Private Decision Tree Evaluation with Malicious Security via Function Secret Sharing

Jiaxuan Fu[1], Ke Cheng[1], Yuheng Xia[1], Anxiao Song[1], Qianxing Li[1], and Yulong Shen[1]

School of Computer Science and Technology, Xidian University, Xi'An, China
{jiaxuanfu,yuhengxia,anxiaosong,qianxing.li}@stu.xidian.edu.cn
chengke@xidian.edu.cn
ylshen@mail.xidian.edu.cn

Abstract. Private Decision Tree Evaluation (PDTE) allows the client to use a decision tree classification model on the server to classify their private data, without revealing the data or the classification results to the server. Recent advancements in PDTE have greatly enhanced its effectiveness in scenarios involving semi-honest security, offering a viable and secure alternative to traditional, less secure approaches for decision tree evaluation. However, this model of semi-honest security may not always align well with real-world problems. In this work, we present FSSTree, a malicious-secure three-party PDTE protocol using function secret sharing (FSS). FSSTree achieves its high performance against malicious adversaries via several innovative cryptographic designs. Especially, 1) we transform a comparison operation into a prefix parity query problem, allowing us to implement malicious-secure comparisons rapidly using lightweight and verifiable FSS. 2) Building upon this, we further propose a constant-round protocol for securely evaluating Conditional Oblivious Selection (COS). 3) We utilize these optimized protocols to enhance the PDTE processes, achieving a considerable decrease in both communication costs and the number of rounds. Experimental results show that FSSTree costs less than 6 seconds per tree on a dataset with 50 thousand samples which outperforms Mostree (ACSAC 2023) by more than 28×.

Keywords: Private Decision Tree Evaluation · Function Secret Sharing · Malicious Model.

1 Introduction

Decision trees are highly popular and influential models in machine learning applied to various domains such as e-commerce recommendations, spam filtering, and medical diagnostics. In the cloud-assisted service paradigm, a model provider can deploy a trained decision tree model in the cloud, enabling remote evaluation of classification models with clients' inputs. This paradigm offers well-known benefits to both model providers and clients, including scalability, ubiquitous access, and economical cost. While outsourcing decision tree evaluation to the

Table 1. Comparison of PDTE Protocols

Protocol	Parties	Security	Feature Selection	Conditional Selection	Communication Costs*	Rounds*
Liu et al. [17]	2PC	○	-	HE,MT	$O(2^d\ell)$	$O(d)$
Zheng et al. [25]	2PC	○	MT	MT,SS	$O(2^d n\ell)$	$O(l)$
Ma et al. [18]	2PC	◑	OT	GC, OT	$O(dnl)$	$O(d)$
Ji et al. [13]	3PC	○	FSS	FSS	$O(d(\log m + \log n + 2l))$	$O(d)$
Bai et al. [2]	3PC	●	RSS	RSS, FSS	$O(dn\ell)$	$O(d\ell)$
Ours	3PC	●	RSS	FSS	$O(dn\ell)$	$O(d)$

* Communications and Rounds only statistics online evaluation phase.
○: semi-honest, ◑: one-side malicious, ●: malicious, m: the number of tree nodes, n: the number of features, d: the longest depth of a tree, ℓ: the bit-length of feature values. HE: homomorphic encryption, MT: triple multiplication, SS: secret sharing, GC: garbled circuit, OT: oblivious transfer, RSS: replicated secret sharing, FSS: function secret sharing.

cloud is quite beneficial, it also raises critical privacy concerns on the decision tree model and the input data. In this paradigm, the outsourced evaluation requires that the model remains confidential, while the model provider should not know the client's input data. This situation necessitates the development of Private Decision Tree Evaluation (PDTE) to ensure there is no leakage of information about the client's input and the model provider's decision tree.

Recently, many PDTE protocols [2, 13, 17, 18, 25] have been proposed with different security and efficiency trade-offs. As shown in Table 1, most existing PDTE protocols [13, 17, 25] assume that the adversary is semi-honest, meaning the adversary honestly follows the protocol specification. However, the semi-honest assumption is difficult to satisfy in reality. Recently, Bai et al. propose Mostree to achieve security against malicious adversaries with sublinear communication, which represents the state-of-the-art. Their design introduces an assistant computing party to form a three-party architecture, in which a malicious adversary can compromise one party, and the compromised party could be anyone of the model provider, client, and assistant computing party.

However, as an initial attempt, Mostree is not entirely satisfactory and its performance still needs to be optimized. In particular, the number of communication rounds during the online phase is linearly related to the number of bits for value representation, leading to significant communication latency. This is because the node feature selection of PDTE involves a secure comparison protocol under a malicious model. Mostree implements these secure comparisons by bit-wise solving on additively shared data, using MSB extraction techniques, which necessitates $O(l)$ communication rounds. We find that in a LAN setting, over 85% of the time overhead during the online phase of Mostree is caused by this malicious secure comparison protocol. This overhead is even more pronounced in a WAN setting, further diminishing the practicality of the scheme. Consequently, to effectively mitigate the implications of pronounced communication latency, it is essential to devise a new PDTE scheme that operates under a malicious model but requires fewer communication rounds, thereby enhancing efficiency and practical applicability.

1.1 Related Works

Semi-honest PDTE schemes. Most of the previous works are mainly focused on PDTE against semi-honest adversaries. Part of this works [3, 12, 17, 23] use homomorphic encryption (HE) to achieve decision tree evaluation. However, the substantial computational and communication costs associated with this approach render it impractical for real-world applications. Subsequent work [21] introduced the notion of path cost without traversing the entire decision tree to reduce communication overhead. Following the concept of path cos, subsequent work has been proposed to further improve computational efficiency, such as [9], [25], and [14]. Recently, [13] has achieved efficient decision tree evaluation based on the Function Secret Sharing (FSS) scheme. This work is built upon distributed point functions (DPF) and distributed comparison functions (DCF), implementing oblivious transfer (OT) protocols and conditional oblivious transfer protocols (COT) with optimal communication complexity. However, this work is designed only to defend against semi-honest adversaries.

Malicious PDTE schemes. PDTE schemes against malicious adversaries have become a focal point for scholars in recent years. Existing work [18, 21, 23] focuses on protecting models from attacks by malicious clients. [21, 23] use Zero-Knowledge proofs (ZKPs) techniques to detect malicious input from clients, [18] uses the malicious-secure version of the Garbled Circuits (GC) protocol [1] to detect malicious input. [8] proposed a PDTE protocol using SPD$\mathbb{Z}_{2^k}$ in a dishonest majority setting. The work most similar to ours is [2], which introduces a Private Decision Tree Evaluation (PDTE) protocol named Mostree. This protocol maintains security in a malicious adversary environment while achieving sublinear communication overhead. Mostree operates in a three-party honest-majority setting, with an untrusted computation party (CP) assisting the feature owner (FO) and the model owner (MO) in secure computation. However, this article does not optimize comparison operations within the decision tree, leading to performance limitations in this aspect.

1.2 Our Contributions

This paper proposes FSSTree, a malicious-secure PDTE protocol using function secret sharing (FSS). Our design follows the same architecture of Mostree [12], and also works in the three-party honest-majority setting [1, 10, 17, 21], yet with significant optimizations to achieve largely boosted performance compared to the state-of-the-art work. Our contributions are summarized as follows:

- First, we transform a comparison operation into a prefix parity query problem, and based on this, propose a verifiable comparison protocol (Π_{VCMP}) with plaintext input. The protocol implements malicious-secure comparisons rapidly using lightweight and verifiable FSS, which do not make use of any public key operations or arithmetic MPC.
- Second, by providing the lightweight Π_{VCMP} protocol, we further propose a secure conditional oblivious selection protocol (Π_{COS}) that achieves malicious security with constant communication rounds. We carefully combine

the proposed Π_{COS} protocol and consistency check mechanism of replicated secret sharing (RSS), to design FSSTree, achieving a considerable decrease in both communication costs and the number of rounds.
– Third, we formally prove the security of all our protocols using the proof-by-simulation technique [15]. We implement FSSTree and evaluate its performance over various public datasets. Experiments demonstrate that FSSTree can obtain better performance in time and communication overhead. Compared with Mostree, the online runtime over a WAN environment is up to 23.5× better. In the meantime, FSSTree offers up to 4.1× savings in communication costs.

2 Preliminaries

In this section, we initially introduce decision tree evaluation, followed by an introduction to various secret-sharing primitives and their foundational computation protocols. Table 2 lists the main notations utilized throughout our work.

Notations	Definitions
$[\![x]\!]$	$\binom{n}{n}$-additive secret shares of the value x
$[\![x]\!]_b$	the $\binom{n}{n}$-shares of x stored in partyP_b
$\langle x \rangle$	$\binom{3}{2}$-additive secret shares of the value$x$
$\langle x \rangle_b = ([\![x]\!]_b, [\![x]\!]_{b+1})$	the $\binom{3}{2}$-shares of x stored in party P_b
ℓ	the bit length of the value x
λ	the security parameter of FSS

2.1 Decision Tree Evaluation

Decision tree evaluation utilizes a binary tree structure for classification, integrating decision nodes and leaf nodes. Each decision node contains a feature attribute and a threshold, guiding the evaluation process. Starting from the root, the feature vector is compared to these thresholds at each node, directing the path to the left child for values that surpass the threshold and to the right for others. This traversal continues until a leaf node is reached, which bears a classification label. This label, generated in the leaf, signifies the classification result of the input feature vector.

