PriVDT: An Efficient Two-Party Cryptographic Framework for Vertical Decision Trees

Hanxiao Chen, *Student Member, IEEE*, Hongwei Li, *Senior Member, IEEE*, Yingzhe Wang, *Student Member, IEEE*, Meng Hao, *Student Member, IEEE*, Guowen Xu, *Member, IEEE*, and Tianwei Zhang, *Member, IEEE*

***Abstract*—Privacy-preserving decision trees (DTs) in vertical federated learning are one of the most effective tools to facilitate various privacy-critical applications in reality. However, the main bottleneck of current solutions is their huge overhead, mainly due to the adoption of communication-heavy bit decomposition to realize complex non-linear operations, such as comparison and division. In this paper, we present PriVDT, an efficient two-party framework for private vertical DT training and inference in the offline/online paradigm. Specifically, we customize several cryptographic building blocks based on an advanced primitive, Function Secret Sharing (FSS). First, we construct an optimized comparison protocol to improve the efficiency via reducing the invocation of FSS evaluations. Second, we devise an efficient and privacy-enhanced division protocol without revealing the range of divisors, which utilizes the above comparison protocol and more importantly new designed FSS-based secure range and digital decomposition protocols. Besides, we further reduce the overhead of linear operations by employing lightweight pseudorandom function-based Beaver's triple techniques. Building on the above efficient components, we implement the PriVDT framework and evaluate it on 5 real-world datasets on both LAN and WAN. Experimental results show that the end-to-end runtime of PriVDT outperforms the prior art by $42 \sim 510\times$ on LAN and $16 \sim 70\times$ on WAN. Moreover, PriVDT provides comparable accuracy to the non-private setting.**

***Index Terms*—Vertical decision trees, Secure two-party computation, Privacy protection.**

I. Introduction

As an efficient and interpretable machine learning algorithm, decision trees (DTs) have been used in various real-world applications, e.g., finance risk management [1], medical diagnosis [2], and stock trading [3]. In reality, the construction and practical application of DTs meet two challenges. First, the training data are *vertically distributed* where parties hold disjoint features of the same samples. Second, the training data are also *privacy-sensitive*. For instance, in the financial risk management task, the judicial and loan information of individuals are usually held by courts and banks, respectively, while these private individual information are not allowed to directly disclose [4], [5] due to current strict policies such as General Data Protection Regulation (GDPR). Driven by these challenges, the privacy-preserving vertical DTs has been proposed as an emerging paradigm [1], [6].

Existing efforts that explore privacy-preserving vertical DTs can be classified into two categories. (1) *Leakage tolerance*. Several methods compromise some privacy guarantees in exchange for lower cryptographic overhead, e.g., samples' label [6], [7], the best split of internal nodes [8], [9], evaluation paths [10], [1] (refer to Section II for details). However, such leakages run counter to the privacy-preserving requirement. (2) *Zero privacy leakage*. Wu et al. recently proposed Pivot [11], the first privacy-preserving solution for vertical DTs without disclosing any sensitive information. In constructing DTs, Pivot exploits the *gini impurity gain* [12] as the metric to find the best split of tree nodes. The evaluation consists of linear operations, division and comparison (refer to Section IV-A for details). For privacy protection, Pivot technically uses the Paillier homomorphic encryption (HE) [13] for linear operations, and the secret sharing (SS) technique (more precisely, the SPDZ framework [14]) for comparison and division.

Despite such desirable privacy guarantees, an intrinsic problem in Pivot is its prohibitively expensive overhead, largely because (1) SS-based protocols for comparison and division rely on heavy bit decomposition and then bit-wise evaluation, which introduces high communication over multiple rounds, and (2) Pivot needs expensive HE operations and conversions between HE ciphertexts and secret-shared values. For example, it requires $3k+2$ homomorphic scalar multiplications in each tree node and $4k$ conversions, where k is the number of classes. In summary, these inappropriate usages cause huge communication and computation overhead, and reducing the cryptographic overhead is the main focus of this work.

In this paper, we present PriVDT, an efficient two-party cryptographic framework for vertical decision trees training and inference. The two-party setting is reasonable for real-world applications [15] and has been widely employed in privacy-preserving machine learning [16], [17]. Specifically, PriVDT is built upon the offline/online paradigm similar as Pivot [11], and employs several new building blocks to provide improved efficiency especially in the online phase. First, exploiting an advanced cryptographic primitive, Function Secret Sharing (FSS) [18], we present an efficient comparison protocol for the choice of the best split. The main challenge is that directly using the general FSS scheme [19] leads to a high evaluation overhead, since it requires two FSS invocations to handle the wrap around problem illustrated in Section V-A. We address this by providing a novel theoretical analysis, which shows that the probability of incurring the wrap around

Hanxiao Chen (This work was done at NTU as a visiting student), Hongwei Li (Corresponding author), Yingzhe Wang and Meng Hao are with School of Computer Science and Engineering, University of Electronic Science and Technology of China, China. (email: hanxiao.chen@std.uestc.edu.cn, hongweili@uestc.edu.cn, yingzhewang@std.uestc.edu.cn, menghao@std.uestc.edu.cn)

Guowen Xu and Tianwei Zhang are with School of Computer Science and Engineering, Nanyang Technological University, Singapore. (email: guowen.xu@ntu.edu.sg, tianwei.zhang@ntu.edu.sg)

problem is negligible with appropriate parameter settings even though we only invoke one FSS evaluation. This achieves approximately $2\times$ reduction in the online runtime compared to the most efficient FSS scheme [19], while resulting in a slight accuracy loss (less than 0.6%) in the training of trees. For communication, our protocol only requires one communication round with 2 ring elements.

Second, we devise an efficient and privacy-enhanced division protocol building on the iterated Goldschmidt's paradigm [20]. Our protocol takes inspiration from [21], but the protocol in [21] reveals the range of divisors, which could cause the leakage of split statistics in the construction of DTs. We solve this privacy leakage issue by integrating the above comparison protocol and more importantly designing new FSS-based secure range and digital decomposition protocols. Our new insight is to decompose the divisor into sub-strings and hence evaluate the range on values with smaller bitlength while hiding intermediate values. As a result, our division protocol achieves an order of magnitude improvement over the alternative in prior works [11], [22] while providing rigorous security guarantees. It is worth noting that our comparison and division protocols can be used in other privacy-critical applications, which may be of independent interest. Besides, we adopt lightweight additive secret sharing primitives to improve the evaluation overhead of linear operations, i.e., addition and multiplication, while similar as [17], we utilize PRFs to generate correlated randomness to further reduce the communication cost.

We give a formal security proof and concrete complexity analysis for the designed building blocks, and implement them to validate the efficiency advantages. We emphasize that our comparison and division protocols empirically are orders of magnitude better than the alternatives of Pivot, both in terms of online runtime and communication. Moreover, we evaluate the `PriVDT` framework end-to-end on five real-world datasets under both the LAN and WAN network setting. Experimental results show that the secure training of `PriVDT` achieves $102 \sim 863\times$ and $62 \sim 91\times$ online runtime improvements compared with Pivot on LAN and WAN, respectively. For the secure inference, $11 \sim 18\times$ and $14 \sim 24\times$ improvements are realized on LAN and WAN, respectively. Besides, `PriVDT` can be directly extended to deal with continuous attributes via an extra discretization process. We evaluate our framework on three datasets and compare it with the state-of-the-art work [23] that is customized for continuous data. Despite directly extending to such a specific setting, we still obtain up to $3.9\times$ runtime improvement and comparable accuracy over [23].

In summary, our key contributions are listed as follows:

- We propose `PriVDT`, an efficient two-party cryptographic framework for vertical DT training and inference.
- We design several new FSS-based protocols including comparison and division as the main building blocks to improve the online efficiency of `PriVDT`.
- Extensive experiments show that `PriVDT` outperforms prior art by up to orders of magnitude while realizing comparable accuracy to the non-private setting.

The rest of this paper is organized as follows. In Section II, we review the latest related works. In Section III, we discuss the system and threat models, as well as the design goals. In Section IV, we introduce the background of DTs and cryptographic preliminaries. Then, we design efficient supporting protocols in Section V. Section VI details our complete `PriVDT` framework. The performance evaluation is given in Section VII. We discuss the possible extension in Section VIII and conclude this paper in Section IX.

II. Related Works

Privacy-preserving collaborative learning of decision trees has received much attention in the scenarios where the data are scarce and distributed among multiple parties. In this learning mode, the global dataset can be partitioned in two different ways. (1) *Vertical partitioning* [24], [25]: parties hold the same samples with different features. An example of the financial risk management task is given in Section I. (2) *Horizontal partitioning* [26], [27]: different parties possess their individual samples but sharing the same features. For example, two regional banks may have different user groups from their respective regions. However, their business is very similar, so the feature spaces are the same. In this paper, we focus on the vertical partitioning setting.

