + 2(\log m + \log n)) + 20\log n + 20\log m + 2\ell\log\ell$ bits for the SS-based COTP. In the online phase, the client needs to secret-share the feature vector, resulting in a communication cost of $2n\ell$ bits. Additionally, for DCF-based COTP, executing $d-1$ instances of the COTP protocol incurs a cost of $2d-2$ rounds and a total communication overhead of $(d-1)(12\ell + 9(\log n + \log m))$ bits, such that the total cost of the online phase is $2d-1$ rounds of $(d-1)(12\ell + 9(\log n + \log m)) + 2n\ell$ bits. For SS-based COTP, executing $d-1$ instances of the COTP protocol incurs a cost of $3d-3$ rounds and a total communication overhead of $(d-1)(14(\log m+\log n)+4\ell\log\ell)$ bits, such that the total cost of the online phase is $3d-2$ rounds of $(d-1)(14(\log m+\log n)+4\ell\log\ell)+2n\ell$ bits.

Our New PPDT Paradigm. Compared to our strawman protocol, the proposed PPDT architecture requires additional permutation operations. In the offline phase, our protocol contains the following communication cost

- Generation of random permutation lists: For n and m length shared random permutation $\langle\pi\rangle$ and $\langle\pi'\rangle$, each element in the permutations requires $\log n$ and $\log m$ bits to represent. As a result, the total communication cost for generating these permutations is $6(m\log m+n\log n)$ bits ($6n\log n$ for shuffle list $(1,\ldots,n)$).
- Secure Permutation: For performing ζ, ζ', σ and σ' on $2m$ feature indices and $2m$ node indices, since the feature indices are in $\mathbb{Z}_n$ and require $\log n$ bits, and the node indices require $\log m$ bits, the total cost of permuting n elements with ℓ-bit values is approximately $15n\ell$. Therefore, the total communication overhead introduced by these permutations is $60m(\log n+\log m)$ bits.
- Node permutation: the permutation of the nodes N_i incurs an additional cost of $15m(\ell+2\log m+2\log n)$ bits; the cost of revealing the entry index is $3(\log n+\log m)$ bits.

Therefore, the total communication cost of our offline phase is $66m\log m+6n\log n+60m\log n+3(\log n+\log m)+6\lambda\ell$ bits for DCF-based COTP and $66m\log m+6n\log n+60m\log n+3(\log n+\log m)+20\log n+20\log m+2\ell\log\ell$ bits for SS-based COTP. Considering the online phase, compared to the strawman protocol, $\Pi_{\mathsf{tree}}(\mathcal{X},\mathcal{T})$ requires one additional permutation on the features, which incurs an extra cost of 4 rounds with $15n\log n$ bits.

5.2. Security Analysis

Our PPDT Scheme. We can easily analyze the security of our PPDT scheme. By examining the messages received by any P_i, we can see that no information about the model or the inputs can be learned. First, the values obtained through secret sharing are uniformly random. Apart from the secret shares, P_i only receives the opened values $\mathcal{I}_0$ and $\mathcal{I}_1$ at each layer. Due to the random permutations, after d layers, the sequences $\mathcal{I}_0$ and $\mathcal{I}_1$ are indistinguishable from selecting d random elements from the lists $(0,\ldots,m)$ and $(0,\ldots,n)$. This demonstrates the security of our decision tree evaluation scheme.

6. Implementation and Benchmark.

In this section, we benchmark our PPDT. We compare our work against the schemes proposed by Ji [17] *et al.* and Ma [25] *et al.*

Benchmark setting. We implement our protocols in C++. For the $\mathcal{F}_{\mathsf{cmp}}$, we utilize the code provided by Lu [22] *et al* and Ji [17] *et al.* For our experiments, the benchmark is performed in a server that runs Ubuntu 18.04.2 LTS with Intel(R) Xeon(R) Silver 4214 CPU @ 2.20GHz, 48 CPUs, 128 GB Memory. Our experiments are performed in a local area network, using traffic control in Linux to simulate three network settings: (1) LAN: 10Gbps bandwidth with 0.01ms round-trip latency (RTT). (2) WAN: 100Mbps bandwidth with 6ms round-trip. We use a ring of size 32 bits for representing decision tree thresholds and labels. We use UCI datasets [12] to construct benchmarks for various tree models. In Table. 3, n, d, and m denote the number of features, tree depth, and number of nodes, respectively.

Communication Cost Comparison. We compare the communication overhead for a single decision tree evaluation with the schemes of Ji [17] *et al.* and Ma [25] *et al.*. Detailed results (in bytes) are shown in Table 3. Ours-1 refers to our strawman scheme, while Ours-2 corresponds to our Π_{tree} scheme, and Ours-3 corresponds to replacing COT on Π_{tree} with DCF. In Ma *et al.*'s work, the online communication overhead is significantly higher than that of Ji *et al.* and ours. Compared to Ji *et al.*'s scheme, our approach reduces the online communication cost by nearly 45% in scenarios where the decision tree is relatively sparse. Through further analysis, we observe that the sparser the tree, the more pronounced the advantage of our scheme becomes. We can further reduce the online phase cost by adopting a DCF-based COT construction. However, this also introduces a significant increase in offline overhead. The performance of Ours-3 reflects this trade-off: while the online communication is substantially reduced, it comes at the cost of heavier offline preprocessing. Specifically, on the Boston dataset, our protocol—when using DCF-based COT—achieves an online communication cost that is only 14% of that in Ji *et al.*'s scheme. It is also worth noting that the oblivious selection of Ji *et al.*'s protocol is also based on Function Secret Sharing (FSS), which requires a large number of PRF evaluations for selection and comparison. In contrast, our scheme performs selection via simple memory access operations, making it substantially more lightweight than FSS-based approaches. In addition, Ji et al.'s protocol incurs substantial offline communication due to the heavy use of Function Secret Sharing (FSS) during preprocessing. For example, on the Breast dataset, Ji et al. require 53,729 bytes in the offline phase, while our scheme only consumes 1,528 bytes, respectively—a dramatic reduction.

Running Time Comparison. Fig. 7 shows the online phase performance of our scheme across different datasets and network settings. Here, Ours corresponds to our Π_{tree} scheme, while Ours(DCF) refers to a variant where we replace our COT with the COT based on Ji *et al.*'s DCF construction. We observe that in poor network conditions, the overall running time is dominated by communication overhead. In such cases, although Ji et al.'s COT based on DCF incurs a higher computational cost during the online phase, its communication cost is only 4ℓ, which is lower than our COT's $4\ell\log\ell$. This makes it more suitable for bandwidth-constrained scenarios. Therefore, we also evaluated a variant of our protocol that adopts a DCF-based

TABLE 3: Comparison of online and offline costs (Bytes) ; Ours-1 refers to our strawman scheme; Ours-2 corresponds to our Π_{tree} scheme; Ours-3 corresponds to replacing COT on Π_{tree} with DCF; We set $\ell = 32$.