2.2 Additive Secret Sharing

Sharing Semantics. Our scheme works on a three-party computation (3PC) model where parties $P = (P_0, P_1, P_2)$ are connected by bidirectional synchronous channels. To further elucidate, we use the following sharing semantics:

$\binom{n}{n}$**-sharing** $[\![x]\!]^n$. A secret value $x \in \mathbb{Z}_{2^\ell}$ is said to be $[\![\cdot]\!]$-shared among n parties, if the party P_b for $b \in \mathbb{Z}_n$ holds $[\![x]\!]^n_b = x_b$ such that $x = \sum_{i=0}^{n-1} x_i$. For the brevity of the description, we omit the corner n when there is no ambiguity. We use $n = 2$ and 3 in this paper.

$\binom{3}{2}$-**sharing** $\langle x \rangle$(a.k.a. Replicated Secret Sharing, RSS). A secret value $x \in \mathbb{Z}_{2^\ell}$ is said to be $\langle \cdot \rangle$-shared among P, if party P_b for $b \in \mathbb{Z}_3$ holds $\langle x \rangle_b = (\llbracket x \rrbracket_b, \llbracket x \rrbracket_{b+1})$.

We extend the above definition to vectors, where $\vec{x} \in \mathbb{Z}^m$ denotes an m-dimensional vector. Accordingly, we use $\llbracket \vec{x} \rrbracket$ and $\langle \vec{x} \rangle$ to denote $\binom{n}{n}$-sharing and $\binom{3}{2}$-sharing of a vector $\vec{x}$, respectively. Without special declaration, all computations are performed in $\mathbb{Z}_{2^\ell}$. For brevity, we omit the notation (mod 2^ℓ). It is important to note that the additive sharing schemes discussed above can be seamlessly expanded to handle Boolean-type data, which we refer to as Boolean sharing.

Semi-honest Protocols. $\binom{3}{2}$- sharing mainly includes the following (semi-honest) computation protocols:

- $\mathsf{Reshare}$: Given $\binom{3}{3}$-shared values $\llbracket x \rrbracket$, to obtain $\langle x \rangle$, P_b for $b \in \mathbb{Z}_3$ locally computes $x'_b = x_b + r_b$, where $r_0 + r_1 + r_2 = 0$ and the auxiliary parameter r_b can be generated offline as [22]. Then P_b sends x'_b to P_{b-1} and sets $\langle x \rangle_b = (x'_b, x'_{b+1})$.
- Add: Given secret shares $\langle x \rangle$ and $\langle y \rangle$, to obtain $\langle x + y \rangle$, P_b for $b \in \mathbb{Z}_3$ locally computes $\langle x+y \rangle_b = (x_b + y_b, x_{b+1} + y_{b+1})$. Additionally, for a public constant c, $\langle x \rangle + c$ can be computed as $(x_0 + c, x_1, x_2)$ where P_b can compute its share locally.
- Mul: Given secret shares $\langle x \rangle$ and $\langle y \rangle$, to obtain $\langle xy \rangle$, P_b for $b \in \mathbb{Z}_3$ locally computes $z_b = x_b y_b + x_{b+1} y_b + x_b y_{b+1}$ so that $z = z_0 + z_1 + z_2 = xy$ and z are $\binom{3}{3}$-shared. Then, parties invoke $\mathsf{Reshare}$ with inputting $\llbracket z \rrbracket$ to get $\langle z \rangle = \langle xy \rangle$.Additionally, for a public constant c, $c \cdot \langle x \rangle$ can be computed as $(c \cdot x_0, c \cdot x_1, c \cdot x_2)$ where P_b can compute its share locally.

Malicious Protocols. Our work uses the following assumed ideal functionalities to achieve malicious security:

- $\mathcal{F}_{\mathsf{mul}}^{\llbracket \cdot \rrbracket}$: Given $\binom{2}{2}$-shared $\llbracket x \rrbracket$, $\llbracket y \rrbracket$, share $\llbracket x \cdot y \rrbracket$ between two parties.
- $\mathcal{F}_{\mathsf{rand}}$: Sample a random value $r \leftarrow \mathbb{Z}$ and share $\langle r \rangle$ between three parties.
- $\mathcal{F}_{\mathsf{open}}$: Given a $\binom{3}{2}$-shared $\langle x \rangle$, reveal x to all three parties.
- $\mathcal{F}_{\mathsf{share}}$: Inputting a secret x held by P_b, share $\langle x \rangle$ between three parties.
- $\mathcal{F}_{\mathsf{recon}}$: Given a $\binom{3}{2}$-shared $\langle x \rangle$ and a party index b, reveal x to P_b.
- $\mathcal{F}_{\mathsf{mul}}^{\langle \cdot \rangle}$: Given secret shares $\langle x \rangle$, $\langle y \rangle$, share $\langle x \cdot y \rangle$ between three parties.
- $\mathcal{F}_{\mathsf{os}}$: Given a $\binom{3}{2}$-shared index $\langle i \rangle$ and a $\binom{3}{2}$-shared list $\langle L \rangle$, share $\langle L[i] \rangle$ between three parties.

All these functionalities can be securely computed with malicious security using well-established protocols [2,7,10,16,24]. In addition, whenever a three-party RSS-based secure computation (3PC for short) is used in a black-box manner, we will directly utilize a 3PC ideal functionality $\mathcal{F}_{\mathsf{3PC}}^{\mathbb{Z}}$ directly for simplicity, which ensures privacy and correctness for a secure computation over $\mathbb{Z}$ against a malicious adversary.

2.3 Function Secret Sharing

A Function Secret Sharing (FSS) scheme [4, 5] for a function family $\mathcal{F}$ splits a function $f(x) \in \mathcal{F}$ into two additive shares $f_0(x)$ and $f_1(x)$, such that each share hides f and ensures that $\forall x \in \mathbb{G}^{in}$, $f_0(x) + f_1(x) = f(x)$.

Definition 1. *(FSS: Syntax [4,5]). A (two-party) FSS scheme is a pair of probabilistic polynomial-time (PPT) algorithms* $\{\mathsf{Gen}(\cdot), \mathsf{Eval}(\cdot)\}$ *such that:*

- $\mathsf{Gen}(1^\lambda, f)$ *is a key generation algorithm that, given the security parameter* λ *and the description of a function* f*, outputs a pair of functional keys* $(\boldsymbol{k}_0, \boldsymbol{k}_1)$*. Each key is capable of efficiently describing* f_0 *and* f_1 *without revealing* f*.*
- $\mathsf{Eval}(b, \boldsymbol{k}_b, x)$ *is a polynomial-time evaluation algorithm that, given the party index* $b \in \{0, 1\}$*, the key* $\boldsymbol{k}_b$*, and the input* x*, it outputs an additive share* $f_b(x)$*, such that* $f_0(x) + f_1(x) = f(x)$*.*

$(\boldsymbol{k}_0, \boldsymbol{k}_1)$ are called FSS *keys* or FSS *shares*. The number of bits required to store an FSS key is called *key size*. In this work, we focus on the Distributed Point Function (DPF), which is an FSS scheme for point functions that evaluate a non-zero value at exactly one index in their domain. Formally, the definition of such a function is as follows:

$$f^{\bullet}_{\alpha,\beta}(x) = \begin{cases} \beta, & x = \alpha \\ 0, & otherwise \end{cases} \tag{1}$$

2.4 Verifiable Distributed Point Function (VDPF)

Since the party responsible for generating and distributing DPF keys may be malicious, it may generate incorrect keys. To prevent this, the Verifiable DPF (VDPF) [6] provides an additional verifiable attribute defined as follows:

Definition 2. *(Verifiable DPF). A (two-party) verifiable distributed point function (VDPF) scheme is a set of probabilistic polynomial-time (PPT) algorithms* $\{\mathsf{Gen}(\cdot), \mathsf{BVEval}(\cdot), \mathsf{BVEval}(\cdot)\}$ *such that:*

- $\mathsf{VDPF.Gen}(1^\lambda, f^{\bullet}_{\alpha,\beta}) \rightarrow (\boldsymbol{k}_0, \boldsymbol{k}_1)$ *is a key generation algorithm that, given the security parameter* λ *and the description of a point function* $f^{\bullet}_{\alpha,\beta}$*, outputs a pair of functional keys* $(\boldsymbol{k}_0, \boldsymbol{k}_1)$*.*
- $\mathsf{VDPF.BVEval}(b, \boldsymbol{k}_b, \{x_i\}_{i\in[L]}) \rightarrow (\llbracket y_b^{(x_i)} \rrbracket_{i\in[L]}, \pi_b)$ *is a polynomial-time evaluation algorithm that, given a batch of* L *inputs* $\{x_i\} i \in [L]$*, outputs a tuple of values. The first set of values are the FSS outputs, where* $\llbracket y_b^{(x_i)} \rrbracket = f^{\bullet}_{\alpha,\beta})(x_i)$*. The second output is a proof* π_b *that is used to verify the well-formedness of the output.*
- $\mathsf{VDPF.Verify}(\pi_0, \pi_1) \rightarrow (\mathsf{Accept}/\mathsf{Reject})$ *is a algorithm that, given a pair of proofs* π_0 *and* π_1*, outputs either* Accept *or* Reject*. The output should only be* Accept *if* $y_0 + y_1$ *defines the truth table of some point function, which occurs if it is non-zero in at most one location.*

3 Secure Conditional Selection

In the PDTE, the most crucial step is conditional selection based on feature value f and threshold t. For each decision node, the formal definition of conditional selection is as follows:

$$next = \begin{cases} l, & f < t \\ r, & otherwise \end{cases} \tag{2}$$

where l and r represent its left and right child index. In this section, we first propose a verifiable comparison protocol with plaintext input that implements the comparison of feature values and thresholds. Based on this, we further design a secure verifiable conditional oblivious selection protocol that hides the access patterns when selecting the child index.