Many prior works have explored the privacy-preserving vertical DT learning [24], [1], [7], [25], [28], [9], [8], [10], [6]. However, as shown in Table I, these works cannot guarantee strict privacy requirements like `PriVDT`. Specifically, Du et al. [6] proposed the first DT training algorithm for vertically partitioned data, where the secure scalar product protocol is proposed to compute the *information gain* of tree nodes. However, the label column is public to all parties, which obviously violates the privacy regulations. After that, Wang et al. [7] improved the above method by using the secure intersection technology [29] and Yao's garbled circuits [30]. Nevertheless, it does not address the critical issue of label leaks. Vaidya et al. [10], [8] introduced two generalized privacy-preserving variants of the ID3 algorithm [31] for vertically partitioned data. They allow some statistics in the training and inference tasks to be revealed in plaintext, such as the available sample IDs on a tree node, which could compromise the privacy of the involved parties. Later, Vaidya et al. [9] utilized a threshold additively homomorphic cryptosystem [32] to develop protocols to implement secure random decision trees over two or more parties. However, the intermediate values are still shared with all parties. Based on these knowledge, adversaries may infer the sensitive information of parties with high probability, including the labels of samples, or whether they have similar features. Recently, SecureBoost [1] is designed to build a gradient-tree boosting model [33] by using the Paillier homomorphic encryption scheme [13]. Although SecureBoost is responsible for protecting the privacy of local features and labels from direct leakage, it shares the split information for determining the best split to the label owner, which discloses the data distribution of another party. In summary, none of above solutions can achieve the desired privacy guarantee. Recently, Wu et al. proposed Pivot [11], the first solution for privacy-preserving vertical trees to ensure that no intermediate information are leaked. However, as described in Section

I, Pivot introduces expensive cryptographic operations that suffers from heavy overhead as shown in Section VII-C, which motivates the design of PriVDT.

In addition, some works have focused on training classification trees on specific settings, such as continuous attributes [34], [23]. Different from widely used discrete attributes with finite values in the above works, continuous attributes come from an infinite set, e.g., temperature and humidity. Specifically, [34] provided a private adaptation of the C4.5 algorithm based on MP-SPDZ [14]. To efficiently compute the statistic information required in the training algorithm, [34] utilized the *sorting network* to obliviously presort the continuous attributes into sorted order. Subsequently it considered each attribute value as a candidate cut-off point, and computed the gini impurity gain for each. Although providing end-to-end privacy protection, this scheme imposes expensive computational overhead. Recently, [23] introduced an alternative strategy to pre-process continuous data without sorting of attribute values, which achieves better efficiency compared with [34]. Generally speaking, a discretization method was designed to discretize the continuous attributes, and thus, subsequent training can be performed on discretized values. We also extend PriVDT to the setting of continuous data, and the experimental results show that PriVDT significantly outperforms [23], the state-of-the-art work customized for continuous data.

Another interesting topic is the secure decision tree classification [35], [36], [37], [38]. Unlike PriVDT, prior works [35], [36], [37] adopt the client-server model, where the *server* owns a well-trained tree and the *client* wants to obtain the prediction of a query sample. Their goal is to protect the tree classifier and private samples from being leaked to each other. Specifically, [35] provided a secure decision tree classification protocol based on the additively homomorphic encryption. Instead of representing the tree as a high-degree polynomial, it proposed a novel path cost mechanism using a linear function. This solution performs better for sparse trees but the runtime grows linearly with the number of leaves. Later, [36] proposed to transform the tree classification task to a secure search problem with the searchable symmetric key encryption to provide privacy protection. However, the runtime of this solution grows exponentially with the number of the nodes. Recently, to avoid the exponential growth of communication, a sublinear secure decision tree classification protocol was proposed in [37]. It designed novel oblivious selection protocols by carefully combining paillier encryption and boolean sharing primitives, to hide the node being accessed. Benefiting from the sublinear protocol construction, this method is scalable when evaluating large trees. In addition, [38] employed two servers to jointly conduct the inference protocol, which is similar to PriVDT. During the inference phase, each node needs to be evaluated via securely comparing the threshold with the corresponding feature. [38] implements it through heavy bit decomposition operations and conversions between different rings. In contrast, PriVDT presents a more efficient evaluation method based on advanced function secret sharing (refer to Section VI-C).

TABLE I: Comparing PriVDT with existing works in terms of privacy guarantees in the training/inference phase.

Method	Training				Inference
	Label	Available sample	Split statistic	Final tree	Evaluation path
[6]	✗	✓	✓	✗	✗
[7]	✗	✓	✗	✓	✗
[8]	✓	✗	✗	✓	✗
[9]	✓	✗	✗	✓	✗
[1]	✓	✓	✗	✓	✗
[11]	✓	✓	✓	✓	✓
[23]	✓	✓	✓	✓	✓
PriVDT	✓	✓	✓	✓	✓

III. System Overview

A. System Model

We consider both private training and inference of DTs. In the training phase, two parties P_0 and P_1 want to build a DT privately with their personal samples. In our setting, the training samples are vertically partitioned between the two parties, namely that they have the ID of the same sample[1] but with different features, and the label is owned by one party (P_1) and cannot be directly shared with the other party (P_0) [1], [6], [10]. After the tree is constructed, both parties collaboratively execute the private inference on the query samples that are secret shared between the two parties. Such inference procedure guarantees that both parties learn zero knowledge about the inputs and results.

B. Threat Model

PriVDT is designed for the two-party semi-honest model [17], [41] with a third party (STP) in the offline/online setting, where STP provides correlated randomness in the offline phase. Given the correlated randomness, the two parties engage in a secure computation protocol during the online phase. In this setting, a probabilistic polynomial-time (PPT) adversary may corrupt one of the parties, meaning that the parties do not collude with each other. During the protocol execution, the adversary honestly follows the protocol specification, but tries to obtain the private information of honest parties by analyzing the corrupted party's view. We consider the static corruption strategy, where the corrupted party is fixed before starting the protocol and remains unchanged during the whole protocol execution. We utilize the standard simulation paradigm for semi-honest security.

C. Design Goals

PriVDT aims to develop efficient and privacy-preserving protocols tailored for vertical DTs without sacrificing the inference accuracy. Our design goals can be summarized as follows.

- **Training Privacy**. In the training phase, the features and labels should be protected against the other party. The final tree (including the best split and available samples of each node) should not be leaked to any party.
- **Inference Privacy**. In the inference phase, the private sample should be protected against the two parties, while

[1]Similar to prior works [39], [40], we assume the parties' samples have been matched using the well-studied *Private Set Intersection* (PSI) technique.

the evaluation path should not be leaked to any party. Both parties learn zero knowledge about the inference results.

- **Efficiency**. Cryptographic protocols should incur tolerable computation and communication overheads, which is crucial for resource-constrained and real-time scenarios.
- **Accuracy**. Compared to the non-private setting, the privacy-preserving training and inference should result in a negligible loss of the classification accuracy and hence guarantee the model usability.

IV. Preliminaries

In this section, we first detail the decision tree training algorithm, and then introduce the underlying cryptographic preliminaries that `PriVDT` utilize. Table II summarizes the frequently used notations.

TABLE II: Summary of Notations

Notation	Description
m	the number of training samples
K	the number of classes
d_k	the number of samples in class $k \in K$
z	the number of leaves in the well-trained tree
$\boldsymbol{L}$	the labels of all leaves in the well-trained tree
x / $\boldsymbol{x}$	scalar / vector
$[x]$ / $[x]^B$	additive / boolean secret share
$\boldsymbol{x} \cdot \boldsymbol{y}$	vector dot product
$\boldsymbol{x} \odot \boldsymbol{y}$	element-wise vector multiplication
$\mathsf{F}^{\mathsf{off}}$ / F^{on}	the offline / online protocols of F

A. Decision Trees

In this work, we mainly focus on building classification trees[2] recursively with binary structures. Specifically, beginning with the root, the training algorithm first selects the best split for the current node and then recurses on each of the resulting sub-trees, until some pruning conditions are satisfied (e.g., feature set is empty or tree reaches the maximum depth). The detailed description is given in Algorithm 1.