Dataset (n, d, m)	Online					Offline				
	Ma *et al.*	Ji *et al.*	Ours-1	Ours-2	Ours-3	Ma *et al.*	Ji *et al.*	Ours-1	Ours-2	Ours-3
Iris(4,4,7)	15172	470	250	265	57	152	28268	67	330	8522
Wine(7,5,11)	19025	621	338	375	85	244	36736	112	650	10890
Linnerud(3,6,19)	22734	733	421	430	96	426	43583	162	1049	13337
Breast(12,7,21)	26775	923	515	596	126	484	53729	218	1528	15864
Digits(47,15,168)	59475	2313	1256	1745	358	4421	129447	1653	18789	49509
Spambase(57,17,58)	68085	2526	1420	2043	416	1572	142607	801	6058	40874
Diabetes(10,28,393)	106876	4192	2401	2464	602	12149	236223	3208	40907	98251
Boston(13,30,425)	114870	4558	2590	2680	656	13477	255926	3485	45850	107290

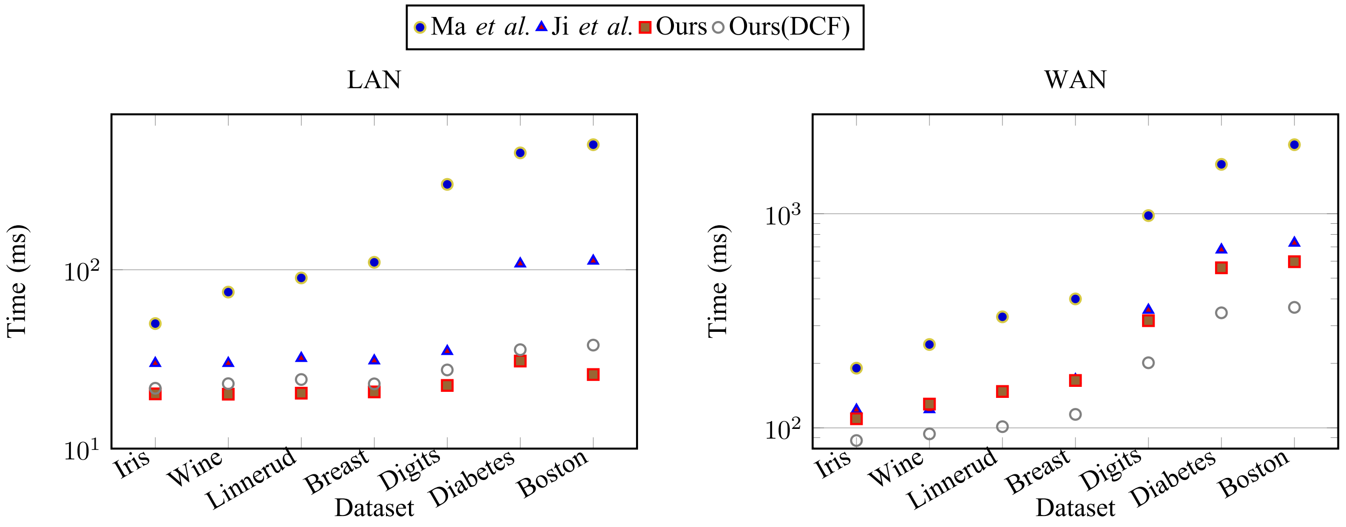


Figure 7: Online Running Time Comparison of Ma [25] *et al.*, Ji [17] *et al.*, and Our Method under LAN and WAN Settings

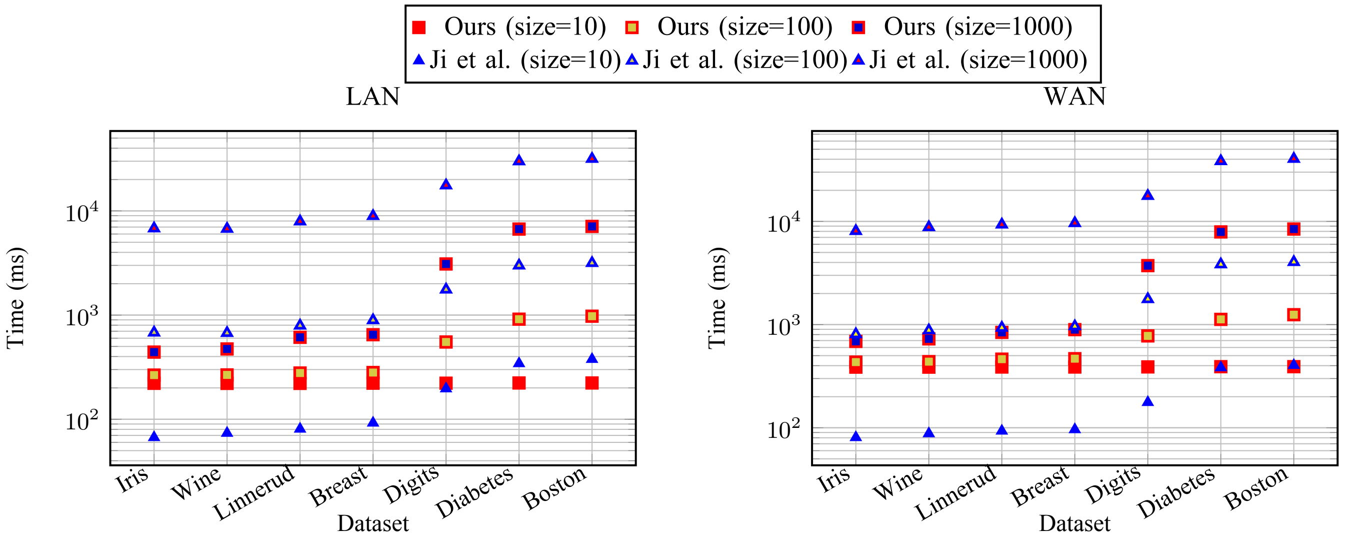
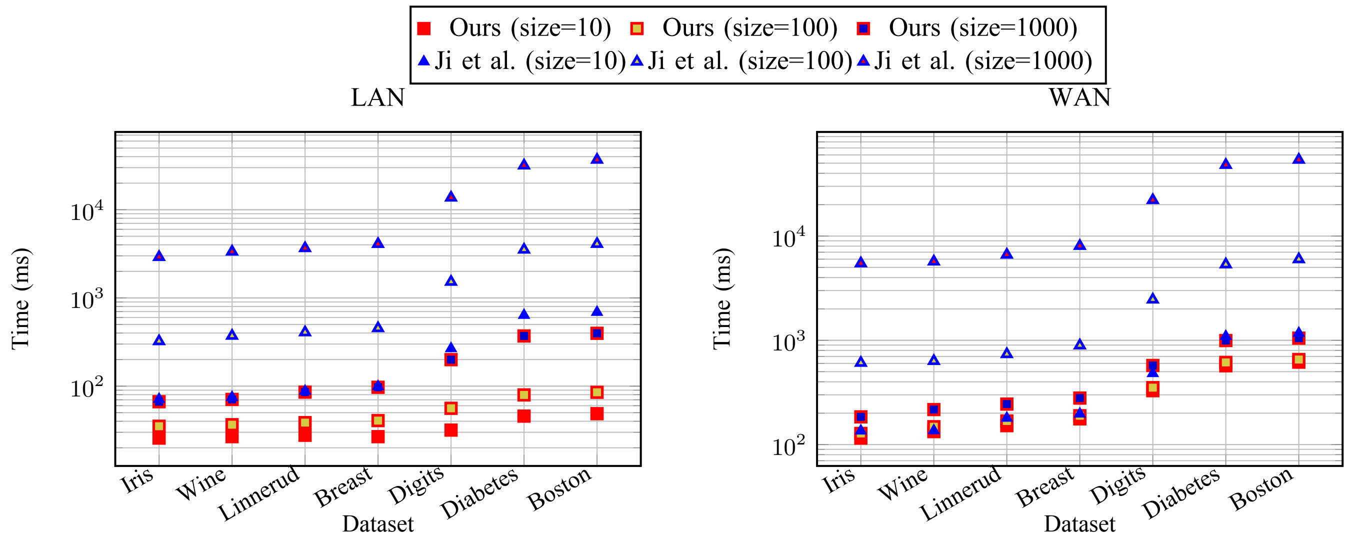


Figure 8: Offline Running Time Comparison under LAN and WAN for different batch sizes.