3.1 Verifiable Comparison Protocol with Plaintext Input

We first propose a verifiable comparison protocol (Π_{VCMP}) in the 2PC setting with plaintext input, which will serve as an important component of the subsequent algorithms. To achieve a verifiable comparison, our strategy to address this challenge is to convert the comparison into a prefix parity query and then employ the VDPF scheme [6] for verification.

Given a bitstring $x = x_0 x_1 \cdots x_{n-1}$ of length n, assume that there exists a substring $x_a x_{a+1} \cdots x_{b-1}$ of x, denoted $x[a, b)$, where $0 \le a < b \le n$. The *parity query* for $x[a, b)$ is defined as follows:

$$parity(x[a, b)) := \bigoplus_{i=a}^{b-1} x_i$$

We observe that when comparing a public value x with a public threshold t, if $x < t$ holds, then on the number line, x always appears to the left of t. That is, for the one-hot bit encoding of x in $\mathbb{Z}_{2^\ell}$, denoted as $\hat{\alpha} = \alpha_0 \alpha_1 \cdots \alpha_{\ell-1}$, where $\hat{\alpha}$ comprises all 0s except for a single 1 at α_x, the equation $\alpha_0 \oplus \alpha_1 \oplus \cdots \oplus \alpha_{t-1} = 1$ always holds. This implies that we can transform a comparison of plaintext values into a prefix parity query on $\hat{\alpha}$. A recent work [20] propose a data structure called *parity-segment tree* which can answer parity queries with a total complexity of $O(\ell)$, and further extends this query to the DPF scheme, allowing the implementation of the comparison functionality through the DPF scheme. Inspired by the parity-segment tree and VDPF [6], we propose the verifiable comparison protocol with plaintext input to compute the following function:

$$f^{>}_{y,1}(x) = \begin{cases} 1, & x > y \\ 0, & otherwise \end{cases} \tag{3}$$

We abbreviate the above equation as $1\{x > y\}$. The details of our protocol are shown in Algorithm 1.

Algorithm 1 Verifiable Comparison Protocol with Plaintext Input (Π_{VCMP})

Require: P_b for $b \in \{0,1\}$ holds a verified DPF key k_b for the point function $f^{\bullet}_{y,1}$ and a public index x. Let $\mathsf{Convert}_{\mathbb{G}} : 0,1^{\lambda} \to \mathbb{G}$ be a map converting a random λ-bit string to a pseudorandom group element of $\mathbb{G}$.

Ensure: P_b outputs res_b where $res_0 \oplus res_1 = f^{>}_{y,1}(x)$

P_b for $b \in \{0,1\}$ does:

Let $x = x_1, \cdots, x_\ell \in \{0,1\}^\ell$ be the bit decomposition
Parse $k_b = s^{(0)}||t^{(0)}||CW^{(1)}|| \cdots ||CW^{(\ell+1)}||cs$
$res_b \leftarrow 0$, $d \leftarrow 0$
for $i \in [1, \ell]$ **do**
  Parse $CW^{(i)} = s_{CW}||t^L_{CW}||t^R_{CW}$
  $\tau^{(i)} \leftarrow G(s^{(i-1)}) \oplus (t^{(i-1)} \cdot [s_{CW}||t^L_{CW}||s_{CW}||t^R_{CW}])$
  Parse $\tau^{(i)} = s^L||t^L||s^R||t^R \in \{0,1\}^{2(\lambda+1)}$
  if $x_i == 0$ **then**
    $s^{(i)} \leftarrow s^L, t^{(i)} \leftarrow t^L$
  else
    $s^{(i)} \leftarrow s^R, t^{(i)} \leftarrow t^R$
  end if
  if $d \neq x_i$ **then**
    $d = x_i$
    $res_b = res_b \oplus t^{(i-1)}$
  end if
end for
if $d == 1$ **then**
  $res_b = res_b \oplus t^{(\ell)}$
end if
return res_b

Our protocol requires a pair of verified DPF keys $k_b, b \in \mathbb{Z}_2$, which we generate by invoking $\mathsf{VDPF.Gen}$ and run the DPF key verification protocol in [6] to check the well-formness of the DPF keys. Regarding the key evaluation phase, it can be seen that the execution logic of our protocol is basically the same as that of the $\mathsf{DPF.Eval}$ algorithm, with the only difference being that we introduce res_b and d, which denote a continuously updated parity value and the current traversal direction of the tree, respectively. The protocol evaluates the verified key according to the traversal method in [20], and outputs the value of $f^{>}_{y,1}(x)$ in the XOR-shared form. Due to the use of the key generation and verification process defined by the VDPF, the protocol maintains similar security in that a malformed shared value can be detected when the evaluator is semi-honest, as defined below:

Definition 3. *(Verifiable Comparison Share Integrity, or Security Against Malicious Share Generation). Let k_b be the (possibly maliciously generated) share received by the server P_b. For an adversarially chosen set of inputs $\{x_i\}_{i=1}^{\eta}$, let $(\{y_b^{(x_i)}\}_{i=1}^{\eta}, \pi_b) \leftarrow \mathsf{VDPF.BVEval}(b, k_b, \{x_i\}_{i=1}^{\eta})$ and $\{z_b^{(x_i)}\}_{i=1}^{\eta} \leftarrow \Pi_{\mathsf{VCMP}}(b, k_b, \{x_i\}_{i=1}^{\eta})$. We say that Π_{VCMP} is secure against malicious share generation if the following holds with all but negligible probability over the adversary's choice of*

randomness. If $\mathsf{VDPF.Verify}(\pi_0, \pi_1)$ *outputs* Accept*, then the values* $z_0^{(x_i)} + z_1^{(x_i)}$ *must satisfy:*

$$z_0^{(x_i)} + z_1^{(x_i)} = \begin{cases} 1, & x_i > p \\ 0, & otherwise \end{cases} \tag{4}$$

It should be noted that the result of the above protocol is a bit of $\binom{2}{2}$-boolean sharing, and further, we need to execute a bit injection $\mathcal{F}_{\mathsf{B2A}}$ to convert it to $\binom{2}{2}$-additive sharing. The details of $\mathcal{F}_{\mathsf{B2A}}$ protocol are given in Appendix F.

3.2 Secure Verifiable Conditional Oblivious Selection Protocol

Next, we present a secure verifiable conditional oblivious selection protocol (Π_{cos}) in the 3PC setting, which consists of two parts: the comparison of feature values and thresholds, and the oblivious selection.

Since the comparison protocol given in the previous section requires public input, we first address how to hide the input to preserve privacy, which allows us to use Π_{VCMP} in combination in an additive secret sharing scheme. In general, we can add an offset to the public input x, that is, $x' = x + r$. However, this may lead to an over-ring problem, resulting in incorrect results for Π_{VCMP}. To address this problem, we use the technique from [11,19] to perform the following transformation:

$$\mathsf{MSB}_\ell(x) = \mathsf{MSB}_\ell(x_0) \oplus \mathsf{MSB}_\ell(x_1) \oplus 1\{y_0 + y_1 \leq 2^{\ell-1}\}$$

where $x = x_0 + x_1$, $y_0 = x_0 \bmod 2^{\ell-1}$, and $y_1 = x_1 \bmod 2^{\ell-1}$. Using the above equation, we get

$$\begin{aligned} 1\{f < t\} &= \mathsf{MSB}_n(d = f - t + r) \\ &= \mathsf{MSB}_\ell(d) \oplus \mathsf{MSB}_\ell(2^\ell - r) \\ &\oplus 1\{2^{\ell-1} - y_0 - 1 < y_1\} \oplus 1 \end{aligned}$$

where $y_0 = d \bmod 2^{\ell-1}$ and $y_1 = (2^\ell - r) \bmod 2^{\ell-1}$. Based on the above equation, we only need to invoke Π_{VCMP} to compute $1\{2^{\ell-1} - y_0 - 1 < y_1\}$.