Similar as Pivot [11], we use *gini impurity* [12] as the metric to find the best split. Formally, assuming the available sample set reaching the current node is D, and F is the set of available features, given any split feature $f_j \in F$ and split value $s \in Domain(f_j)$, D can be split into two partitions D_l and D_r. The gini impurity can be represented as follows:

$$I(D) = \sum_{k \in K} p_k (1 - p_k) = 1 - \sum_{k \in K} p_k^2, \tag{1}$$

where $p_k = d_k / D$ is the probability of a randomly selected instance from D belonging to class k. Then the *gini impurity gain* of s is $g = I(D) - (\lambda_l \cdot I(D_l) + \lambda_r \cdot I(D_r))$, where $\lambda_l = |D_l|/|D|$ and $\lambda_r = |D_r|/|D|$. The split with the maximum gain is considered as the best split of the node. Note that we can further simplify g without affecting the sorting result as follows:

$$\tilde{g} = \sum_{k \in K} \frac{d_{kl}^2}{|D_l|} + \sum_{k \in K} \frac{d_{kr}^2}{|D_r|}, \tag{2}$$

where d_{kl} and d_{kr} denote the number of samples belonging to class $k \in K$ in D_l and D_r respectively.

[2]Note that our scheme can be directly extended to regression trees with minor modifications to the protocols.

Algorithm 1 Plaintext Classification Tree Training

```
Input: Feature set $F$, Sample set $D_0$.
Output: A well-trained decision tree $T$.
1:  for each node $n_i$ do
2:      if prune conditions are satisfied then
3:          Set $n_i$ as a leaf node of $T$ with majority class.
4:      else
5:          for each split $s_i \in F$ do
6:              Split $D_i$ into $D_l$ and $D_r$ according to whether the corresponding feature value is smaller than $s_i$.
7:              Compute the gini impurity gain $\tilde{g}_j$ using Eq.2.
8:          end for
9:          Determine the best split $s_i^*$ with the maximum impurity gain, where the corresponding sample partitions are $D_l^*$ and $D_r^*$.
10:         Assign available sample set $D_{2i+1} = D_l^*$ to the left child $n_{2i+1}$ and $D_{2i+2} = D_r^*$ to the right child $n_{2i+2}$.
11: end for
12: return the well-trained tree $T$
```

Notice that from Eq.2, the evaluation of impurity gains consists of multiplication, addition and division operations. The comparison operation is also needed to obtain the best split. Therefore, the main goal of `PriVDT` is to design lightweight secure protocols for comparison, division and linear operations to construct end-to-end DTs training and inference.

B. Cryptographic Primitives

1) Secret Sharing: In `PriVDT`, all private values are secret-shared between two parties. We adopt lightweight 2-out-of-2 additive sharing over the ring $\mathbb{Z}_{2^n}$ [42]. $\mathrm{Shr}(x)$ denotes the sharing algorithm that inputs x in $\mathbb{Z}_{2^n}$ and outputs two shares $[x]_0$ and $[x]_1$ satisfying $[x]_0 + [x]_1 = x$ in $\mathbb{Z}_{2^n}$. The reconstruction algorithm $\mathrm{Rec}([x]_0, [x]_1)$ takes as input the two shares and outputs $x = [x]_0 + [x]_1$ in $\mathbb{Z}_{2^n}$. When $n = 1$, we use $[x]^B$ to denote the boolean shares, i.e., $x = [x]_0^B \oplus [x]_1^B$ in $\mathbb{Z}_2$. The security guarantees that given $[x]_0$ or $[x]_1$, the value of x is perfectly hidden.

2) Oblivious Transfer: In the oblivious transfer (OT) protocol, one party (i.e., sender S) inputs two values x_0 and x_1, while the other party (i.e., receiver R) inputs a choice bit $b \in \{0, 1\}$ and want to obtain x_b without learning anything about x_{1-b} or disclosing b to the sender. In the offline protocol $\mathsf{OT}^{\mathsf{off}}$, a semi-honest third party (STP) generates random a_0, a_1 to S and $r \in \{0, 1\}$, a_r to R. In the online protocol $\mathsf{OT}^{\mathsf{on}}$, given that S's inputs m_0, m_1 and R's input a choice bit b, R computes and sends $b' = b \oplus r$ to S. S generates and sends a tuple (s_0, s_1) to R, which equals $(a_0 \oplus m_0, a_1 \oplus m_1)$ if $b' = 0$ and $(a_0 \oplus m_1, a_1 \oplus m_0)$ otherwise. Finally, R can obtain $m_b = s_r \oplus a_r$. `PriVDT` utilizes a special OT flavor, correlated OT (COT) [43], where S inputs a correlation function f and obtains a random x_0 and $x_1 = f(x_0)$. The online communication of COTs is $n + 1$ bits within 2 rounds.

3) Lookup Table: The lookup table (LUT) protocol [44] contains a truth-table $T : \{0, 1\}^\sigma \leftarrow \{0, 1\}^\delta$, which maps a σ-bit shared input $[x]$ to a δ-bit shared output $[y]$ such that $y =$

$T(x)$. In the offline protocol $\mathsf{LUT}^{\mathsf{off}}$, STP generates (T^0, r) and (T^1, s), and sends them to P_0 and P_1, respectively, such that $T^0[j] \oplus T^1[j] = T[r \oplus s \oplus j]$ for all $j \in [2^\sigma]$, where r and s are random values to permute the table T. In the online protocol $\mathsf{LUT}^{\mathsf{on}}$, P_0 and P_1 reconstruct $z = x \oplus r \oplus s$, and then learn $[y]_0 = T^0(z)$ and $[y]_1 = T^1(z)$ such that $y = T(x)$. The online communication is $2n$ bits within 1 round.

4) Function Secret Sharing: Function secret sharing (FSS) [18] works on splitting a function f into two succinct function shares such that each share does not reveal anything about the function f, but when the evaluations at a given point x are combined, the result is $f(x)$. Formally, an FSS scheme is a pair of algorithms $\mathsf{Gen}(\cdot)$ and $\mathsf{Eval}(\cdot)$ with the following syntax.

- $(k_0, k_1) \leftarrow \mathsf{Gen}(1^\kappa, f)$: Given the security parameter κ and a function f, it outputs two keys k_0 and k_1 for P_0 and P_1, respectively.
- $[f(x)]_i \leftarrow \mathsf{Eval}(k_i, x)$: Given the key k_i and public input $x \in \mathbb{Z}_{2^n}$, it outputs $[f(x)]_i$ s.t. $[f(x)]_i + [f(x)]_{1-i} = f(x)$.

We identify two FSS constructions [19], [45] as a natural fit for PriVDT: (1) Distributed Point Function (DPF) ($\mathsf{Gen}^{\bullet}_{\alpha,\beta}$, $\mathsf{Eval}^{\bullet}_{\alpha,\beta}$) that satisfies $f_{\alpha,\beta}(x) = \beta$ is $x = \alpha$ and 0 otherwise, and (2) Distributed Comparison Function (DCF) ($\mathsf{Gen}^{<}_{\alpha,\beta}$, $\mathsf{Eval}^{<}_{\alpha,\beta}$) that satisfies $f_{\alpha,\beta}(x) = \beta$ is $x < \alpha$ and 0 otherwise. Note that the input x of $\mathsf{Eval}(\cdot)$ is public, and we show how to extend FSS for private data in Section V-A.

V. Building Blocks

In this section, we detail the building blocks of PriVDT, which are divided into the offline and online phases, and maintain the invariant that parties start with secret-shares of inputs over the ring $\mathbb{Z}_{2^n}$ and end with secret-shares of outputs over the same ring. In the offline phase, similar to [17], [19] we assume an STP to generate correlated randomness. To further improve the communication efficiency, the PRF seeds, i.e., $\mathsf{seed_{t0}}$, $\mathsf{seed_{t1}}$ and $\mathsf{seed_{01}}$, are constructed between STP and P_0, STP and P_1, P_0 and P_1, respectively. Note that STP is not involved in the online process. In Section VIII, we discuss how to distribute STP in a secure two-party protocol.

A. Secure Comparison Protocol

In PriVDT, the comparison operation is used to select the maximum gini impurity gain. Algorithm 2 gives a specific comparison protocol $\mathsf{Compare}([x],[y])$ based on FSS, which outputs the shares of $z = 1\{y > x\}$. Note that the comparison protocol is executed over the secret-shared inputs rather than public values, which should be supported by our designed FSS scheme. Following [45], the key idea is to construct the FSS scheme for the *offset* function $f^{[r]}(x) = f(x + r)$, where r is randomly selected from $\mathbb{Z}_{2^n}$ and secret-shared between P_0 and P_1. In this way, P_0 and P_1 first reconstruct $x + r$ and then evaluate $f^{[r]}(x + r)$, which exactly equals to evaluating $f(x)$. Note that the offset function fails if $x+r$ wraps around[3].