(a) Online time under LAN

(b) Online time under WAN

Figure 9: Online Running Time Comparison under LAN and WAN for different batch sizes.

COT for comparison. Under the LAN setting, our scheme maintains online latency between 15 ms and 40 ms across all datasets, which is significantly lower than Ji et al. and Ma et al., whose latency ranges from 40 ms to over 300 ms. The performance gap becomes especially apparent as dataset complexity increases. On the Boston dataset, our scheme is nearly 20$\times$ faster than that of Ma *et al.*, and approximately 4.5$\times$ faster than Ji *et al.*'s scheme. In the WAN setting, although network conditions introduce additional delay, our method remains substantially more efficient. For complex datasets such as Diabetes and Boston, our method is approximately 3–5$\times$ faster than Ma *et al.*, and 1.5–2$\times$ faster than Ji *et al.*. Considering batched inputs, the advantages of our protocol become even more pronounced. As shown in Fig. 9, we evaluate the performance of our protocol under different batch sizes. When the batch size increases, our protocol outperforms Ji *et al.*'s scheme, achieving up to a 50$\times$ speedup in the best case.

Offline Running Time Comparison. Fig. 8 illustrates the offline running time comparison between our proposed method and that of Ji et al. under both LAN and WAN settings, evaluated across varying batch sizes (10, 100, and 1000). For small models and small batch sizes, our scheme incurs slightly higher offline overhead compared to Ji *et al.*'s scheme, possibly due to factors such as the number of communication rounds. However, as the batch size increases and the model becomes larger, communication volume becomes the dominant factor affecting runtime. In such cases, our scheme significantly outperforms Ji *et al.*'s in the offline phase. For example, with a batch size of 1000 on the Boston dataset, our scheme is 5$\times$ faster than Ji *et al.*'s. For the Breast dataset with a batch size of 1000, our protocol achieves a 10.7$\times$ speedup over Ji *et al.*'s. This highlights the efficiency of our offline design, especially in high-throughput scenarios where preprocessing costs become a bottleneck.

Running Time Comparison under Different Batch Sizes. Fig. 9 illustrates the performance of our protocol under varying batch sizes. Compared to Ji et al.'s protocol, our scheme demonstrates significantly better scalability. As the batch size increases, our runtime grows at a much slower rate, while Ji et al.'s runtime exhibits an almost linear increase with respect to the batch size. This indicates that our protocol is better optimized for handling large batches of inputs, making it more suitable for practical deployment in high-throughput settings. For instance, under the Boston dataset with a batch size of 1000, our protocol achieves over 50$\times$ speedup over Ji et al.'s protocol in the online phase. This performance gain becomes more pronounced as the model and input size grow, showcasing the advantage of our design in amortizing computation and communication costs across large input batches. This trend can also be observed across other datasets, where our protocol maintains a slow growth in runtime as the batch size increases from 10 to 1000. In contrast, Ji et al.'s protocol suffers from substantial increases in both communication and computational overhead, particularly in wide-area network (WAN) environments. These results validate the superior scalability and practicality of our method for privacy-preserving inference at scale.

7. Conclusion

In this work, we eliminate the need for oblivious selection in privacy-preserving decision tree evaluation, resulting in significant performance improvements. Our scheme reduces online communication by 86% compared to the state-of-the-art (SOTA), achieves a $4.5 \sim 20\times$ speedup in the online phase, and delivers over 5$\times$ improvement in offline performance.

References

[1] Toshinori Araki, Jun Furukawa, Yehuda Lindell, Ariel Nof, and Kazuma 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*, CCS '16, page 805–817, New York, NY, USA, 2016. Association for Computing Machinery.

[2] Gilad Asharov, Koki Hamada, Dai Ikarashi, Ryo Kikuchi, Ariel Nof, Benny Pinkas, Katsumi Takahashi, and Junichi Tomida. Efficient secure three-party sorting with applications to data analysis and heavy hitters. CCS, 2022.

[3] Jianli Bai, Xiangfu Song, Shujie Cui, Ee-Chien Chang, and Giovanni Russello. Scalable private decision tree evaluation with sublinear communication. In *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*, pages 843–857, 2022.

[4] Jianli Bai, Xiangfu Song, Xiaowu Zhang, Qifan Wang, Shujie Cui, Ee-Chien Chang, and Giovanni Russello. Mostree: Malicious secure private decision tree evaluation with sublinear communication. In *Proceedings of the 39th Annual Computer Security Applications Conference*, pages 799–813, 2023.

[5] Donald Beaver. Efficient multiparty protocols using circuit randomization. In *CRYPTO*, 1991.

[6] Raphael Bost, Raluca Ada Popa, Stephen Tu, and Shafi Goldwasser. Machine learning classification over encrypted data. 2015.

[7] Justin Brickell, Donald E. Porter, Vitaly Shmatikov, and Emmett Witchel. Privacy-preserving remote diagnostics. pages 498–507, 2007.

[8] Nan Cheng, Naman Gupta, Aikaterini Mitrokotsa, Hiraku Morita, and Kazunari Tozawa. Constant-round private decision tree evaluation for secret shared data. PoPETs, 2023.

[9] Nan Cheng, Naman Gupta, Aikaterini Mitrokotsa, Hiraku Morita, and Kazunari Tozawa. Constant-round private decision tree evaluation for secret shared data. *Proceedings on Privacy Enhancing Technologies*, 2024.

[10] Martine De Cock, Rafael Dowsley, Caleb Horst, Raj Katti, Anderson C. A. Nascimento, Wing-Sea Poon, and Stacey Truex. Efficient and private scoring of decision trees, support vector machines and logistic regression models based on pre-computation. *IEEE Trans. Dependable Secur. Comput.*, 16(2):217–230, February 2017.

[11] Jack Doerner and Abhi Shelat. Scaling oram for secure computation. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 523–535, 2017.

[12] D. Dua and C. Graff. UCI machine learning repository. http://archive.ics.uci.edu/ml, 2017.

[13] Jiaxuan Fu, Ke Cheng, Yuheng Xia, Anxiao Song, Qianxing Li, and Yulong Shen. Private decision tree evaluation with malicious security via function secret sharing. In *European Symposium on Research in Computer Security*, pages 310–330. Springer, 2024.

[14] Jiaxuan Fu, Ke Cheng, Yuheng Xia, Anxiao Song, Qianxing Li, and Yulong Shen. Private decision tree evaluation with malicious security via function secret sharing. In Joaquin Garcia-Alfaro, Rafał Kozik, Michał Choraś, and Sokratis Katsikas, editors, *Computer Security – ESORICS 2024*, pages 310–330, Cham, 2024. Springer Nature Switzerland.