Recall that for the $\langle\cdot\rangle$-shared values $\langle l\rangle$ and $\langle r\rangle$ in the 3PC setting. P_b and P_{b+1} hold common share values l_{b+1} and r_{b+1}. This implies that P_b and P_{b+1} can initially perform a secure comparison of f with t to obtain the result $[\![\theta]\!]$ in $\binom{2}{2}$-sharing form. Next, the parties jointly compute $[\![next]\!]_b^2 = l_{b+1} \cdot [\![\theta]\!] + r_{b+1} \cdot (1 - [\![\theta]\!])$ in an oblivious choice of l_{b+1} or r_{b+1}. This process is repeated for two out of three parties. At the end of the three-round execution, each party holds two $\binom{2}{2}$-shares (since each party executes oblivious selection twice with the other two parties). After that, the parties locally sum up the two $\binom{2}{2}$-shares. In this manner, three parties jointly produce a $\binom{3}{3}$-sharing of $next$, which can be reshared back to an RSS-sharing $\langle next\rangle$. Since the result of the comparison of f and t is the same for each round of execution, the parties will always choose both l or both r, which makes it easy to see that $next$ satisfies Eq. 2. Figure 1 sketches the above process. The details of the Π_{cos} protocol are shown in Algorithm 2.

Algorithm 2 Verifiable Conditional Oblivious Selection Protocol (Π_{cos})

[*Offline phase*]:

Upon initialization, the party $P_b, b \in \mathbb{Z}_3$ does:

Randomly sample $\alpha_b \leftarrow \mathbb{Z}_{2^\ell}$ and $\alpha_b^0 \leftarrow \mathbb{Z}_{2^\ell}$
$\alpha_b^1 = \alpha_b - \alpha_b^0$
$x_b = 2^\ell - \alpha_b$
$y_b = x_b \bmod 2^{\ell-1}$
Randomly sample $y_b^0 \leftarrow \mathbb{Z}_{2^{\ell-1}}$
$y_b^1 = y_b - y_b^0 \bmod 2^{\ell-1}$
$(k_0^\bullet, k_1^\bullet) \leftarrow \mathsf{VDPF.Gen}(1^\lambda, y_b, 1, \{0,1\})$
$c = \mathsf{MSB}_n(x_b) \oplus 1$
Randomly sample c_0 from $\mathbb{Z}_2$
$c_1 = c \oplus c_0$
$k_i = c_i || k_i^\bullet$, for $i \in \{0,1\}$.
P_b sends $(\alpha_b^0, k_b^0, y_b^0)$ to P_{b+2} and sends $(\alpha_b^1, k_b^1, y_b^1)$ to P_{b+1}.
P_{b+1} and P_{b+2} jointly run the DPF key verification protocol from [6] to check the well-formness of DPF keys. If the check fails, abort.
P_{b+1} and P_{b+2} expand its DPF key on domain $\mathbb{Z}_{2^\ell}$ to produce a shared vector $[\![\vec{v}]\!]$ in the $\binom{2}{2}$-sharing. Then, P_{b+1} and P_{b+2} jointly compute:
$[\![t]\!] = \Sigma_{i \in \mathbb{Z}_{2^\ell}} [\![\vec{v}[i]]\!]$
$[\![s]\!] = [\![y]\!]_b - \Sigma_{i \in \mathbb{Z}_{2^\ell}} (i \cdot [\![\vec{v}[i]]\!])$
P_{b+1} and P_{b+2} open $[\![t]\!]$ and $[\![s]\!]$ and check if $t = 1$ and $s = 0$. If the check fails, abort.

[*Online phase*]:

Upon receiving the shared messages $\langle l \rangle$, $\langle r \rangle$ and the shared conditions $(\langle f \rangle, \langle t \rangle)$, the party $P_b, b \in$ does:

$\langle d \rangle = \langle f \rangle - \langle t \rangle$
Set $\langle \alpha_b \rangle = (\alpha_b^0, \alpha_b^1)$, $\langle \alpha_{b+1} \rangle = (0, \alpha_{b+1}^0)$, and $\langle \alpha_{b+2} \rangle = (\alpha_{b+2}^1, 0)$
for $i \in \{0,1,2\}$ **do**
    $\langle d_i \rangle = \langle d \rangle + \langle \alpha_i \rangle$.
    Party P_{i+1} obtains d_i by invoking $\mathcal{F}_{\mathsf{recon}}(\langle d_i \rangle, i+1)$
    Party P_{i+2} obtains d_i by invoking $\mathcal{F}_{\mathsf{recon}}(\langle d_i \rangle, i+2)$
end for
for $i \in \{0,1\}$ **do**
    Parse k_{b+i+1}^i as $c_i || k_i^\bullet$
    $\hat{y}_i^b = d_{b+i+1} \bmod 2^{\ell-1}$
    $res_i^b \leftarrow \Pi_{\mathsf{VCMP}}(k_i^\bullet, 2^{\ell-1} - \hat{y}_i^b - 1)$
    $\theta_i^b = i \cdot \mathsf{MSB}_n(d_{b+i+1}) \oplus c_i \oplus res_i^b$
end for

The party P_b with P_{b+1} does:

Obtain $[\![\theta_0^b]\!]$ by invoking $\mathcal{F}_{\mathsf{B2A}}$ with P_b input θ_0^b and P_{b+1} input θ_1^{b+1}.
$[\![next_0]\!] = [\![l]\!]_{b+1} \cdot [\![\theta_0^b]\!] + [\![r]\!]_{b+1} \cdot (1 - [\![\theta_0^b]\!])$

The party P_b with P_{b+2} does:

Obtain $[\![\theta_1^b]\!]$ by invoking $\mathcal{F}_{\mathsf{B2A}}$ with P_b input θ_1^b and P_{b+2} input θ_0^{b+1}.

The party P_b with P_{b+2} do:

$[\![next_1]\!] = [\![l]\!]_b \cdot [\![\theta_1^b]\!] + [\![r]\!]_b \cdot (1 - [\![\theta_1^b]\!])$

The party $P_b, b \in \mathbb{Z}_3$ does:

$[\![next]\!] = [\![next_0]\!] + [\![next_1]\!]$
$\langle next \rangle \leftarrow$ reshare $[\![next]\!]$ using the PRG-based re-sharing trick.
return $\langle next \rangle$

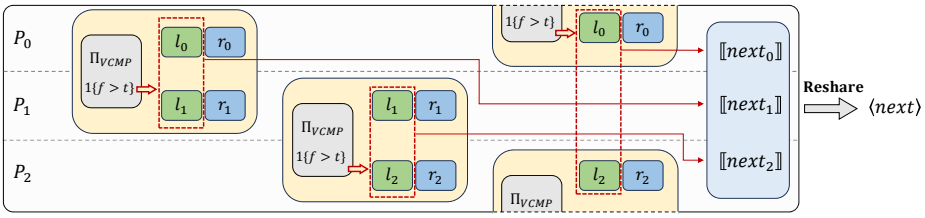

Fig. 1. Our design of conditional oblivious selection.

Next, we show how our design resists malicious attacks. For our protocol, multiple malicious attacks can occur in different interactive parts. First, when generating and distributing DPF keys, the key provider P_{b+2} can be captured and may generate two different malicious attacks. The corrupted key generator may generate an incorrect DPF key that does not conform to the protocol definition, that is, generate a key for the point function $f^{\bullet}_{\alpha,\beta}$ with $\beta \neq 1$. This will be detrimental to Π_{VCMP} protocol as it operates correctly only when $\beta = 1$. In fact, the corrupted party may use arbitrary key generation strategies to disrupt the normal execution of the protocol. On the other hand, even if the captured party generates the correct DPF key for $f^{\bullet}_{\alpha,\beta}$ and shares it, it may still share an incorrect index $\alpha* \neq \alpha$ between P_b and P_{b+1}. As a consequence, this attack results in an incorrect opening of $d^{in} = d + \alpha + e$, where the error is $e = \alpha^* - \alpha$. This corresponds to an incorrect conditional selection in our design. Our protocol employs the VDPF scheme [6] to prevent a malicious party from generating incorrect DPF keys. This scheme allows two key executors to run an effective verification protocol to test the correctness of the keys. The protocol ensures the well-form of the DPF keys and verifies the accuracy of their execution. To verify that the key conforms to the definition of the protocol, we use the lightweight verification method from [2] to ensure that $\beta = 1$. Specifically, during the DPF key full-domain verification, a $(\frac{2}{2})$-shared vector $[\![v]\!]$ can be obtained. By checking whether the sum of all elements in $[\![v]\!]$ is equal to 1, we can determine whether $\beta = 1$. Additionally, by computing $[\![s]\!] = [\![\alpha]\!]_b - \Sigma_{i \in \mathbb{Z}_{2^\ell}}(i \cdot [\![\vec{v}[i]]\!])$ and opening $[\![s]\!]$, and then verifying if $s = 0$, we can ascertain the correctness of the α corresponding to the point function.