[3]For example, assume $x = 10$, then $\{x > 0\} = 1$. PriVDT executes protocols over the ring $\mathbb{Z}_{2^{64}}$. Therefore, r is randomly sampled from $\mathbb{Z}_{2^{64}}$. If $r = 2^{64} - 1$, then $x + r = 10 + 2^{64} - 1 = 9 \mod 2^{64}$. As a result, $\{x + r > r\} = \{9 > 2^{64} - 1\} = 0$ in $\mathbb{Z}_{2^{64}}$, which is not equal to $\{x > 0\} = 1$. Thus, the offset function fails if $x + r$ wraps around.

[45] deals with it by invoking 2 DCFs, but we rather prove that with proper parameters, the probability of wrap around is negligible in Theorem 1. This probability is quantified as $|x|/2^n$ for example 1 in millions in our setting, and such error does not affect the model accuracy as shown in Table III. Thus, our protocol only invokes 1 DCF and introduces $2n$ communication bits within 1 round in the online phase.

Theorem 1. *For $x \in \mathbb{Z}_{2^n}$ and given that r is picked uniformly at random from $\mathbb{Z}_{2^n}$, the failure probability denoted as $P\{1\{x < 2^{n-1}\} \neq 1\{x + r \mod 2^n \geq r\}\}$ is $\frac{|x|}{2^n}$, where $|x| = x$ if x is non-negative and $|x| = 2^n - x$ otherwise. The proof refers to Appendix C.*

Algorithm 2 Secure Comparison Protocol $\mathsf{Compare}([x],[y])$

```
1: //offline
2: STP and $P_0$ generate $[r]_0$ using PRFs with the seed $\mathsf{seed_{t0}}$.
3: STP samples $r \in \mathbb{Z}_{2^n}$ and sends $[r]_1 = r - [r]_0$ to $P_1$.
4: STP evaluates $(k_0, k_1) \leftarrow \mathsf{Gen}^{<}_{r,1}$ and sends $k_i$ to $P_i$, $i \in \{0,1\}$.
5: //online
6: $P_i$ sends $[y]_i - [x]_i + [r]_i$ to $P_{1-i}$, and reconstructs $y - x + r$.
7: $P_i$ evaluates $[z]_i \leftarrow \mathsf{Eval}^{<}_{r,1}(i, k_i, y - x + r)$.
8: Return $[z]$
```

Theorem 2. *The protocol $\mathsf{Compare}([x],[y])$ securely realizes the functionality $\mathcal{F}_{\mathsf{Compare}}$ in Table IX, assuming the existence of secure protocols for PRF, FSS and multiplication procedures. The proof refers to Appendix D.*

B. Secure Division Protocol

To securely execute division operations, we design an efficient and privacy-enhanced division protocol in Algorithm 5 building on the iterated Goldschmidt's paradigm [20]. This paradigm requires a suitable initial approximation for the division result, followed by several iterations to improve upon this approximation. To determine an initial approximation, the divisor should be normalized to [0.5, 1) [46], [21]. Therefore, the range of divisor should first be obtained, which is the main hurdle for secure computation. Existing and the most efficient protocol [21] suffers from leaking the range of divisors, which could cause severe privacy leakage about split statistics in the construction of trees.

We address this issue by proposing a secure range protocol as shown in Algorithm 4 via properly applying FSS and LUT techniques, where $\mathsf{Range}([x]) = [k]$ if and only if $2^k \leqslant x < 2^{k+1}$. The new insight here is to reduce the range evaluation into smaller bitlength. Specifically, we first decompose n-bit integer $[x]$ into $d = n/c$ sub-strings $[x_{d-1}], \cdots, [x_0]$ of c-bits. Then, we compute the range of each sub-string $[x_j]$ by taking into account its position j in x. Then $\mathsf{Range}([x]) = \mathsf{Range}([x_j]) + j \cdot c$ if $x_j \neq 0$ and $x_i = 0$ for all $i > j$. Specifically, in Algorithm 4, for $j \in [d]$, we introduce the LUT T_j that takes as input $[y_j]$ and outputs $[k_j]$ such that $2^{k_j - jc} \leq y_j < 2^{k_j - jc + 1}$. Next the parties compute $[z_j] = 1\{y_i = 0\}$ via evaluating the DPF. Then they have $e_j = k_j \cdot (1 \oplus z_j) \cdot$

$\prod_{m>j} z_m$, where $e_j = k_j$ if $y_j \neq 0$ and $y_m = 0$ for all $m > j$ and 0 otherwise. Note that at most one e_j is non-zero and $\mathsf{Range}([x]) = [k] = \sum_{j=0}^{d-1}[e_j]$.

To decompose $[x]$ to d sub-strings, we design an FSS-based digit decomposition protocol as shown in Algorithm 3, which takes $[x]$ as input and outputs $\{[x_j]\}_{j\in[d]}$, where $x = x_{j-1}||\cdots||x_0$ and the bitwidth of each x_j is $c = n/d$. To obtain $[x_j]$ for $j \in [d]$, the parties need to compute the carry of the lower sub-string into this sub-string. Let $[X]_{j,i} = [x_j]_i||\cdots||[x_0]_i$ for party P_i, we have $x_j = [x_j]_0 + [x_j]_1 + c_j$ where $c_j = [X]_{j-1,0} + [X]_{j-1,1} \geq 2^{jc}$.

With the above design, our division protocol communicates $(8n+15)(d-1) + n(2d+2n+35) + 2$ bits in the online phase. Another variant to securely compute division is the fixed-point division evaluation such as [22]. With specific modifications, [22] can be extended to the offline/online setting with $6n^2$ online communication. In our implementation with $n = 64$ and $d = 8$, the division protocol in `PriVDT` is 2.3× communication improvement compared with [22].

Algorithm 3 Digit Decomposition Protocol $\mathsf{DigDec}([x])$

```
1: //offline
2: $P_0$, $P_1$ and STP generate $d-1$ boolean Beaver's triples using Alg.6 and $d-1$ COTs using $\mathsf{OT}^{\mathsf{off}}$ in Section IV-B2.
3: for $j = 0$ to $d-2$ do
4:     $P_0$, $P_1$ and STP evaluate the offline $\mathsf{Compare}(\cdot)$.
5:     STP invokes $(k_{j,0}, k_{j,1}) \leftarrow \mathsf{Gen}^{\bullet}_{r,2^n+r-1}$ and sends $k_{j,i}$ to $P_i$.
6: end for
7: //online
8: $P_i$ parses $[x]_i$ as $[x_{d-1}]_i||\cdots||[x_0]_i$.
9: for $j = 0$ to $d-2$ do
10:     $P_0$ and $P_1$ learn $[z]_i = \mathsf{Compare}([[x_j]_0 + [x_j]_1], [2^n])$.
11:     $P_i$ evaluates $[e_j]^B_i \leftarrow \mathsf{Eval}^{\bullet}_{r,2^n+r-1}(i, k_{j,i}, x_j + r)$.
12: end for
13: $P_i$ initiates $[u_0]^B_i = 0$ and $[\delta_0] = [x_0]_i$.
14: for $j = 1$ to $d-1$ do
15:     $P_0$, $P_1$ compute $[\omega_{j-1}]^B = [u_{j-1}]^B \wedge [e_{j-1}]^B$ using the boolean triple.
16:     $P_i$ computes $[u_j]^B = [\omega_{j-1}]^B_i \oplus [z_{j-1}]^B_i$ locally.
17:     $P_0$, $P_1$ learn $[u_j]$ using $\mathsf{OT}^{\mathsf{on}}$ in Section IV-B2 with input $[u_j]^B$.
18:     $P_i$ computes $[x_j]_i = [x_j]_i + [u_j]_i$ locally.
19: end for
20: Return $[x_{d-1}]||\cdots||[x_0]$
```

Theorem 3. *The protocol* $\mathsf{DigDec}([x])$ *securely realizes the functionality* $\mathcal{F}_{\mathsf{DigDec}}$ *defined in Table IX, assuming the existence of secure protocols for comparison, FSS, multiplication and OT procedures. The proof refers to Appendix E.*

Theorem 4. *The protocol* $\mathsf{Range}([x])$ *securely realizes the functionality* $\mathcal{F}_{\mathsf{Range}}$ *defined in Table IX, assuming the existence of secure protocols for digit decomposition, OT, LUT, multiplication and FSS procedures. The proof refers to Appendix F.*