[15] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *STOC*, 1987.

[16] Yuval Ishai and Anat Paskin. Evaluating branching programs on encrypted data. pages 575–594, 2007.

[17] Keyu Ji, Bingsheng Zhang, Tianpei Lu, Lichun Li, and Kui Ren. Uc secure private branching program and decision tree evaluation. *IEEE Transactions on Dependable and Secure Computing*, 20(4):2836–2848, 2023.

[18] Marc Joye and Fariborz Salehi. Private yet efficient decision tree evaluation. In *DBSec 2018*, volume 10980, pages 243–259, Bergamo, Italy, July 16–18 2018.

[19] Ágnes Kiss, Masoud Naderpour, Jian Liu, N. Asokan, and Thomas Schneider. SoK: Modular and efficient private decision tree evaluation. 2019(2):187–208, April 2019.

[20] Lin Liu, Jinshu Su, Rongmao Chen, Jinrong Chen, Guangliang Sun, and Jie Li. Secure and fast decision tree evaluation on outsourced cloud data. In *ML4CS 2019*, pages 361–377, Xi'an, China, September 19–21, 2019.

[21] Tianpei Lu, Xin Kang, Bingsheng Zhang, Zhuo Ma, Xiaoyuan Zhang, Yang Liu, Kui Ren, and Chun Chen. Efficient 2PC for constant round secure equality testing and comparison. USENIX Security, 2025.

[22] Tianpei Lu, Bingsheng Zhang, Lichun Li, Yuzhou Zhao, and Kui Ren. Lightning fast secure comparison for 3PC PPML. Cryptology ePrint Archive, Paper 2023/1890, 2023.

[23] Tianpei Lu, Bingsheng Zhang, Xiaoyuan Zhang, and Kui Ren. A new PPML paradigm for quantized models, 2025.

[24] Tianpei Lu, Qizhi Zhang, BingSheng Zhang, Lichun Li, and Shan Yin. Secure computation for g-module and its applications. *ICCCN*, 2025.

[25] Jack P. K. Ma, Raymond K. H. Tai, Yongjun Zhao, and Sherman S.M. Chow. Let's stride blindfolded in a forest: Sublinear multi-client decision trees evaluation. 2021.

[26] Raymond K. H. Tai, Jack P. K. Ma, Yongjun Zhao, and Sherman S. M. Chow. Privacy-preserving decision trees evaluation via linear functions. pages 494–512, 2017.

[27] Anselme Tueno, Florian Kerschbaum, and Stefan Katzenbeisser. Private evaluation of decision trees using sublinear cost. 2019(1):266–286, January 2019.

[28] Cong Wang, Kui Ren, Jia Wang, and Qian Wang. Harnessing the cloud for securely outsourcing large-scale systems of linear equations. *IEEE Transactions on Parallel and Distributed Systems*, 24(6):1172–1181, 2013.

[29] David J. Wu, Tony Feng, Michael Naehrig, and Kristin E. Lauter. Privately evaluating decision trees and random forests. 2016(4):335–355, October 2016.

[30] Andrew C. Yao. Protocols for secure computations. In *SFCS*, 1982.

[31] Yifeng Zheng, Huayi Duan, and Cong Wang. Towards secure and efficient outsourcing of machine learning classification. pages 22–40, 2019.

[32] Lijing Zhou, Ziyu Wang, Hongrui Cui, Qingrui Song, and Yu Yu. Bicoptor: Two-round secure three-party non-linear computation without preprocessing for privacy-preserving machine learning. In *S&P 2023*, pages 1295–1312, 2023.

Functionality $\mathcal{F}_{\mathsf{cmp}}[i]$

In the offline phase,

- P_{i-1} and P_i send $(\mathsf{input}, \mathsf{sid}, \langle x\rangle_{i+1}, \langle y\rangle_{i+1})$ to $\mathcal{F}_{\mathsf{cmp}}$;
- P_{i+1} and P_i send $(\mathsf{input}, \mathsf{sid}, \langle x\rangle_{i-1}, \langle y\rangle_{i-1})$ to $\mathcal{F}_{\mathsf{cmp}}$;
- $\mathcal{F}_{\mathsf{cmp}}$ picks random $z_1 \leftarrow \{0,1\}$ and sends z_1 to P_{i-1} and P_{i+1};

In the online phase,

- P_{i+1} and P_{i-1} send $(\mathsf{input}, \mathsf{sid}, \langle x\rangle_i, \langle y\rangle_i)$ to $\mathcal{F}_{\mathsf{cmp}}$;
- $\mathcal{F}_{\mathsf{cmp}}$ calculates $z = (\sum_j^2 = 0\langle x\rangle_j) \geq (\sum_j^2 = 0\langle y\rangle_j)$, picks random $z_0 \leftarrow \{0,1\}$
- $\mathcal{F}_{\mathsf{cmp}}$ calculates $z_0 = z \oplus z_0$ and sends z_0 to P_i.