Second, we notice that even if the DPF keys are correctly generated and α is correctly shared, how to correctly reconstruct the public input d_{b+2} to P_b and P_{b+1} remains a question. If only P_b and P_{b+1} are involved in the reconstruction of d_{b+2}, it is impossible to ensure correctness because an adversary who corrupts either P_b or P_{b+1} can always add errors during the reconstruction process. Fortunately, for the reconstruction of d_{b+2}, the parties can define a consistent $\langle\cdot\rangle$-sharing of $\langle\alpha\rangle$ from $[\![\alpha]\!] = ([\![\alpha]\!]_0, [\![\alpha]\!]_1)$ as:

$$\langle\alpha\rangle = (([\![\alpha]\!]_0, [\![\alpha]\!]_1), ([\![\alpha]\!]_1, 0), (0, [\![\alpha]\!]_0))$$

Therefore, the parties can reconstruct the offset input $\langle d\rangle = \langle d\rangle + \langle\alpha\rangle$ through $\mathcal{F}_{\mathsf{recon}}$, which ensures consistency.

Finally, a corrupted party may maliciously add errors before resharing the $\binom{3}{3}$-sharing selected secret, resulting in an additive attack. All the above attacks only compromise correctness, not privacy. For the first two attacks, we have implemented a well-formed consistency verification. So far, our protocol is only limited to adding errors in the re-sharing phase, which will be verified in subsequent use.

4 The Proposed FSSTree Scheme

In this section, we present the FSSTree scheme, in which the encoding of the decision tree is similar to previous work [13]. For a decision tree with depth d and number of nodes m, our design follows the encoding of previous work [2] to hide data access patterns. The main idea is that during the privacy-preserving evaluation, once a leaf node is reached, the server repeatedly accesses itself until the protocol reaches the d steps corresponding to the depth of the tree. Thus, we treat each leaf node as a decision node and have its left and right children point to themselves. The advantage of this design is that it eliminates the need to fill the entire tree into a full binary tree with still m nodes, thereby saving on storage overhead. We further encode the decision tree as a multi-dimensional array $\vec{T}$, where each element of the array represents a node of the decision tree. The i-th node contains the following five values: left child index l, right child index r, feature selection vector $\vec{f}$, threshold value t and classification value c, denoted as $T[i] \leftarrow l||r||\vec{f}||t||c$. Note that we use a selection vector (for index k, its selection vector $\vec{k}$ comprises all 0s except for a single 1 at k.) to encode the feature value index, since the input features are generally not excessive, which is relevant to reduce subsequent computational overhead.

We assume that this encoding is performed by the model provider because the operation is lightweight and its cost is negligible. For a decision node, the classification value is set to a random value. For a leaf node, the left/right child indexes are both points to the itself. The threshold value is set to a random value. Since there is only one leaf node in a path and the decision tree must rest on some leaf node at the end of its evaluation, the above encoding ensures the correct evaluation of the decision tree.

Since Π_{cos} protocol used in FSSTree remains an issue that is being limited by adding errors, We use SPDZ-like MACs to detect these errors. Specifically, we pre-compute the MAC values of contents within the tree $\vec{T}$ during the setup phase to facilitate consistency verification of the Π_{cos}. Intuitively, any error introduced would affect the relationship between the chosen data and its MAC, allowing detection through Batch MAC Check protocol Π_{MacCheck} [2].

The algorithm 3 shows the details of the FSSTree scheme. During the setup phase, the model provider encodes the decision tree T and shares it between each party. Upon receiving $\vec{T}$, the parties jointly compute the MAC values for each node in the decision tree $\vec{T}$ and store them in $\vec{M}$. At this point, the parties are ready to proceed with decision tree inference.

Algorithm 3 The FSSTree scheme (Π_{FSSTree})

Require: A shared tree model $\langle\vec{T}\rangle$, an input feature vector $\langle\vec{F}\rangle$
Ensure: The RSS sharing result $\langle res\rangle$

[*Setup*]:

The model provider encodes the decision tree T as a array $\vec{T}$. Then the parties jointly invoke $\mathcal{F}_{\mathsf{share}}$ in which the model provider inputs $\vec{T}$.
The parties jointly invoke $\alpha \leftarrow \mathcal{F}_{\mathsf{rand}}$ to share a MAC key $\alpha \in \mathbb{Z}_{2^\ell}$
for $i = 1$ to m **do**
    Compute MACs: $\langle\vec{M}\rangle = \mathcal{F}_{\mathsf{mul}}^{\langle\cdot\rangle}(\langle\alpha\rangle, \langle\vec{T}[i]\rangle)$
end for
The parties run $\Pi_{\mathsf{MacCheck}}(\langle\vec{T}\rangle, \langle\vec{M}\rangle)$. Abort if the check fails.
Output $(\langle\vec{T}\rangle, \langle\vec{M}\rangle)$.

[*Inference*]:

Upon receiving $\langle\vec{F}\rangle$ from the client, the party $P_b, b \in$ does:
Parties jointly initialize $\langle res\rangle = \langle 0\rangle$
Parse $\langle l\rangle||\langle r||\langle\vec{f}\rangle||\langle t\rangle||\langle c\rangle \leftarrow \langle\vec{T}\rangle[0]$
Parse $\langle l_m\rangle||\langle r_m\rangle||\langle\vec{f}_m\rangle||\langle t_m\rangle||\langle c_m\rangle \leftarrow \langle\vec{M}\rangle[0]$
for $i = 1$ to d **do**
    $\langle x\rangle = \sum_{j\in[n]}\langle\vec{F}[j]\rangle \cdot \langle\vec{f}[j]\rangle$
    $l_{pack} = \langle l\rangle||\langle l_m\rangle$
    $r_{pack} = \langle r\rangle||\langle r_m\rangle$
    $\langle\eta\rangle||\langle\eta_m\rangle = \Pi_{\mathsf{cos}}((l_{pack}, r_{pack}), (\langle x\rangle, \langle t\rangle))$
    $\langle\vec{T}[\eta]\rangle||\langle\vec{M}[\eta]\rangle = \mathcal{F}_{\mathsf{os}}(\langle\eta\rangle, \langle\vec{T}\rangle||\langle\vec{M}\rangle)$
    Parse $\langle l\rangle||\langle r\rangle||\langle\vec{f}\rangle||\langle t\rangle||\langle c\rangle \leftarrow \langle\vec{T}[\eta]\rangle$
    Parse $\langle l_m\rangle||\langle r_m\rangle||\langle\vec{f}_m\rangle||\langle t_m\rangle||\langle c_m\rangle \leftarrow \langle\vec{M}\rangle[\eta]$
    $\langle res\rangle = \langle c\rangle$
end for
Use Π_{MacCheck} to check each pair of $\langle\eta\rangle||\langle\eta_m\rangle$ from Π_{cos} and $\langle\vec{T}[\eta]\rangle||\langle\vec{M}[\eta]\rangle$ from $\mathcal{F}_{\mathsf{os}}$. Abort if the check fails.
Invoke $\mathcal{F}_{\mathsf{recon}}$ to reconstruct res for the client.

During the inference phase, we start from the root node and use the selection vector $\vec{f}$ to obtain the required feature value x for the current node. Subsequently, by invoking the Π_{cos} protocol, we determine the next direction of traversal based on the comparison result between the feature value x and the threshold t. It is important to note that we simultaneously select both the child selection vector and its MAC value, resulting in a pair of selection outcomes $(\vec{\eta}, \vec{\eta}_m)$. The child selection vector $\vec{\eta}$ is used to choose the next node. At the end of the entire loop, by calling $\mathcal{F}_{\mathsf{MacCheck}}$, we verify each pair of selections during the traversal. If there is any MAC mismatch, abort. Finally, through $\mathcal{F}_{\mathsf{recon}}$, only the client can obtain the final result res.

Security. We formally prove the security of all our protocols using the proof-by-simulation technique, the details can be found in Appendix **??**.

5 Experiment

5.1 Setup

Our scheme is implemented in Python 3.8 with the Pytorch library. Our experiments are carried out on a server running Ubuntu 20.04 with an Intel Core i9-10900k @ 3.7GHz processor, 128GB RAM, and NVIDIA GeForce RTX 3090. We use the Linux tc tool to simulate local-area networks (LAN, RTT: 0.1 ms, 1 Gbps), metropolitan-area networks (MAN, RTT: 6 ms, 100 Mbps), and wide-area networks (WAN, RTT: 80 ms, 40 Mbps). FSSTree works in the secret sharing domain, where all data are represented as integers and shared over the ring $\mathbb{Z}_{2^{32}}$ with the security parameter $\lambda = 128$. Consistent with the prior art [2], we conduct experiments on the public data sets from the UCI repository(https://archive.ics.uci.edu/ml) as listed in Table 3. Since the data sets above consist of rational numbers, we scale the data to an integer representation with a factor of 10^6 for computation in our protocols.