Algorithm 4 Secure Range Protocol $\mathsf{Range}([x])$

```
1: //offline
2: $P_0$, $P_1$ and STP invoke the offline $\mathsf{DigDec}([x])$ in Alg.3.
3: $P_0$, $P_1$ and STP generate $2(d-1)$ boolean Beaver's triples using Alg.6 and $2d$ COTs using $\mathsf{OT}^{\mathsf{off}}$ in Section IV-B2.
4: for $j = 0$ to $d-1$ do
5:     STP calls $(k_{j,0}, k_{j,1}) \leftarrow \mathrm{Gen}^{\bullet}_{r,1}$ and sends $k_{j,i}$ to $P_i$.
6:     Both $P_i$, $i \in \{0,1\}$ generate the LUT $\mathsf{T_j}^i$ using $\mathsf{LUT}^{\mathsf{off}}$ in Section IV-B3 to map $c$-bit input $a$ to $\log n$-bit output $b$ such that $2^{b-jc} \leq a < 2^{b-jc+1}$.
7: end for
8: //online
9: $P_0$, $P_1$ invoke the online $\mathsf{DigDec}([x])$ and learn $[x_j]$, $j \in [d]$, s.t., $x = x_0\|x_1\|\cdots\|x_{d-1}$.
10: for $j = 0$ to $d-1$ do
11:     $P_i$ evaluates $\mathsf{T_j}^i$ with $[x_j]$ using $\mathsf{LUT}^{\mathsf{on}}$ in Section IV-B3 and learns $[k_j]^i$ such that $2^{k_j-jc} \leq x < 2^{k_j-jc+1}$.
12:     $P_i$ evaluates $[z_j]^B_i \leftarrow \mathsf{Eval}^{\bullet}_{r,1}(i, k_{j,i}, x_j + r)$.
13:     $P_i$ computes $[z'_j]^B_i = i \oplus [z_j]^B_i$.
14: end for
15: $P_0$, $P_1$ compute $[e_{d-1}] = [k_{d-1}][z'_{d-1}]^B$ with 2 COTs using $\mathsf{OT}^{\mathsf{on}}$ in Section IV-B2, and sets $[\omega_{d-1}]^B = [1]$.
16: for $j = d-2$ to 0 do
17:     $P_0$, $P_1$ compute $[\omega_j]^B = [\omega_{j+1}]^B \wedge [z_{j+1}]^B$ and $[\omega'_j]^B = [\omega_j]^B \wedge [z'_j]^B$ using the boolean Beaver's triples.
18:     $P_0$, $P_1$ learn $[e_j] = [k_j][\omega'_j]^B$ with 2 COTs using $\mathsf{OT}^{\mathsf{on}}$.
19: end for
20: Return $[k] = \sum_{j=0}^{d-1}[e_j]$
```

Theorem 5. *The protocol* $\mathsf{Div}([x],[y])$ *securely realizes the functionality* $\mathcal{F}_{\mathsf{DIV}}$ *defined in Table IX, assuming the existence of secure protocols for range, LUT, OT and multiplication procedures. The proof refers to Appendix G*

C. Secure Multiplication Protocol

Similar as Chameleon [17], we evaluate multiplication operations using the Beaver triple technique [42], and further reduce the communication cost using PRFs. In `PriVDT`, we consider two variants to compute $z = xy$. The first one is that $[x]$ and $[y]$ are secret-shared between P_0 and P_1, i.e., $\mathsf{SMul}([x],[y])$ described in Algorithm 6. Compared to prior alternatives in the private decision tree works [11], [16], our STP-based multiplication protocol with PRFs just needs to communicate $[c]_1$ from STP to P_1 in the offline phase, achieving a 5× reduction in communication. The second variant is that x and y are owned by P_0 and P_1 respectively, i.e., $\mathsf{Mul}(x,y)$ described in Algorithm 7. In the online phase, $\mathsf{Mul}(x,y)$ requires one communication round in which each party sends just n bits.

Remark. In our fixed-point representations, to prevent values from overflowing due to the multiplication operations, we use the truncation technique from [16], in line with several existing methods [17], [47], [48]. This technique simply truncates the extra LSBs of fixed-point values, theoretically albeit at the

Algorithm 5 Secure Division Protocol $\mathsf{Div}([x],[y])$

```
1: //offline
2: $P_0$, $P_1$ and STP invoke the offline $\mathsf{Range}([y])$.
3: $P_0$, $P_1$ and STP generates $2t+2$ Beaver's triples using Alg.6 and $n$ COTs using $\mathsf{OT}^{\mathsf{off}}$ in Section IV-B2.
4: Both $P_i$, $i \in \{0,1\}$ generate the LUT $\mathsf{T}^i$ using $\mathsf{LUT}^{\mathsf{off}}$ in Section IV-B3 to map $\log n$-bit input $a$ to $n$-bit output $\{b_j\}$, $j \in [n]$ such that $b_j = 1$ if $j = a$ and 0 otherwise.
5: //online
6: $P_0$, $P_1$ invoke the online $\mathsf{Range}([y])$ and obtain $[k]$.
7: $P_i$ evaluates $\mathsf{T}^i$ with $[k]$ using $\mathsf{LUT}^{\mathsf{on}}$ in Section IV-B3 and learns $\{[k_j]^B\}$, $j \in [n]$.
8: $P_0$, $P_1$ invoke $\mathsf{OT}^{\mathsf{on}}$ with $[k_j]^B$, $j \in [n]$, and learn $[k_j]$.
9: $P_i$ sets $d_j = 2^{n-1-j}$ for $j \in [n]$ and computes $[k] = \sum_{j=0}^{n-1} d_j \cdot [k_j]_i$.
10: $P_0$, $P_1$ compute $[\tilde{y}] = [y][k]$ using Beaver's triples.
11: $P_i$ computes $[\omega_0]_i = 2.9142 - 2[\tilde{y}]_i$.
12: $P_0$, $P_1$ compute $[\epsilon_0] = 1 - [\tilde{y}][\omega_0]$, $[\epsilon_1] = [\omega_0][\omega_0]$ and $[z] = [\epsilon_0](1+[\epsilon_0])(1+[\epsilon_1])$ using Beaver's triples.
13: $P_i$ sets $d'_j = 2^{j-n+1}$ for $j \in [n]$ and computes $[k'] = \sum_{j=0}^{n-1} d'_j \cdot [k_j]_i$.
14: $P_0$, $P_1$ compute $[y'] = [\tilde{y}][k']$ and obtain $[z] = [x][y']$ using Beaver's triples.
15: Return $[z]$
```

cost of a 1-bit error. However, as shown in Table III, this error empirically introduces only a slight accuracy loss (less than 0.6%) in secure decision tree training. For completeness, we present the detailed truncation operation in Appendix B.

Algorithm 6 Secure Sharing Multiplication $\mathsf{SMul}([x],[y])$

```
1: //offline
2: STP and $P_0$ generate $[a]_0$, $[b]_0$ and $[c]_0$ using PRFs with the seed $\mathsf{seed_{t0}}$.
3: STP and $P_1$ generate $[a]_1$ and $[b]_1$ using PRFs with the seed $\mathsf{seed_{t1}}$.
4: STP computes $[c]_1 = ([a]_0+[a]_1)([b]_0+[b]_1) - [c]_0$, and then sends $[c]_1$ to $P_1$.
5: //online
6: For $i \in \{0,1\}$, $P_i$ computes $[e]_i = [x]_i - [a]_i$ and $[d]_i = [y]_i - [b]_i$, and then sends them to $P_{1-i}$.
7: $P_0$ and $P_1$ reconstruct $e = \mathrm{Rec}([e]_0,[e]_1)$ and $d = \mathrm{Rec}([d]_0,[d]_1)$.
8: For $i \in \{0,1\}$, $P_i$ computes $[z]_i = i \cdot e \cdot d + d \cdot [a]_i + e \cdot [b]_i + [c]_i$.
9: Return $[z]$.
```

Theorem 6. *The protocol* $\mathsf{SMul}([x],[y])$ *securely realizes* $\mathcal{F}_{\mathsf{SMul}}$ *in Table IX, assuming the existence of PRFs. The proof refers to Appendix H.*

Theorem 7. *The protocol* $\mathsf{Mul}(x,y)$ *securely realizes* $\mathcal{F}_{\mathsf{Mul}}$ *in Table IX, assuming the existence of PRFs. The proof refers to Appendix I.*

Algorithm 7 Secure Multiplication Protocol $\mathsf{Mul}(x,y)$

```
1: //offline
2: STP and $P_0$ generate $a$ and $[c]_0$ using PRFs with the seed $\mathsf{seed_{t0}}$.
3: STP and $P_1$ generate $b$ using PRFs with the seed $\mathsf{seed_{t1}}$.
4: STP computes $[c]_1 = ab - [c]_0$, and then sends $[c]_1$ to $P_1$.
5: //online
6: $P_0$ computes $e = x - a$ and sends $e$ to $P_1$.
7: $P_1$ computes $d = y + b$ and sends $d$ to $P_0$.
8: $P_0$ computes $[z]_0 = da - [c]_0$ and $P_1$ computes $[z]_1 = ye - [c]_1$.
9: Return $[z]$.
```

VI. PriVDT

A. Overview

The model setting in PriVDT is similar as recent works such as [11], [48], [34], [23]. Specifically, during the secure training process, the inputs are the training samples vertically distributed between the two parties. After this process completes, the split threshold of each node is secret-shared between the two parties, namely that none of the parties can obtain the well-trained tree in plaintext, which effectively guarantees the privacy of training samples and labels that may be inferred from the tree. During the secure inference process, the two parties jointly and privately perform the prediction on a secret-shared sample from a query party, such that they can not obtain the input and output information. Finally, the plaintext result can be reconstructed to the target query party. Before the secure training, the parties reach consensus on some hyper-parameters, such as security parameters k and pruning conditions. Also, the parties and STP construct the PRF seeds for the generation of correlated randomness.