Figure 10: Functionality of Comparison.

```
Protocol $\Pi_{\mathsf{cmp}}(i, \langle x \rangle, \langle y \rangle)$

$P_j$ and $P_k$ hold the common seed $\eta_{j,k} \in \{0,1\}^{\lambda}$.
Input : $\langle\cdot\rangle$-shared value of $x \in \mathbb{Z}_{2^{\ell-1}}$ and $y \in \mathbb{Z}_{2^{\ell-1}}$.
Output : $P_i$ gets $z_0$, $P_{i-1}$ and $P_{i+1}$ get $z_1$, while $z = z_0 + z_1 = x \overset{?}{>} y$.
Preprocessing:
- $P_i$ does:
    1) set $r = \langle x \rangle_{i-1} + \langle x \rangle_{i+1} - \langle y \rangle_{i-1} + \langle y \rangle_{i+1}$
    2) calculate $\hat{r} = r - \mathsf{sign}(r) \cdot 2^{\ell-1} \in \mathbb{Z}_{2^{\ell-1}}$
    3) extract $\hat{r}$ as $\{r_0, \ldots, r_{\ell-2}\}$
    4) secret share $r_k$ to $P_1$ and $P_2$ for $k \in \mathbb{Z}_{\ell-1}$, taking the
       smallest prime $p$ bigger then $\ell$; $P_1$ holds $r_k^{(1)}$, $P_2$ holds
       $r_k^{(2)}$ and $r = r_k^{(1)} + r_k^{(2)} \pmod{p}$;
• $P_{i-1}$ and $P_{i+1}$ pick $\Delta \leftarrow \mathbb{Z}_2$ using same seed.
Online:
- $P_{i-1}$ and $P_{i+1}$ do:
    1) set $m = \langle x \rangle_i - \langle y \rangle_i \mod 2^{\ell}$;
    2) set $\hat{m} = m - \mathsf{sign}(m) \cdot 2^{\ell-1}$ and bitexact it as
       $\{\hat{m}_k \in \{0,1\}\}_{k \in \mathbb{Z}_{\ell-1}}$ while $\sum_{k=0}^{\ell-2} 2^{\ell-2-k} \hat{m}_k = \hat{m}$;
    3) set $\hat{m}_{\ell-1} = 1$ and $(r_{\ell-1}^{(1)}, r_{\ell-1}^{(2)}) = (0,0)$;
    4) $P_1$ sets $m_k^{(1)} = \hat{m}_k + r_k^{(1)} - 2\hat{m}_k \cdot r_k^{(1)}$ for $k \in \mathbb{Z}_{\ell}$.
    5) $P_2$ sets $m_k^{(2)} = r_k^{(2)} - 2\hat{m}_k \cdot r_k^{(2)}$ for $k \in \mathbb{Z}_{\ell}$.
    6) pick same random values $\{w_k\}_{k \in \mathbb{Z}_{\ell}} \in (\mathbb{Z}_p^*)^{2\ell}$ via PRF
       with seed $\eta_{1,2}$;
    7) $P_1$ calculates ${m'}_k^{(1)} = \sum_{t=1}^{k} m_t^{(1)} - 2 \cdot m_k^{(1)} + 1$ and
       $u_k^{(1)} = w_k \cdot {m'}_k^{(1)} \cdot (1 \oplus \mathsf{sign}(m) \oplus \hat{m}_k \oplus \Delta) + w_k \cdot$
       $(\mathsf{sign}(m) \oplus \hat{m}_k \oplus \Delta)$ for $k \in \mathbb{Z}_{\ell}$;
    8) $P_2$ calculates ${m'}_k^{(2)} = \sum_{t=1}^{k} m_t^{(2)} - 2 \cdot m_k^{(2)}$ and
       $u_k^{(2)} = w_k \cdot {m'}_k^{(1)} \cdot (1 \oplus \mathsf{sign}(m) \oplus \hat{m}_k \oplus \Delta)$ for $k \in \mathbb{Z}_{\ell}$;
    9) pick a random permutation $\pi$ via PRF with seed $\eta_{1,2}$ and
       permute the list $\{\hat{u}_k^{(1)}\}_{k \in \mathbb{Z}_{\ell}} = \pi(\{\hat{u}_k^{(1)}\}_{k \in \mathbb{Z}_{\ell}})$;
   10) send $\{\hat{u}_k^{(1)}\}_{k \in \mathbb{Z}_{\ell}}$ and $\{\hat{u}_k^{(2)}\}_{k \in \mathbb{Z}_{\ell}}$ to $P_i$;
- $P_i$ calculates $\hat{u}_k = \hat{u}_k^{(1)} + \hat{u}_k^{(2)} \mod p$ for $k \in \mathbb{Z}_{\ell}$;
- $P_i$ sets $t = \mathsf{sign}(r)$ if $\exists \hat{u}_k = 0$ for $k \in \mathbb{Z}_{\ell}$ else
  $t = 1 \oplus \mathsf{sign}(r)$ to $P_j$, for $j \in \{i-1, i+1\}$;
- Output $(\Delta, t)$.
```

Figure 11: Secure Comparison Protocol.

Appendix A. Secure Comparison and Conditional OT

Fig. 10 depicts the functionality $\mathcal{F}_{\mathsf{cmp}}[i]$ of the secure comparison, which takes the secure shared values of x and y and outputs $z = (x < y)$. It secret shares z as $z = z_0 + z_1$, while P_i holds z_0 and other two parties hold z_1. Fig. 11 illustrates the comparison protocol proposed by Lu *et al.*. [21], [22] It implements the functionality $\mathcal{F}_{\mathsf{cmp}}[i]$. For two values x and y, the protocol first computes $x - y$ and then checks whether the sign bit is 0. This method requires that both x and y have a non-negative sign bit. Therefore, for k-bit valid inputs x and y, the computation is performed over a $(k+1)$-bit ring. It achieves lower communication overhead compared to other constant-round approaches, such as garbled circuit and function secret sharing. Their protocol realizes the secure comparison functionality $\mathcal{F}_{\mathsf{cmp}}[i]$ with

```
Protocol $\Pi_{\mathsf{cotp}}(\langle x \rangle^{\ell}, \langle y \rangle^{\ell}, (\langle L_j \rangle^{\ell'})_{j \in \mathbb{Z}_N}, (\langle R_j \rangle^{\ell'})_{j \in \mathbb{Z}_N})$

$P_j$ and $P_k$ hold the common seed $\eta_{j,k} \in \{0,1\}^{\lambda}$.
Input : $\langle\cdot\rangle^{\ell}$-shared value of $x$, $y$, and$\langle\cdot\rangle^{\ell'}$-shared value of list
$(L_j)_{j \in \mathbb{Z}_N}, (R_j)_{j \in \mathbb{Z}_N}$.
Output : List $(I^{(j)})_{j \in \mathbb{Z}_N}$, if $x \geq y$, $I^{(j)} = L_j$, otherwise,
$I^{(j)} = R_j$.
Preprocessing:
- $P_0$ picks $(r'_{0,j}, r''_{0,j})_{j \in \mathbb{Z}_N}$ and secret share them to $P_1$ and
  $P_2$;
- $P_1$ picks $(r'_{1,j}, r''_{1,j})_{j \in \mathbb{Z}_N}$ and secret share them to $P_0$ and
  $P_2$;
- $P_0$ and $P_1$ generate $\langle x' \rangle_2$ and $\langle y' \rangle_2$ with same seed. $P_0$ and
  $P_2$ generate $\langle x' \rangle_1$ and $\langle y' \rangle_1$ with same seed.
- $P_1$ and $P_0$ generate $\langle x'' \rangle_2$ and $\langle y'' \rangle_2$ with same seed. $P_1$
  and $P_2$ generate $\langle x'' \rangle_0$ and $\langle y'' \rangle_0$ with same seed.
- Input $\langle x' \rangle_2$, $\langle x' \rangle_1$, $\langle y' \rangle_2$, $\langle y' \rangle_1$ and $\mathsf{sid} = 1$ to $\mathcal{F}_{\mathsf{cmp}}[0]$ and
  receive $\delta_0$ from $\mathcal{F}_{\mathsf{cmp}}[0]$;
- Input $\langle x'' \rangle_2$, $\langle x'' \rangle_0$, $\langle y'' \rangle_2$, $\langle y'' \rangle_0$ and $\mathsf{sid} = 2$ to $\mathcal{F}_{\mathsf{cmp}}[1]$
  and receive $\delta_1$ from $\mathcal{F}_{\mathsf{cmp}}[1]$;
- Set $\langle \delta_0 \rangle = (\delta_0, 0, 0)$ and $\langle \delta_1 \rangle = (0, \delta_1, 0)$.
- Evaluate $\gamma_{0,j} = \delta_0 + r'_{0,j} - 2\delta_0 \cdot r'_{0,j}$ and
  $\gamma_{1,j} = \delta_1 + r'_{1,j} - 2\delta_1 \cdot r'_{1,j}$ using $\mathcal{F}_{\mathsf{mult}}$ under replicated
  share.
- Evaluate $\Gamma_{0,j} = r''_{0,j}(1 - 2\delta_0)$ and $\Gamma_{1,j} = r''_{1,j}(1 - 2\delta_1)$
  using $\mathcal{F}_{\mathsf{mult}}$ under replicated share.
Online:
- $P_i$, for $i \in \{1,2\}$ does
    – $c_i^x = \langle x \rangle_{3-i} - \langle x' \rangle_{3-i}$, $c_i^y = \langle y \rangle_{3-i} - \langle y' \rangle_{3-i}$
    – sends to each other
    – computes $\langle x' \rangle_0 = \langle x \rangle_0 + d_1^x + d_2^x$,
      $\langle y' \rangle_0 = \langle y \rangle_0 + d_1^y + d_2^y$
    – input $\langle x' \rangle_0$ and $\langle y' \rangle_0$ to $\mathcal{F}_{\mathsf{cmp}}[0]$
- $P_i$, for $i \in \{0,2\}$ does
    – $d_i^x = \langle x \rangle_{2-i} - \langle x'' \rangle_{2-i}$, $d_i^y = \langle y \rangle_{2-i} - \langle y'' \rangle_{2-i}$
    – sends to each other
    – computes $\langle x'' \rangle_1 = \langle x \rangle_1 + d_1^x + d_2^x$,
      $\langle y'' \rangle_1 = \langle y \rangle_1 + d_1^y + d_2^y$
    – input $\langle x'' \rangle_1$ and $\langle y'' \rangle_1$ to $\mathcal{F}_{\mathsf{cmp}}[1]$
• All parties calculate $\langle \Delta_j \rangle^{\ell'} = \langle L_j \rangle^{\ell'} - \langle R_j \rangle^{\ell'}$;
• All parties invoke $\mathcal{F}_{\mathsf{mult}}$ to evaluate
  $\langle \sigma_{0,j} \rangle = \langle \Delta_j \rangle^{\ell'} \langle \gamma_{0,j} \rangle^{\ell'} + \langle R_j \rangle^{\ell'} + \langle \Gamma \rangle_{0,j}^{\ell'}$ and
  $\langle \sigma_{1,j} \rangle = \langle \Delta_j \rangle^{\ell'} \langle \gamma_{1,j} \rangle^{\ell'} + \langle R_j \rangle^{\ell'} + \langle \Gamma \rangle_{1,j}^{\ell'}$;
• All parties reveal $\sigma_{0,j}$ to $P_1$ and $P_2$;
• All parties reveal $\sigma_{1,j}$ to $P_0$;
• $\mathcal{F}_{\mathsf{cmp}}[0]$ outputs $t_0$ to $P_0$;
• $\mathcal{F}_{\mathsf{cmp}}[1]$ outputs $t_1$ to $P_1$;
• $P_0$ sends $\zeta_{0,j} = (t_0 - r'_{0,j})$ and
  $\eta_{0,j} = (t_0 - r'_{0,j})(\langle \Delta_j \rangle_1^{\ell'} + \langle \Delta_j \rangle_2^{\ell'}) - r''_{0,j}$ to $P_1$ and $P_2$;
• $P_1$ sends $\zeta_{1,j} = (t_1 - r'_{1,j})$ and
  $\eta_{1,j} = (t_1 - r'_{1,j})(\langle \Delta_j \rangle_0^{\ell'} + \langle \Delta_j \rangle_2^{\ell'}) - r''_{1,j}$ to $P_0$
• $P_1$ and $P_2$ calculate
  $I^{(j)} = \sigma_{0,j} + \zeta_{0,j}(1 - 2\delta_0)\langle \Delta_j \rangle_0 + \eta_{0,j}(1 - 2\delta_0)$;
• $P_0$ calculates
  $I^{(j)} = \sigma_{1,j} + \zeta_{1,j}(1 - 2\delta_1)\langle \Delta_j \rangle_1 + \eta_{1,j}(1 - 2\delta_1)$;
```