Table 3. Paramters of Public Datasets

Dataset	Depth d	Features n	#(Nodes) m
wine	5	7	23
breast	7	12	43
digits	15	47	337
spambase	17	57	171
diabetes	28	10	787
Boston	30	13	851
MNIST	20	784	4179

Table 4. Runtimes(s) Comparision of Conditional Oblivious Selection Protocol

Input Size	Scheme	LAN			WAN		
		CPU	GPU	A.R.	CPU	GPU	A.R.
10^2	ASS-based [2]	0.16	0.36	**0.45**	64.08	64.32	**0.99**
	Ours	**0.03**	**0.07**	0.44	**3.24**	**3.35**	0.97
10^4	ASS-based [2]	**0.60**	0.67	0.89	74.01	74.36	1.00
	Ours	3.55	**0.08**	**43.59**	**5.16**	**3.18**	**1.62**
10^6	ASS-based [2]	60.06	51.04	1.18	1290.82	1300.81	0.99
	Ours	**44.84**	**3.22**	**13.91**	**102.80**	**72.17**	**1.42**

* A.R. represents the accelerative ratio.

5.2 Evaluation of Verifiable Conditional oblivious selection

Since our work focuses on optimizing conditional oblivious selection, we first benchmark the performance of Π_{COS} as the building blocks used for FSSTree. We perform experiments in different network settings with different input sizes and compare them with the ASS-based protocol used in the state-of-the-art [2] to demonstrate the superiority of our protocol. Since the protocol provided in [2] cannot support GPU acceleration, we re-implement the conditional oblivious selection performed in their protocol based on their open-source code (consisting

of one verifiable comparison and one verifiable dot multiplication) using our experimental environment to make a fair comparison.

Table 4 shows the detailed results of the comparison. In most cases, our protocol performs better. Only in the LAN setting is our protocol slower than ASS-based protocols, especially when the input size is 10^4. This is because our protocol is based on FSS, which has better communication overhead, but also higher computation overhead compared to the ASS-based protocol, and thus does not have an advantage when computing small batches of data in a good network environment. When the network environment is poor or the input size is large, our protocol performs significantly better than the ASS-based protocol. On the other hand, when the computation is accelerated using GPUs, all computations need to be performed on the Video RAM. However, the data exchanged among parties first needs to be transmitted to the CPU RAM, causing substantial IO overhead and thus impacting the performance of the protocol. Since our protocol has constant communication rounds, it is more suitable for acceleration using GPUs. From the experimental results, it can be seen that the ASS-based protocol cannot be accelerated by GPUs in all network settings, while our protocol can be accelerated by GPUs even in the worst network settings.

5.3 Evaluation of FSSTree scheme

We evaluate the FSSTree scheme using different public datasets and compare it with the state-of-the-art work Mostree.Compared to Mostree, during the key generation phase, we need to perform an additional key generation and full domain evaluation for DPF keys in the conditional oblivious selection operation, which takes 5 minutes for one FSS key. Since the number of comparisons of the PDTE is only related to the depth of the decision tree and only one value is used for comparison at a time, we consider the additional overhead in this phase to be acceptable. Furthermore, our decision tree encoding and sharing phase is the same as in Mostree. Next, we focus on the performance of the online decision tree evaluation phase.

Table 5 illustrates the runtime and communication costs for the online evaluation phase between Mostree and our scheme in the LAN, MAN, and WAN network settings. Our scheme demonstrates a lower runtime across the various network settings than Mostree, particularly in the WAN setting, where our work

Table 5. Runtime(s) and Communication Costs(KB) Comparisons of FSSTree scheme

Datasets	Runtime						Comm. Costs	
	LAN		MAN		WAN			
	Mostree	Ours	Mostree	Ours	Mostree	Ours	Mostree	Ours
wine	0.56	0.05	25.65	1.34	317.79	14.54	22.82	6.32
breast	0.68	0.06	35.96	1.83	443.38	19.87	31.45	8.36
digits	3.72	2.01	77.94	4.69	954.75	41.81	104.32	54.84
spambase	2.01	0.17	87.14	4.33	1081.71	46.72	130.19	74.11
diabetes	3.07	0.27	143.85	7.12	1782.19	76.18	121.89	29.53
Boston	3.42	0.31	154.11	7.59	1909.18	81.56	136.93	37.96
MNIST	6.10	3.88	106.15	9.43	1281.20	56.05	1175.51	1109.53

improves the evaluation runtime by approximately $1.5 - 23.5\times$ over Mostree in different datasets. Regarding the MNIST and digits datasets, in the LAN setting, our work has only a $1.5\times$ improvement compared to Mostree, as these two datasets have more features and nodes, leading to plenty of multiplication operations. However, when the network environment becomes worse, e.g., in MAN and WAN settings, our work achieves $11.2-22.3\times$ improvement. This is because the feature comparison of Mostree requires a large number of interactions and is thus greatly affected by communication latency. In contrast, our work has constant communication rounds, and its runtime mainly depends on the depth of the decision tree, making it less susceptible to communication delays.

Regarding communication overhead, since our Π_{COS} protocol features significantly lower communication rounds and costs, and the main overhead of PDTE stems from condition selection, FSSTree offers better communication efficiency compared to Mostree, which offers up to $4.1\times$ savings in communication costs. We also give the theoretical analysis of our scheme in Appendix A, the above experimental results are consistent with our analysis.

Table 6. Runtime(s) on CPU and GPU

Dataset	Scheme	LAN			MAN			WAN		
		CPU	GPU	A.R.	CPU	GPU	A.R.	CPU	GPU	A.R.
digits	Mostree	3.72	3.38	1.10	77.94	78.89	0.99	954.75	960.60	0.99
	Ours	2.01	0.69	2.91	4.69	3.98	1.18	41.82	41.54	1.01
MNIST	Mostree	6.10	4.63	1.32	106.15	105.38	1.01	1281.20	1283.30	1.00
	Ours	3.88	1.04	3.73	9.43	5.46	1.73	56.05	55.33	1.01

For digits and MNIST datasets, we accelerated their computation by the GPUs, with the results illustrated in Table 6. Our work achieves an accelerative ratio of $2.9-3.5\times$ in the LAN setting, while Mostree is only $1.1-1.3\times$. Whereas in the MAN setting, our scheme can still be accelerated using the GPUs, Mostree suffers from a communication bottleneck and even runs slower using the GPUs. In the WAN setting, neither scheme will have acceleration using the GPUs, and the bottleneck at this point comes entirely from communication, so our scheme has better performance in this setting.

6 Conclusion

In this paper, we propose FSSTree, a secure three-party PDTE protocol under the malicious model, which enables clients and model providers to evaluate decision trees without revealing both clients' inputs and model privacy. Technically, we design a verifiable comparison protocol with plaintext input, and based on this, propose a constant-round protocol for securely evaluating conditional oblivious selection. This serves as the foundational bedrock for FSSTree, which enhances the PDTE processes to achieve a considerable decrease in both runtime and communication costs. Extensive experiments demonstrated that compared with the state-of-the-art, FSSTree achieves up to $23.5\times$ better online runtime over a WAN setting, as well as enjoys up to $4.1\times$ savings in communication cost.

A Complexity Analysis

In this section, we analyze the communication complexity and round complexity of our proposed protocols.

For the verifiable plaintext comparison protocol, it completely follows the VDPF key generation process with a key size of $2(\ell(\lambda+2)+\lambda+1)$, so they have the same computational complexity and communication complexity. For the evaluation phase, there is no interaction generation. The computational complexity depends on the bit length of the input x, that is, $O(\ell)$, which is the same as the complexity of the current state-of-the-art plaintext comparison protocol [11, 20] under the semi-honest model.

For the verifiable conditional oblivious selection protocol, each party in the offline phase receives two VDPF keys of length $\ell(\lambda+2)+\lambda+1$, and four variables of length ℓ, which incurs 2 rounds of communication with $2(\ell(\lambda+2)+\lambda+1)+4\ell$ bits. In the verification, each party exchanges π and reconstructs t and s. This incurs 2 rounds of communication with 6ℓ bits. During the online phase, our design requires 6 calls to $\mathcal{F}_{\mathsf{recon}}$ to open the public input, resulting in 4 communication rounds for each server, along with 4ℓ bits. 2 calls to protocol $\mathsf{B2A}$ incur 6 communication rounds with 10ℓ bits. Finally, re-sharing incurs 1 communication round with 2ℓ bits. Thus, our design features a constant complexity level in terms of communication rounds, which offers a significant advantage over the linear-round complexity seen in previous work [2].

For the FSSTree protocol, its setup process is almost identical to previous work [2], with our protocol additionally handling the VDPF keys used for comparison. Specifically, for each comparison during the inference stage, an additional VDPF key of length $\ell(\lambda+2)+\lambda+1$ is generated, along with an additional round of communication of 3 bits to check the well-formness of the VDPF key.

For the inference stage, due to our use of a more efficient comparison protocol, the number of communication rounds required to execute feature comparisons comes to the level of $O(1)$. This results in our protocol having a communication round complexity of $O(d)$.