Secure Tree Training. Let $[\gamma] = ([\gamma_1], \cdots, [\gamma_m])$ to indicate which sample is available on the tree node. Starting from the root with $[\gamma] = ([1], \cdots, [1])$, the parties jointly compute impurity gain of all splits and then select the best split with the maximum gain. Based on the best split, they can split the node to the left and right children with vectors $[\boldsymbol{\gamma_l}]$ and $[\boldsymbol{\gamma_r}]$ respectively. This process recurses on each node until pruning conditions are satisfied.

Secure Tree Inference. The parties can perform secure inference given a sample $\boldsymbol{x}$ on the well-trained tree. Briefly, starting from the root with the marker $[mk] = [1]$, the parties recursively compute the markers of its children by comparing the corresponding feature value of $\boldsymbol{x}$ with the split threshold, until all leaves are reached. The markers of leaves are assigned to a vector $[\boldsymbol{\mu}]$. After the inference, only one element in $[\boldsymbol{\mu}]$ is 1, specifying the real inference path on $\boldsymbol{x}$. Then the inference result $[y]$ can be computed via $[y] = [\boldsymbol{L}][\boldsymbol{\mu}]$, where $\boldsymbol{L}$ includes the labels of all leaves.

B. Secure Tree Training

In the following, we first give an example of the training phase in plaintext, followed by the cryptographic evaluation. Consider an example in Figure 1, where P_1 owns labels

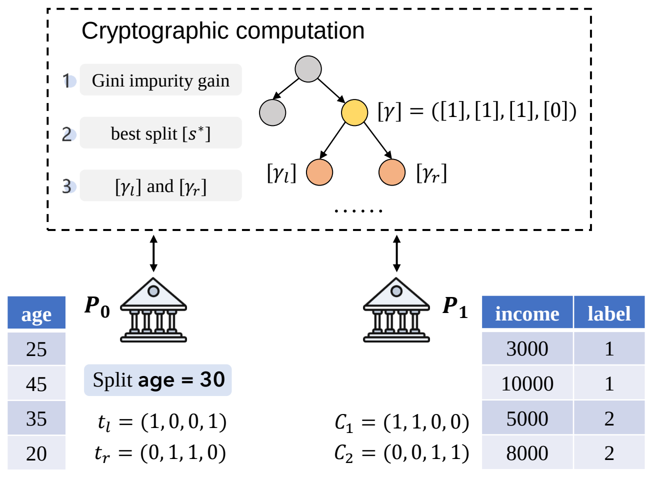


Fig. 1: **Secure tree training.** Given $[\gamma]$ of the node, the training phase includes three step: 1) compute the impurity gain of each split, 2) obtain the best split, 3) update $[\gamma_l]$ and $[\gamma_r]$ for the left and right children.

with two classes 1, 2 and each party holds one feature, i.e., age or income. The parties aim to split a node with $[\gamma] = ([1],[1],[1],[0])$, i.e., samples 1, 2, 3 are available on this node. First, P_1 generates $\boldsymbol{C_1} = (1,1,0,0)$ to indicate whether the sample belongs to class 1 (resp. $\boldsymbol{C_2}$). Then they compute $[\boldsymbol{\delta_1}] = \boldsymbol{C_1} \odot [\gamma] = ([1],[1],[0],[0])$ to obtain the available samples belonging to class 1 (resp. $[\boldsymbol{\delta_2}]$). Suppose that P_0 considers a split "age = 30". P_0 first divides the samples into two partitions based on whether the age is larger than 30, obtaining $\boldsymbol{t_l} = (1,0,0,1)$ for the left child and $\boldsymbol{t_r} = (0,1,1,0)$ for the right child. On the left child, both parties compute $[d_{l1}] = \boldsymbol{t_l} \cdot [\boldsymbol{\delta_1}] = [1]$ that equals the number of available samples belonging to class 1 (resp. $[d_{l2}]$), and $[|D_l|] = \boldsymbol{t_l} \cdot [\gamma] = [1]$, i.e., the total number of available samples. The same operations are performed on the right child. After that, they can compute the impurity gain shown in Eq.2 based on the above statistics and obtain the best split $[s^*]$, and next update $[\boldsymbol{\gamma_l}]$ (resp. $[\boldsymbol{\gamma_r}]$) on the left (resp. right) child.

Algorithm 8 gives the specific operations of one node with $[\gamma]$. If pruning conditions are satisfied, this node will be returned as a leaf. Otherwise, it will derive left and right children based on the gini impurity gain. Specifically, (1) **Leave label computation (line 1-10)**: For each class $k \in K$, the label owner P_1 can generate $\boldsymbol{C_k}$ locally. Then the parties jointly learn $[\boldsymbol{\delta_k}] = \boldsymbol{C_k} \odot ([\gamma]_0 + [\gamma]_1) = \mathsf{Mul}(\boldsymbol{C_k}, [\gamma]_0) + \boldsymbol{C_k} \odot [\gamma]_1$, followed by summing all elements in $[\boldsymbol{\delta_k}]$ to obtain the number of samples belonging to each class k. Next, the parties compute the label $[k^*]$ of this node, which has the largest m_k, via comparing them one by one. If pruning conditions are not met, the following steps will be performed. (2) **Gini Impurity Evaluation (line 12-21)**: Given a split s_j of feature f_b, the feature owner can construct $\boldsymbol{t_l}$ and $\boldsymbol{t_r}$ locally. Based on such information, the parties can jointly evaluate $[|D_l|]$, $[|D_r|]$, $[d_{kl}^2]$ and $[d_{kr}^2]$, by invoking our multiplication protocol. To the end, with the above secret-shared statistics, the parties can obtain the impurity gain $[\tilde{g}_j]$ shown in Eq.2 by invoking our division protocol. (3) **Best Split Selection (line 22-35)**: After obtaining the impurity gain of all splits, the parties attempt to select the best split with the maximum impurity gain. To reduce multiplication invocations, our strategy is to first select the best split of each feature as a candidate split, e.g., $[s_b^*]$ for feature f_b. Subsequently, we obtain the best split $[s^*]$ of the current node, as well as the feature $[f^*]$ to which that this split belongs. (4) **Model Update (line 36-38)**: By now, the parties can update $[\boldsymbol{\gamma_l}]$ and $[\boldsymbol{\gamma_r}]$ for the current node's children. They first learn $[\boldsymbol{\Psi_r}]$ indicating the sample index that the feature value belonging to f^* is smaller than $[s^*]$. Next they jointly invoke our multiplication protocol to obtain $[\gamma_r]$ for the right child, and compute $[\gamma_l] = [\gamma] - [\gamma_r]$ for the left child.