Figure 12: The Conditional Oblivious Transfer with Comparison Predicate.

inputs $\langle x \rangle$ and $\langle y \rangle$ as follows: In the offline phase, all parties provide $\langle x\rangle_{i-1}$, $\langle x\rangle_{i+1}$, $\langle y\rangle_{i-1}$, and $\langle y\rangle_{i+1}$ to $\mathcal{F}_{\mathsf{cmp}}$; $\mathcal{F}_{\mathsf{cmp}}$ outputs z_1 to P_{i-1} and P_{i+1} in the online phase, P_{i-1} and P_{i+1} input $\langle x\rangle_i$ and $\langle y\rangle_i$. The functionality then outputs z_0 to P_i, such that $z_0 \oplus z_1 = z$, where z is the result of the comparison.

Building on this comparison functionality $\mathcal{F}_{\mathsf{cmp}}$, we design and implement a COTP protocol to obliviously select the node index and feature index for the next evaluation round.

For secret-shared input L and R, COTP output plaintext value I using the output (z_0, z_1) of $\mathcal{F}_{\mathsf{cmp}}$. It can be expressed as $I = z \cdot (L - R) + R$. Let $\Delta = L - R$, then the core of the COTP evaluation lies in computing $z \cdot \Delta$. Obtaining the output of the COTP via this multiplication of $z \cdot \Delta$ requires at least two rounds: in the first round, the bit z is reshared into an additively replicated secret sharing form; in the second round, the multiplication is computed locally and the result is opened. As a result, obtaining the final opened COTP output requires a total of three communication rounds (one round for $\mathcal{F}_{\mathsf{cmp}}$).

3-round COTP. It is possible to reduce the number of rounds for COTP from three to two by trading off additional communication. Fig. 12 illustrates our SCOT protocol. Since $\mathcal{F}_{\mathsf{cmp}}$ already consumes one round, achieving a two-round protocol means that the conversion of $z = z_0 \oplus z_1$ into the arithmetic domain, the multiplication $z \cdot \Delta$, and the opening must all be completed within a single round. To meet this requirement, we shift as much computation as possible to the offline phase. We aim to construct the protocol that P_0 can randomize z_0 and directly send the masked value to P_1 and P_2. Once P_1 and P_2 receive this message, they can locally reconstruct I without any further interaction. First, we have $z = z_0 + z_1 - 2z_0z_1$ in arithmetic form. Considering P_{i-1} and P_{i+1} obtaining I within two rounds, we observe that $I := z\Delta + R$ can be expanded as follows.

$$\begin{aligned}
I :=& z\Delta + R \\
=& (z_0 + z_1 - 2z_0z_1)\Delta + R \\
=& (z_1 + r' + (z_0 - r')(1 - 2z_1) - 2z_1r')\Delta + R \\
=& (z_1 + r' - 2z_1r')\Delta + R + (z_0 - r')(1 - 2z_1)\Delta \\
=& (z_1 + r' - 2z_1r')\Delta + R + (z_0 - r')(1 - 2z_1)\langle\Delta\rangle_i \\
&\quad + (z_0 - r')(1 - 2z_1)(\langle\Delta\rangle_{i-1} + \langle\Delta\rangle_{i+1}) \\
=& (z_1 + r' - 2z_1r')\Delta + R + \overbrace{(z_0 - r')}^{P_i \text{ holds}}\overbrace{(1 - 2z_1)\langle\Delta\rangle_i}^{P_{i-1}/P_{i+1} \text{ hold}} \\
&+ \overbrace{((z_0 - r')(\langle\Delta\rangle_{i-1} + \langle\Delta\rangle_{i+1}) - r'')}^{P_i \text{ holds}}\ \overbrace{(1 - 2z_1)}^{P_{i-1}/P_{i+1} \text{ hold}} \\
&\quad + r''(1 - 2z_1)
\end{aligned} \tag{1}$$