B Security Definition

Definition 4. *Let $\mathcal{F}$ be the three-party functionality. A protocol Π securely computes $\mathcal{F}$ with abort in the presence of one malicious party, if for every party P_b corrupted by a probabilistic polynomial time (PPT) adversary $\mathcal{A}$ in the real world there exists a PPT simulator S in the ideal world with F, such that,*

$$\{\mathit{IDEAL}_{\mathcal{F},\mathcal{S}(z),b(x_0,x_1,x_2,n)}\} \stackrel{c}{\equiv} \{\mathit{REAL}_{\Pi,\mathcal{A}(z),b(x_0,x_1,x_2,n)}\}$$

where $x_i \in \{0,1\}^*$ is the input provided by P_b for $b \in \mathcal{Z}_3$, and $z \in \{0,1\}^*$ is the auxiliary information that includes the public input length information $\{|x_b|\}_{j\in\mathcal{F}_3}$. The protocol Π securely computes $\mathcal{F}$ with abort in the presence if one malicious party with statistical error $2^{-\lambda}$ if there exists a negligible function $\mu(\cdot)$ such that the distinguishing probability of the adversary is less than $2^{-\lambda}+\mu(\kappa)$

C Security Proof of Verifiable Comparison Protocol Π_{VCMP}

We first prove that the verifiable plaintext comparison protocol given in Section 3.1 is secure. We will focus on proving the following lemma.

Lemma 1. *(Detection of Malicious Function Shares). The verifiable plaintext comparison protocol satisfies Definition 3.*

Proof. We proof this by contradiction. According to Lemma 2 and Lemma 3 in [6], when $\mathsf{VDPF.Verify}(\pi_0, \pi_1)$ outputs Accept, then the values $y_0^{(x_i)} + y_1^{(x_i)}$ must be a subset of the truth table of the point function $f_{p,1}^{\bullet}$. Suppose there exists an input $u \in \{x_i\}_{i=1}^{\eta}$ such that u does not satisfy Eq. 4. We categorize the discussion based on the values of u, when $u > q$, based on the assumption, we have $z_0^{(u)} + z_1^{(u)} = 0$. However, according to the definition of Π_{VCMP}, we know that $z_0^{(u)} + z_1^{(u)} = parity(p[0, u))$, since $u > q$, it follows that $parity(p[0, u)) = z_0^{(u)} + z_1^{(u)} = 1$. This contradicts the assumption that $z_0^{(u)} + z_1^{(u)} = 0$. For the case where $u \leq p$, we can arrive at a similar conclusion, therefore the verifiable plaintext comparison protocol satisfies Definition 3.

D Security Proof of Verifiable Conditional Oblivious Selection Protocol Π_{cos}

Theorem 1. *Π_{cos} securely compute $\mathcal{F}_{cos}$ in the $\{\mathcal{F}_{recon}, \mathcal{F}_{mul}^{[\![\cdot]\!]}\}$-hybrid model in the presence of a malicious adversary in the three-party honest-majority setting.*

Proof. For any PPT adversary $\mathcal{A}$, we construct a PPT simulator $\mathcal{S}$ that can simulate the adversary's view with accessing the functionality $\mathcal{F}_{\mathsf{cos}}$. In the cases where $\mathcal{S}$ aborts or terminates the simulation, $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs.

Simulating offline phase. When the corrupted party P_b plays the role of DPF key generator, $\mathcal{S}$ plays the role of honest parties P_{b+1} and P_{b+2}, and receives a pair of DPF keys from $\mathcal{A}$. $\mathcal{S}$ can check whether the keys are correct and abort if not. When an honest party takes the turn for key generation, $\mathcal{S}$ generates a pair of VDPF keys and samples a random share $r_b \leftarrow \mathbb{Z}_m$. $\mathcal{S}$ sends the key (k_b, r_b) to $\mathcal{A}$. $\mathcal{S}$ runs VDPF verification protocol with $\mathcal{A}$, and aborts if $\mathcal{A}$ aborts in the verification protocol. For all other messages sent from honest parties to the corrupted one, the simulator samples them randomly.

The above simulation is indistinguishable from real-world execution. First, if $\mathcal{A}$ sends incorrect DPF keys, then the real-world execution would have aborted due to VDPF check. In the ideal world, $\mathcal{S}$ can check keys directly, thus the ideal world aborts with indistinguishable probability. When the corrupted party plays the role of DPF evaluator, the DPF keys are honestly generated by $\mathcal{S}$. If $\mathcal{A}$ follows the VDPF key verification protocol, the check will always pass, otherwise the check protocol will abort. Therefore, $\mathcal{S}$ just runs the check protocol with $\mathcal{A}$ as in the real-world protocol, and it can always abort with indistinguishable probability.

Simulating online phase. If the protocol does not abort after the simulating stage, then all DPF keys generated by $\mathcal{A}$ in $\mathcal{S}$ are correct. The only issue in the online phase is whether $\mathcal{A}$ follows the protocol honestly. Also, $\mathcal{S}$ has to successfully extract the error term in the simulation. We assume that the shared messages $\langle l \rangle$, $\langle r \rangle$ and shared conditions $(\langle f \rangle, \langle t \rangle)$ are already correctly shared/simulated in the first place, which means the honest parties hold correct and consistent RSS shares.

For all local computations, $\mathcal{S}$ is easy to simulate. $\mathcal{S}$ only needs to simulate the view between honest parties and the corrupted party and to abort with indistinguishable probability. Note that only a few parts in the selection phase require interaction. The first part is on securely reconstructing d. Since $\langle d \rangle = \langle f \rangle - \langle t \rangle + \langle \alpha \rangle$ and $\langle f \rangle$, $\langle t \rangle$ and $\langle \alpha \rangle$ are correct RSS sharings, $\langle d \rangle$ is also a consistent sharing by definition. This means that $\mathcal{S}$ can cross-check whether $\mathcal{A}$ sends the correct value using the share of honest party P_{b+2}: 1) if $\mathcal{A}$ sends incorrect share, $\mathcal{S}$ just aborts; 2) if $\mathcal{A}$ sends the correct share, $\mathcal{S}$ samples a random $d \in \mathbb{Z}_m$ and computes the shares of P_{b+1} and P_{b+2} according to d and the RSS shares of P_b, and sends to $\mathcal{A}$. In this case, the parties open the random d to $\mathcal{A}$. The second part is from $\mathcal{F}_{\mathsf{B2A}}$. $\mathcal{F}_{\mathsf{B2A}}$ invokes the $\mathcal{F}_{\mathsf{mul}}^{[\![\cdot]\!]}$, whose security is proved in [24]. The last part is from re-sharing the comparison result $[\![next]\!]$ from$\binom{3}{3}$-sharing back to RSS-sharing. Here $\mathcal{A}$ can add an error and $\mathcal{S}$ extracts the error as follows: $\mathcal{S}$ receives the share $next_i^*$ from the corrupted party to an honest party and updates P_{b+1}'s local share correspondingly. $\mathcal{S}$ also locally computes value $next_i$, which is the correct value that the corrupted party should send ($\mathcal{S}$ can do this since he knows all necessary shares to compute z_i). Then, S computes $res \leftarrow next_i - next_i^*$ to $\mathcal{F}_{\mathsf{cos}}$. S sends res to $\mathcal{F}_{\mathsf{cos}}$, completing simulation.

Note that in simulating the open step of d, $\mathcal{S}$ only uses shares of honest parties. These shares are either randomly simulated or computed from available local data. Also, they are consistent with the shares/data held by the corrupted party. Therefore, the simulation is perfectly indistinguishable from real protocol execution. Towards re-sharing phase, the simulation randomly samples P_{b+2}'s share and sends it to $\mathcal{A}$. In the real protocol execution, this is generated by a PRF F. Therefore, the simulation is computationally indistinguishable from real protocol execution due to the security of PRF. Overall, the simulation is in distinguishable from real-world execution.

E Security Proof of the FSSTree Evaluation Protocols Π_{FSSTree}

Theorem 2. *$\Pi_{FSSTree}$ securely compute $\mathcal{F}_{FSSTree}$ in the $\{\mathcal{F}_{recon}, \mathcal{F}_{os}, \mathcal{F}_{coin}, \mathcal{F}_{rand}, \mathcal{F}_{open}, \mathcal{F}_{mul}^{\langle\cdot\rangle}, \mathcal{F}_{CheckZero}\}$-hybrid model in the presence of a malicious adversary in the three-party honest-majority setting.*

Proof. For any PPT adversary $\mathcal{A}$, we construct a PPT simulator $\mathcal{S}$ that can simulate the adversary's view with accessing the functionality $\mathcal{F}_{\mathsf{FSS}}$. In the cases where $\mathcal{S}$ aborts or terminates the simulation, $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs.

Simulating setup. $\mathcal{S}$ samples random shares for $\vec{T}$ and hands the shares to $\mathcal{A}$. $\mathcal{S}$ also randomly samples shares for the honest parties in order to perform subsequent simulations. $\mathcal{S}$ plays the role of $\mathcal{F}_{rand}$ and uses the simulator of $\mathcal{F}_{rand}$ for simulation. For $j \in [m]$, $\mathcal{S}$ plays the role of $\mathcal{F}_{mul}^{\langle\cdot\rangle}$ and receives d_j from the adversary. From d_j, $\mathcal{S}$ adds the error into the share of MAC value $\langle\sigma(\vec{T}[j])\rangle$ for honest party P_{i+1}. $\mathcal{S}$ simulates $\mathcal{F}_{\mathsf{rand}}$ and $\mathcal{F}_{\mathsf{CheckZero}}$ in Π_{MacCheck}. If there exists any $j \in [m]$ such that $d_j \neq 0$ or the simulation aborts from $\mathcal{F}_{\mathsf{rand}}$ and $\mathcal{F}_{\mathsf{CheckZero}}$ in Π_{MacCheck} ($\mathcal{S}$ uses existing simulation strategy for $\mathcal{F}_{\mathsf{rand}}$ and $\mathcal{F}_{\mathsf{CheckZero}}$), $\mathcal{S}$ aborts.