Algorithm 8 Secure Tree Training

```
1:  $[\boldsymbol{\delta_k}] \leftarrow \mathsf{Mul}(\boldsymbol{C_k}, [\gamma]_0) + \boldsymbol{C_k} \odot [\gamma]_1$ for $k \in K$
2:  if pruning conditions are satisfied then
3:      $[m_k] = \sum_i [\boldsymbol{\delta_k}][i]$ for each $k \in K$
4:      $[k^*] = [1]$, $[m^*] = [m_1]$
5:      for $i = 2$ to $|K|$ do
6:          $[\Psi] = \mathsf{Compare}([m^*], [m_i])$
7:          $[k^*] = \mathsf{SMul}([\Psi], i - [k^*]) + [k^*]$
8:          $[m^*] = \mathsf{SMul}([\Psi], [m_i] - [m^*]) + [m^*]$
9:      end for
10:     Return $[l^*]$ // the label with majority class
11: for $f_b \in F$ do
12:     for $s_j \in f_b$ do
13:         $\boldsymbol{t_l} = (1\{t_1 \le s_j\}, \cdots, 1\{t_m \le s_j\})$, $\boldsymbol{t_r} = \boldsymbol{1} - \boldsymbol{t_l}$
14:         $[|D_l|] \leftarrow \mathsf{Mul}(\boldsymbol{t_l}, [\gamma]_0) + \boldsymbol{t_l} \cdot [\gamma]_1$ (resp. $[|D_r|]$)
15:         for $k \in K$ do
16:             $[d_{kl}] \leftarrow \mathsf{Mul}(\boldsymbol{t_l}, [\boldsymbol{\delta_k}]_0) + \boldsymbol{t_l} \cdot [\boldsymbol{\delta_k}]_1$ (resp. $[d_{kr}]$)
17:             $[d_{kl}^2] \leftarrow \mathsf{SMul}([d_{kl}], [d_{kl}])$ (resp. $[d_{kr}^2]$)
18:             $[\rho_{lk}] \leftarrow \mathsf{Div}([d_{lk}^2], [|D_l|])$ (resp. $[\rho_{rk}]$)
19:         end for
20:         $[\tilde{g}_j] = \sum_{k \in K}([\rho_{lk}] + [\rho_{rk}])$  //gini impurity gain
21:     end for
22:     $[s_b^*] = [s_0]$, $[g_b^*] = [\tilde{g}_0]$  //$s_b^*$ is the best split of $f_b$
23:     for $s_j \in f_b$ do
24:         $[\Psi] = \mathsf{Compare}([\tilde{g}_j], [g_b^*])$
25:         $[g_b^*] = \mathsf{SMul}([\Psi], [\tilde{g}_j] - [g_b^*]) + [g_b^*]$
26:         $[s_b^*] = \mathsf{SMul}([\Psi], [s_j] - [s_b^*]) + [s_b^*]$
27:     end for
28: end for
29: $[g^*] = [g_0^*]$, $[s^*] = [s_0^*]$, $[f^*] = [f_0]$ //$s^*$ is the best split
30: for $b = 1, \cdots, |F| - 1$ do
31:     $[\Psi] = \mathsf{Compare}([g_b^*], [g^*])$
32:     $[g^*] = \mathsf{SMul}([\Psi], [g_i^*] - [g^*]) + [g^*]$
33:     $[s^*] = \mathsf{SMul}([\Psi], [s_i^*] - [s^*]) + [s^*]$
34:     $[f^*] = \mathsf{SMul}([\Psi], [f_i] - [f^*]) + [f^*]$
35: end for
36: $f^* = \mathsf{Rec}([f^*]_0, [f^*]_1)$
37: $[\Psi_r[j]] = \mathsf{Compare}([fv_j], [s^*])$ for each $fv_j \in f^*$
38: $[\gamma_r] = \mathsf{SMul}([\boldsymbol{\Psi_r}], [\gamma])$, $[\gamma_l] = [\gamma] - [\gamma_r]$
```

Remark. Note that like Pivot [11], the feature f^* is public in `PriVDT`. Also, the parties can choose to hide f^* with minor modifications to the protocols. Specifically, let $\boldsymbol{T}^{m \times n_s} = (\boldsymbol{t}_1, \cdots, \boldsymbol{t}_{n_s})$ (n_s is the total number of splits) be the split indicator matrix, where $\boldsymbol{t}_j$ indicates the j-th split. Given $\boldsymbol{T}$

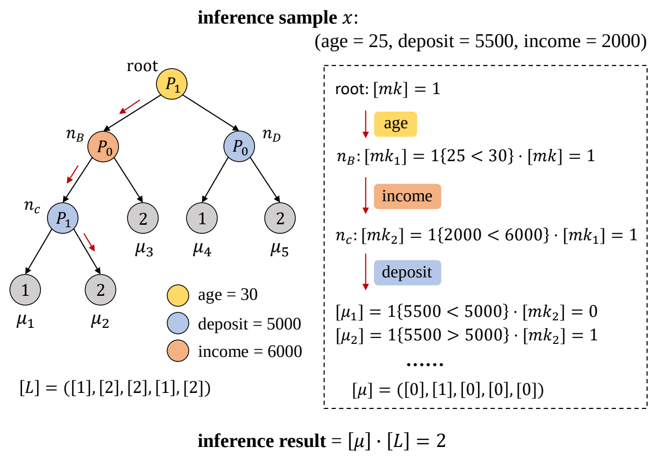

inference sample x:
(age = 25, deposit = 5500, income = 2000)
root
P_1
n_B
P_0
n_D
P_0
n_c
P_1
2
1
2
μ_3
μ_4
μ_5
1
2
μ_1
μ_2
age = 30
deposit = 5000
income = 6000
$[L] = ([1],[2],[2],[1],[2])$
root: $[mk] = 1$
age
n_B: $[mk_1] = 1\{25 < 30\} \cdot [mk] = 1$
income
n_c: $[mk_2] = 1\{2000 < 6000\} \cdot [mk_1] = 1$
deposit
$[\mu_1] = 1\{5500 < 5000\} \cdot [mk_2] = 0$
$[\mu_2] = 1\{5500 > 5000\} \cdot [mk_2] = 1$
......
$[\mu] = ([0],[1],[0],[0],[0])$
inference result = $[\mu] \cdot [L] = 2$

Fig. 2: **Secure tree inference.** The key insight is to recursively compute the marker of each node until leaves are reached.

and $[\boldsymbol{V}] = ([v_1], \cdots, [v_{n_s}])$ such that $[v_j] = [1]$ if $j = s^*$ and 0 otherwise[4], then $[\boldsymbol{t}_{s^*}] = \boldsymbol{T} \cdot [\boldsymbol{V}]$. After that we can update $[\gamma_l] = \mathsf{SMul}([\boldsymbol{t}_{s^*}], [\gamma])$. The same manner applies to $[\gamma_l]$. This way can eliminate the leakage of features, but damage the interpretability. In fact, there is a trade-off between privacy and interpretability for the tree-based models.

C. Secure Tree Inference

Given a secret-shared well-trained tree, the parties can perform secure inference on a sample $\boldsymbol{x}$. Consider an example in Figure 2, where the label vector of the tree is $[\boldsymbol{L}] = ([1],[2],[2],[1],[2])$. The split threshold of the root with $[mk] = [1]$ is "age = [30]", and the age of $\boldsymbol{x}$ is [25]. Thus $[mk_1] = [1\{25 < 30\}] \cdot [mk] = [1]$ is assigned to the right child n_B and $[1\{25 > 30\}] \cdot [mk] = [0]$ to its left child n_D. The parties can recursively compute the marker of each node in the same way, i.e., comparing the corresponding feature value of $\boldsymbol{x}$ with the split threshold of current node, and then invoking multiplication to obtain the marker. To the end, the parties learn $[\boldsymbol{\mu}] = ([\mu_1],[\mu_2],[\mu_3],[\mu_4],[\mu_5]) = ([0],[1],[0],[0],[0])$, and the secret-shared inference result is $[\boldsymbol{\mu}] \cdot [\boldsymbol{L}] = [2]$.

Formally, the secure inference scheme is given in algorithm 9. Given that each internal node n_j is associated with one feature f_j with the best split $[s_j]$, this protocol is described as follows. For n_j with the marker $[mk_j]$, the parties jointly compare $[s_j]$ with the corresponding feature value $[x_j]$ of the sample $\boldsymbol{x}$ to obtain $[\Psi_j]$. Then, the parties jointly compute the marker $[mk_{2*j+2}] = [mk_j][\Psi_j]$ of its right child, and $[mk_{2*j+1}]$ for its left child via invoking our multiplication protocol. To the end, the parties obtain $[\boldsymbol{\mu}] \in \{0,1\}^z$ where each element denotes the marker of a leaf and only one element is 1. Finally, the inference result is computed via $[y] = [\boldsymbol{L}][\boldsymbol{\mu}]$.

D. Security Analysis

Theorem 8. *The scheme in Algorithm 8 is a secure tree training scheme against semi-honest adversaries, assuming the existence of secure multiplication, comparison and division procedures. The proof refers to Appendix J.*

[4]It can be obtained via the well-studied Private Information Retrieval (PIR) technique [49], [19].