The logic behind expanding this equation is to decompose I into two parts: terms that can be precomputed (and opened to P_{i-1}/P_{i+1}), and product terms of z_0 and others held by P_{i-1} and P_{i+1}. For these product terms, our goal is to introduce random values (r', r'') on z_0 so that P_i can directly open it to P_{i-1}/P_{i+1} without compromising privacy. This allows P_{i-1} and P_{i+1} to locally reconstruct I. Specifically, let r' and r'' be random numbers generated in the offline phase and known only to P_i. We proceed as follows: 1. The term $(z_1 + r' - 2z_1r')\Delta + r''(1 - 2z_1) + R$ can be precomputed (or calculated in the same round of $\mathcal{F}_{\mathsf{cmp}}$) and opened to P_{i-1} and P_{i+1}. 2. For the term related to online generated z_0, $(z_0 - r')(1 - 2z_1)\langle\Delta\rangle_i$, once P_i obtains z_0, it can compute and send $z_0 - r'$ to P_{i-1} and P_{i+1}. Since both P_{i-1} and P_{i+1} already hold $(1 - 2z_1)\langle\Delta\rangle_i$, they can locally evaluate $(z_0 - r')(1 - 2z_1)\langle\Delta\rangle_i$. 3. Similarly, for the term $((z_0 - r')(\langle\Delta\rangle_{i-1} + \langle\Delta\rangle_{i+1}) - r'')(1 - 2z_1)$, since $z_0 - r'$ has been sent to P_{i-1} and P_{i+1}, directly revealing $(z_0 - r')(\langle\Delta\rangle_{i-1})$ will leak $\langle\Delta\rangle_{i-1}$. We introduce r'' to mask it. P_i sends $((z_0 - r')(\langle\Delta\rangle_{i-1} + \langle\Delta\rangle_{i+1}) - r'')$, and P_{i-1} and P_{i+1}, who hold $(1 - 2z_1)$, can locally compute $((z_0 - r')(\langle\Delta\rangle_{i-1} + \langle\Delta\rangle_{i+1}) - r'')(1 - 2z_1)$. The above process requires only one round of communication, and together with one round from $\mathcal{F}_{\mathsf{cmp}}$, it results in a total of two communication rounds. For the case where P_i needs to obtain I, we can swap the roles of P_i and P_{i-1}/P_{i+1} and re-execute the above procedure to derive it (at the same round). Considering that $\mathcal{F}_{\mathsf{cmp}}$ requires the offline inputs $\langle\cdot\rangle_{i-1}$ and $\langle\cdot\rangle_{i+1}$, we address this issue using reshare operation. For the details, refer to Appendix. A. Fig. 12 depicts our COTP protocol. We ensure a two-round online cost by switching the roles of P_0 and P_1/P_2 and executing the comparison-based selection process twice in parallel. It is worth noting that in the protocol by Lu et al., the functionality $\mathcal{F}_{\mathsf{cmp}}[0]$ requires $\langle\cdot\rangle_1$ and $\langle\cdot\rangle_2$ as inputs during the offline phase. In contrast, for the invocation of $\mathcal{F}_{\mathsf{cmp}}[1]$ used to generate z', the offline inputs are $\langle\cdot\rangle_0$ and $\langle\cdot\rangle_2$. As shown in Fig. 12, we pre-assign the corresponding secret-sharing slices for the feature and threshold inputs—specifically, $\langle x'\rangle_1$, $\langle x'\rangle_2$, and $\langle x''\rangle_0$, $\langle x''\rangle_2$, in the offline phase. Then, once the threshold and feature values are provided as inputs, we reconstruct the new shares by combining them with the pre-assigned offline secret-sharing slices accordingly. In particular, we aim to transfer input $\langle x\rangle$ to $\langle x'\rangle$ with pre-assigned shares $\langle x'\rangle_1$ and $\langle x'\rangle_2$. Obviously, it holds $\langle x'\rangle_0 = \langle x\rangle_0 + \langle x\rangle_1 + \langle x\rangle_2 - \langle x'\rangle_1 - \langle x'\rangle_2$. We let P_1 calculate $d_1 = \langle x\rangle_2 - \langle x'\rangle_2$ and P_2 calculate $d_2 = \langle x\rangle_1 - \langle x'\rangle_1$, then they can exchange d_1 and d_2 to reconstruct $\langle x'\rangle_0$.

In our COTP protocol, we rely on the multiplication protocol Π_{mult} defined over replicated shares. The workflow of Π_{mult} [1] is as follows: For multiplication $z = x \cdot y$ with input $\langle x\rangle$, $\langle y\rangle$ and output $\langle z\rangle$, it can be written as

$$\begin{aligned}
xy =& \langle x\rangle_0\langle y\rangle_0 + \langle x\rangle_1\langle y\rangle_1 + \langle x\rangle_2\langle y\rangle_2 + \langle x\rangle_0\langle y\rangle_1 + \langle x\rangle_1\langle y\rangle_0 \\
&+ \langle x\rangle_0\langle y\rangle_2 + \langle x\rangle_2\langle y\rangle_0 + \langle x\rangle_1\langle y\rangle_2 + \langle x\rangle_2\langle y\rangle_1 \ .
\end{aligned}$$

Let P_i calculate $\langle z\rangle_{i-1} = \langle x\rangle_{i-1}\langle y\rangle_{i-1} + \langle x\rangle_{i-1}\langle y\rangle_{i+1} + \langle x\rangle_{i+1}\langle y\rangle_{i-1}$, it holds $z = z_0 + z_1 + z_2$. Then P_i sends $\langle z\rangle_{i-1}$ to P_{i+1}.

Efficiency. $\mathcal{F}_{\mathsf{cmp}}$ uses a PRG-based optimization only in the offline phase, achieving a communication overhead of one round with $2\ell \log \ell$ bits in the online phase and $\ell \log \ell$ bits in the offline phase. Notably, their protocol can be adapted to achieve information-theoretic (IT) security by removing the PRG optimization. (Overall, our scheme can achieve IT security by removing the use of pseudorandom generators (PRGs).) Our COTP protocol, illustrated in Fig. 11, ensures a total cost of three rounds through two invocations of the comparison primitive. Considering that calculation and opening of selection index I incurs a overall communication cost of $14\ell'$ bits (where $\ell' = \log m$ or $\log n$, depending on whether the index corresponds to a node or a feature), each COTP in our decision tree requires a total online communication cost of $14(\log m + \log n) + 4\ell \log \ell$.

Theorem 1. *Let $\langle x \rangle$ and $\langle y \rangle$ be secret shared over $\mathbb{Z}_{2^\ell}$, $\langle L_j \rangle^{\ell'}$ and $\langle R_j \rangle^{\ell'}$ are secret shared over $\mathbb{Z}_{2^{\ell'}}$. The protocol Π_{cotp} as depicted in Fig. 12 realizes $\mathcal{F}_{\mathsf{cotp}}$ in the $(\mathcal{F}_{\mathsf{cmp}}, \mathcal{F}_{\mathsf{mult}})$-hybrid model against semi-honest PPT adversaries who can statically corrupt up to one party.*

Proof. We construct a PPT simulator Sim that interacts with $\mathcal{F}_{\mathsf{cotp}}$ and simulates the view of Adv.

Case 1: P_0 (or P_1) is corrupted.