The above simulation is indistinguishable from real protocol execution. Since the protocol is designed in a hybrid model, existing simulation strategy for $\mathcal{F}_{\mathsf{rand}}$, $\mathcal{F}_{\mathsf{mul}}^{\langle\cdot\rangle}$, and $\mathcal{F}_{\mathsf{CheckZero}}$ are available, thus $\mathcal{S}$ uses them directly. $\mathcal{S}$ can also extract the strategy of $\mathcal{A}$ by receiving the error term in $\mathcal{F}_{\mathsf{mul}}^{\langle\cdot\rangle}$ thus $\mathcal{S}$ can abort correspondingly. Note that the MAC we use is over $\mathbb{F}_{2^\ell}$. In real protocol execution, any additive error will incur MAC check fail except with probability $\frac{1}{2^\ell-1}$. Overall, the above simulation is statistically indistinguishable from the real-world execution, with statistical error $\frac{1}{2^\ell-1}$.

Simulating inference. $\mathcal{S}$ plays the role of $\mathcal{F}_{\mathsf{mul}}^{\langle\cdot\rangle}$ and use the simulator of $\mathcal{F}_{\mathsf{cos}}$ for inner product and comparison. If $\mathcal{S}$ receives any non-zero error in simulating $\mathcal{F}_{\mathsf{cos}}$, $\mathcal{S}$ aborts at the end of the protocol. For oblivious selection operation, $\mathcal{S}$ use existing simulation strategy $\mathcal{F}_{\mathsf{os}}$ in [2]. This part is indistinguishable from real-world execution. Since $\mathcal{S}$ receives the error terms from $\mathcal{A}$ per comparison, $\mathcal{S}$ just aborts if $\mathcal{A}$ sends any non-zero error. Overall, the above simulation is statistically indistinguishable from real-protocol execution, with statistical error $\frac{1}{2^\ell-1}$.

F Verifiable B2A Extraction

The B2A extraction protocol for 1bit (2_2)-boolean sharing is shown in Algorithm 4, which requires performing a 2PC (2_2)-additive sharing multiplication. We use the well-established verifiable multiplication protocol proposed in [24] for this purpose.

Algorithm 4 B2A Π_{B2A}

Require: P_b for $b \in \{0,1\}$ holds a boolean-shared data ${}_2[\![x]\!]$
Ensure: Additive share $[\![x]\!]$
1: P_0 set two variables $[\![a]\!]_0 =_2 [\![x]\!]_0, [\![b]\!]_1 = 0$.
2: P_1 set two variables $[\![a]\!]_1 = 0, [\![b]\!]_1 =_2 [\![x]\!]_1$.
3: P_0 and P_1 set $[\![x]\!] = [\![a]\!] + [\![b]\!] - 2 \cdot \mathcal{F}_{\mathsf{mul}}^{[\![\cdot]\!]}([\![a]\!], [\![b]\!])$.
4: **return** $[\![x]\!]$

References

1. Asharov, G., Lindell, Y., Schneider, T., Zohner, M.: More efficient oblivious transfer and extensions for faster secure computation. In: Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security. pp. 535–548 (2013)
2. Bai, J., Song, X., Zhang, X., Wang, Q., Cui, S., Chang, E.C., Russello, G.: Mostree: Malicious secure private decision tree evaluation with sublinear communication. In: Proceedings of the 39th Annual Computer Security Applications Conference. pp. 799–813 (2023)
3. Bost, R., Popa, R.A., Tu, S., Goldwasser, S.: Machine learning classification over encrypted data. Cryptology ePrint Archive (2014)
4. Boyle, E., Gilboa, N., Ishai, Y.: Function secret sharing. In: Annual international conference on the theory and applications of cryptographic techniques. pp. 337–367. Springer (2015)
5. Boyle, E., Gilboa, N., Ishai, Y.: Function secret sharing: Improvements and extensions. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. pp. 1292–1303 (2016)
6. de Castro, L., Polychroniadou, A.: Lightweight, maliciously secure verifiable function secret sharing. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 150–179. Springer (2022)
7. Chida, K., Genkin, D., Hamada, K., Ikarashi, D., Kikuchi, R., Lindell, Y., Nof, A.: Fast large-scale honest-majority mpc for malicious adversaries. In: Advances in Cryptology–CRYPTO 2018: 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19–23, 2018, Proceedings, Part III 38. pp. 34–64. Springer (2018)
8. Damgård, I., Escudero, D., Frederiksen, T., Keller, M., Scholl, P., Volgushev, N.: New primitives for actively-secure mpc over rings with applications to private machine learning. In: 2019 IEEE Symposium on Security and Privacy (SP). pp. 1102–1120. IEEE (2019)
9. De Cock, M., Dowsley, R., Horst, C., Katti, R., Nascimento, A.C., Poon, W.S., Truex, S.: Efficient and private scoring of decision trees, support vector machines and logistic regression models based on pre-computation. IEEE Transactions on Dependable and Secure Computing **16**(2), 217–230 (2017)
10. Furukawa, J., Lindell, Y., Nof, A., Weinstein, O.: High-throughput secure three-party computation for malicious adversaries and an honest majority. In: Annual international conference on the theory and applications of cryptographic techniques. pp. 225–255. Springer (2017)
11. Gupta, K., Jawalkar, N., Mukherjee, A., Chandran, N., Gupta, D., Panwar, A., Sharma, R.: Sigma: secure gpt inference with function secret sharing. Cryptology ePrint Archive (2023)
12. Ishai, Y., Paskin, A.: Evaluating branching programs on encrypted data. In: Theory of Cryptography Conference. pp. 575–594. Springer (2007)
13. Ji, K., Zhang, B., Lu, T., Li, L., Ren, K.: Uc secure private branching program and decision tree evaluation. IEEE Transactions on Dependable and Secure Computing (2022)
14. Kiss, Á., Naderpour, M., Liu, J., Schneider, T., Asokan, N.: Sok: Modular and efficient private decision tree evaluation. Proceedings on Privacy Enhancing Technologies (2019)
15. Lindell, Y.: How to simulate it–a tutorial on the simulation proof technique. In: Tutorials on the Foundations of Cryptography, pp. 277–346. Springer (2017)

16. Lindell, Y., Nof, A.: A framework for constructing fast mpc over arithmetic circuits with malicious adversaries and an honest-majority. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. pp. 259–276 (2017)
17. Liu, L., Su, J., Chen, R., Chen, J., Sun, G., Li, J.: Secure and fast decision tree evaluation on outsourced cloud data. In: Machine Learning for Cyber Security: Second International Conference, ML4CS 2019, Xi'an, China, September 19-21, 2019, Proceedings 2. pp. 361–377. Springer (2019)
18. Ma, J.P., Tai, R.K., Zhao, Y., Chow, S.S.: Let's stride blindfolded in a forest: Sublinear multi-client decision trees evaluation. In: NDSS (2021)
19. Rathee, D., Rathee, M., Kumar, N., Chandran, N., Gupta, D., Rastogi, A., Sharma, R.: Cryptflow2: Practical 2-party secure inference. In: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security. pp. 325–342 (2020)
20. Storrier, K., Vadapalli, A., Lyons, A., Henry, R.: Grotto: Screaming fast $(2 + 1)$-pc for $\mathbb{Z}_{2^n}$ via (2, 2)-dpfs. Cryptology ePrint Archive, Paper 2023/108 (2023), https://eprint.iacr.org/2023/108, https://eprint.iacr.org/2023/108
21. Tai, R.K., Ma, J.P., Zhao, Y., Chow, S.S.: Privacy-preserving decision trees evaluation via linear functions. In: Computer Security–ESORICS 2017: 22nd European Symposium on Research in Computer Security, Oslo, Norway, September 11-15, 2017, Proceedings, Part II 22. pp. 494–512. Springer (2017)
22. Wagh, S., Tople, S., Benhamouda, F., Kushilevitz, E., Mittal, P., Rabin, T.: Falcon: Honest-majority maliciously secure framework for private deep learning. arXiv preprint arXiv:2004.02229 (2020)
23. Wu, D.J., Feng, T., Naehrig, M., Lauter, K.: Privately evaluating decision trees and random forests. Cryptology ePrint Archive (2015)
24. Xu, G., Han, X., Deng, G., Zhang, T., Xu, S., Ning, J., Yang, A., Li, H.: Verifyml: Obliviously checking model fairness resilient to malicious model holder. IEEE Transactions on Dependable and Secure Computing (2023)
25. Zheng, Y., Duan, H., Wang, C.: Towards secure and efficient outsourcing of machine learning classification. In: Computer Security–ESORICS 2019: 24th European Symposium on Research in Computer Security, Luxembourg, September 23–27, 2019, Proceedings, Part I 24. pp. 22–40. Springer (2019)