Simulator: The simulator Sim internally simulates $\mathcal{F}_{\mathsf{cmp}}$ and $\mathcal{F}_{\mathsf{mult}}$. Sim simulates the following interactions with P_0 controlled by Adv.

- plays the role of P_1 and sends random values as the share of $r'_{1,j}$ and $r''_{1,j}$ to P_0.
- simulates $\mathcal{F}_{\mathsf{cmp}}$ and sends randomly picked value δ_1 to P_0.
- simulates $\mathcal{F}_{\mathsf{mult}}$ and sends randomly picked value $\langle \gamma_{0,j} \rangle_1$, $\langle \gamma_{0,j} \rangle_2$, $\langle \gamma_{1,j} \rangle_1$ and $\langle \gamma_{1,j} \rangle_2$ to P_0.
- simulates $\mathcal{F}_{\mathsf{mult}}$ and sends randomly picked value $\langle \Gamma_{0,j} \rangle_1$, $\langle \Gamma_{0,j} \rangle_2$, $\langle \Gamma_{1,j} \rangle_1$ and $\langle \Gamma_{1,j} \rangle_2$ to P_0.
- plays the role of P_2 and sends randomly picked value d_0^x and d_0^y to P_0.
- simulates $\mathcal{F}_{\mathsf{mult}}$ and sends randomly picked value $\langle \sigma_{0,j} \rangle_1$, $\langle \sigma_{0,j} \rangle_2$, $\langle \sigma_{1,j} \rangle_1$ and $\langle \sigma_{1,j} \rangle_2$ to P_0.
- simulates $\mathcal{F}_{\mathsf{cmp}}$ and sends randomly picked value $t_0 \in \mathbb{Z}_2$ to P_0.
- picks $\eta_{1,j} \in \mathbb{Z}_{2^{\ell'}}$ and $\zeta_{1,j} \in \mathbb{Z}_{2^{\ell'}}$.
- receive $I^{(j)}$ from $\mathcal{F}_{\mathsf{cotp}}$, set $\sigma_{1,j} = I^{(j)} - \zeta_{1,j}(1 - 2\delta_1)\langle \Delta_j \rangle_1 + \eta_{1,j}(1 - 2\delta_1)$
- plays the role of P_1 and P_2 and reveals $\sigma_{1,j}$ to P_0.
- plays the role of P_1 and sends randomly picked value $\eta_{1,j}$ and $\zeta_{1,j}$ to P_0.

Indistinguishability: $\mathsf{Ideal}_{\mathcal{F},\mathsf{Sim},\mathcal{Z}}(1^\kappa)$ is indistinguishable with $\mathsf{Real}_{\Pi_{\mathsf{cotp}},\mathsf{Adv}}(1^\kappa)$.

Proof. It is easy to verify that the share of $r'_{1,j}$, $r''_{1,j}$, and δ_1, $\langle \gamma_{0,j} \rangle_1$, $\langle \gamma_{0,j} \rangle_2$, $\langle \gamma_{1,j} \rangle_1$, $\langle \gamma_{1,j} \rangle_2$, $\langle \Gamma_{0,j} \rangle_1$, $\langle \Gamma_{0,j} \rangle_2$, $\langle \Gamma_{1,j} \rangle_1$, $\langle \Gamma_{1,j} \rangle_2$, $\langle \sigma_{0,j} \rangle_1$, $\langle \sigma_{0,j} \rangle_2$, $\langle \sigma_{1,j} \rangle_1$, $\langle \sigma_{1,j} \rangle_2$ are indistinguishable with random value, since they are secret share as the output of $\mathcal{F}_{\mathsf{mult}}$, $\mathcal{F}_{\mathsf{cmp}}$ and secret sharing scheme.

For d_2^x and d_2^y, since they can be viewed as ciphertexts encrypted using $\langle x'' \rangle_0$ and $\langle y'' \rangle_0$ through one-time pad encryption, and $\langle x'' \rangle_0$ and $\langle y'' \rangle_0$ are not visible to P_0, they are indistinguishable from random values.

For $\eta_{1,j}$ and $\zeta_{1,j}$, since they can be viewed as ciphertexts encrypted using r' and r'' through one-time pad encryption, and r' and r'' are not visible to P_0, they are indistinguishable from random values.

For $\sigma_{1,j}$, it is calculated by $I^{(j)}$ in the ideal world, which is the same as the real world.

□

Case 2: P_2 is corrupted.

Simulator: The simulator Sim internally simulates $\mathcal{F}_{\mathsf{cmp}}$ and $\mathcal{F}_{\mathsf{mult}}$. Sim simulates the following interactions with P_0 controlled by Adv.

- plays the role of P_0 and sends random values as the share of $r'_{0,j}$ and $r''_{0,j}$ to P_2.
- plays the role of P_1 and sends random values as the share of $r'_{1,j}$ and $r''_{1,j}$ to P_2.
- simulates $\mathcal{F}_{\mathsf{cmp}}[0]$ and sends randomly picked value δ_0 to P_0.
- simulates $\mathcal{F}_{\mathsf{cmp}}[0]$ and sends randomly picked value δ_1 to P_0.
- simulates $\mathcal{F}_{\mathsf{mult}}$ and sends randomly picked value $\langle \gamma_{0,j} \rangle_0$, $\langle \gamma_{0,j} \rangle_1$, $\langle \gamma_{1,j} \rangle_0$ and $\langle \gamma_{1,j} \rangle_1$ to P_2.
- simulates $\mathcal{F}_{\mathsf{mult}}$ and sends randomly picked value $\langle \Gamma_{0,j} \rangle_0$, $\langle \Gamma_{0,j} \rangle_1$, $\langle \Gamma_{1,j} \rangle_0$ and $\langle \Gamma_{1,j} \rangle_1$ to P_2.
- plays the role of P_1 and sends randomly picked value c_2^x and c_2^y to P_0.
- plays the role of P_0 and sends randomly picked value d_2^x and d_2^y to P_0.
- simulates $\mathcal{F}_{\mathsf{mult}}$ and sends randomly picked value $\langle \sigma_{0,j} \rangle_0$, $\langle \sigma_{0,j} \rangle_1$, $\langle \sigma_{1,j} \rangle_0$ and $\langle \sigma_{1,j} \rangle_1$ to P_2.
- picks $\eta_{0,j} \in \mathbb{Z}_{2^{\ell'}}$ and $\zeta_{0,j} \in \mathbb{Z}_{2^{\ell'}}$;
- receives $I^{(j)}$ from $\mathcal{F}_{\mathsf{cotp}}$, set $\sigma_{0,j} = I^{(j)} - \zeta_{0,j}(1 - 2\delta_0)\langle \Delta_j \rangle_0 + \eta_{0,j}(1 - 2\delta_0)$
- plays the role of P_1 and P_2 and reveals $\sigma_{0,j}$ to P_0.
- plays the role of P_1 and sends randomly picked value $\eta_{0,j}$ and $\zeta_{0,j}$ to P_2.

Indistinguishability: $\mathsf{Ideal}_{\mathcal{F},\mathsf{Sim},\mathcal{Z}}(1^\kappa)$ is indistinguishable with $\mathsf{Real}_{\Pi_{\mathsf{cotp}},\mathsf{Adv}}(1^\kappa)$.

Proof. The proof of indistinguishability is similar to the case of a malicious P_0. □

□