Algorithm 9 Secure Tree Inference

```
1: for $j = 1, \cdots, z-1$ do
2:     If $j == 1$, then $[mk_j] = [1]$.
3:     $[\Psi_j] = \mathsf{Compare}([s_j], [x_j])$
4:     $[mk_{2*j+2}] = \mathsf{SMul}([mk_j], [\Psi_j])$
5:     $[mk_{2*j+1}] = \mathsf{SMul}([mk_j], 1 - [\Psi_j])$
6: end for
7: if $n_k$ is a leaf with the marker $[\mu_k] = [mk_k]$ then
8:     Put $[\mu_k]$ in the correct position in $[\boldsymbol{\mu}]$
9: $[y] = \mathsf{SMul}([\boldsymbol{\mu}], [L])$
```

Theorem 9. *The scheme in Algorithm 9 is a secure tree inference scheme against semi-honest adversaries, assuming the existence of secure multiplication and comparison procedures. The proof refers to Appendix J.*

E. Extend `PriVDT` to continuous attributes

Utilizing the discretization method in [23], called equal-width binning (EWB), `PriVDT` can be extended to evaluate continuous data. The main idea of EWB is to divide the range of attributes into a pre-defined number of bins with the same width. However, as shown in Section VII-D, directly utilizing this method cannot yield desirable accuracy in `PriVDT`.

We address this problem by proposing a simple but effective discretization method. In detail, we first sort the training samples D based on the continuous attribute. Subsequently, given a pre-defined number of bins P, we place the sorted samples into these P bins, each containing $\frac{D}{P}$ samples (except for the last bin). Finally the split threshold of each bin can be obtained as the maximum continuous attribute value in the corresponding bin. Notice that in our vertical setting, the discretization process can be performed locally by the parties, since the attributes are owned by either party in plaintext. The experimental results in Section VII-D show that our discretization method can achieve better accuracy in `PriVDT`.

VII. Evaluation

A. Experimental Setup

We implement `PriVDT` in C++ using the communication backend of the Porthos framework in EzPC [50]. PRF is built on the block cipher AES using the OpenSSL-AES library [51]. Besides, the FSS schemes are implemented based on the LibFSS library [52]. `PriVDT` is executed on three desktops with Intel(R) 562 Xeon(R) CPU E5-2620v4 (2.10 GHz) and 16 GB of RAM running the Ubuntu 18.4 system, each of which is the instantiation of one party (P_0, P_1 and STP). The communication overhead reported in the following contains the communication between STP and the other two parties, as well as between the two parties. The runtime comes from the computational costs of the local computation of these three entities, and the communication latency among them. For the experiments on LAN, the bandwidth is 2GBps and the echo latency is 0.3ms. On WAN, the bandwidth is 40MBps and the network delay is 40ms. We set secret-sharing protocols over the ring $\mathbb{Z}_{2^{64}}$ following existing works [22], [16], and encode inputs using a fixed-point representation with the 20-bit precision.

Datasets. We evaluate PriVDT on five real-world datasets taken from the UCI machine learning repository [53].

- *Iris dataset* is used for the pattern recognition of iris flowers, which contains 150 samples within 3 classes. Each sample has 4 features and each class refers to a type of iris plant.
- *Heart Disease dataset* aims to infer the presence or absence of heart diseases in patients, which contains 303 samples with 75 features, and the label is 0 (no presence) to 4.
- *Bank Marketing dataset* aims to predict if a client will subscribe a term deposit, which contains 4,521 samples with 17 features.
- *Credit Card dataset* is to predict if a client is credible or not, which contains 30,000 samples with 23 features.
- *Handwritten Digits dataset* contains 5,620 images of handwritten digits with 63 features. All input attributes are integers in the range $[0, 16]$ and the class label is from $[0, 9]$.

B. Model Accuracy

To validate the accuracy goal of our cryptographic framework, Table III compares the accuracy in PriVDT with the non-private baseline under different datasets. We observe that PriVDT achieves comparable accuracy performance with the non-private setting. The slight accuracy loss is possibly cause by the fixed-point presentation and the approximate division operation. These issues are unavoidable in secure multi-party computation techniques [11], [21].

TABLE III: Accuracy (%)

Dataset	Tree depth	PriVDT	Non-private
Iris	3	97.77	97.78
Heart Disease	4	81.13	81.17
Bank Marketing	4	87.96	88.17
Credit Card	4	83.74	83.84
Handwritten Digits	5	75.35	75.94

C. Comparison with Pivot [11]

In this section, we give the performance comparison between PriVDT and Pivot [11]. We run the Pivot system with the same experimental setup using their provided reference implementation [54].

Microbenchmark comparison. Table IV gives the online performance comparison of the addition and multiplication protocols in PriVDT with Pivot on LAN. Note that in PriVDT, the addition can be evaluated locally in the secret-shared values without any cryptographic computation. In contrast, in Pivot the addition is evaluated on the homomorphic ciphertext. Thus the communication of both schemes are 0 but the runtime of Pivot is more expensive. For the multiplication, although Pivot introduces zero communication, it additionally requires the communication-heavy domain conversion to be compatible with secret-shared values. Our computation overhead is more lightweight than Pivot since we just involve non-cryptographic operations. Fig.3 compares the online performance of comparison and division protocols in PriVDT with Pivot on LAN. We observe that both protocols in PriVDT are orders of magnitude better than those of Pivot.

TABLE IV: Comparing the online runtime and communication overhead of addition and multiplication operations with Pivot.

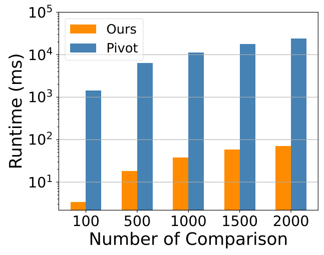
Number	Addition				Multiplication			
	Runtime (ms)		Comm. (KB)		Runtime (ms)		Comm. (KB)	
	Ours	Pivot	Ours	Pivot	Ours	Pivot	Ours	Pivot
100	0.001	0.68	0.00	0.00	0.19	0.69	3.12	0.00
500	0.001	3.28	0.00	0.00	0.33	3.37	15.62	0.00
1000	0.001	6.57	0.00	0.00	0.49	6.66	31.25	0.00
1500	0.001	10.09	0.00	0.00	0.66	10.67	46.87	0.00
2000	0.001	13.28	0.00	0.00	0.79	13.48	62.50	0.00

End-to-end comparison. Table V gives the end-to-end comparison on runtime with Pivot, where the tree depth is 3, and the maximum number of splits is 5, 3, 5, 14 for the four datasets respectively. Note that we only report the online runtime of Pivot since the offline benchmark and the communication cost cannot be measured in Pivot. Besides, we give a theoretical comparison of the communication complexity analysis in Appendix K. The online training of PriVDT over four datasets are <4.3s on LAN. Compared to Pivot, for example, PriVDT realizes a speedup of $130\times$ on the Bank Marking dataset. Besides, PriVDT is still practical on WAN and takes <86s online runtime over four datasets, achieving at least $85\times$ boost over Credit Card dataset compared with Pivot. Note that PriVDT remains superior on the total cost, since Pivot requires communication-heavy bit decomposition and computation-expensive homomorphic operations. In addition, PriVDT can perform online inference on single sample in 2ms, which is at least $11\times$ faster on LAN and $15\times$ faster on WAN compared with Pivot.

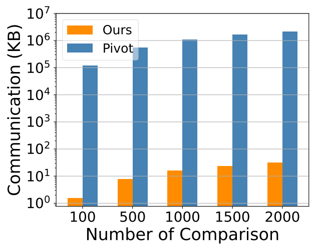
D. Comparison with [23] on continuous data

As described in Section VI-E, PriVDT can be extended to deal with continuous attributes via an extra discretization process. In Figure 4, we show the performance of PriVDT on three continuous datasets (Breast Cancer dataset (BC), ECG Heartbeat dataset (ECG), and Lower Back Pain Symptoms dataset (BACK)) and give the comparison with [23], the most advanced private DTs training over continuous data. For a fair evaluation, we use the same experimental settings as [23], where entities are connected via Gigabit Ethernet network. The tree depth is 4 for BC and BACK and 1 for ECG, and the number of bins is 5 for BC/BACK and 3 for ECG. We observe that from Figure 4(a), directly utilizing the discretization method of [23] cannot yield satisfactory accuracy in PriVDT, e.g., 86.84% and 70.96% on BC and BACK, respectively. The accuracy of ECG is up to 1 due to its simple classification task. In contrast, our discretization method detailed in Section VI-E can achieve better accuracy, e.g., 93.86% on BC and 78.91% on BACK, which is generally consistent with the results in [23]. Figure 4(b) shows the online runtime of PriVDT under different discretization methods, and the comparison with [23]. Under the similar accuracy, PriVDT achieves $2.1 \sim 3.9\times$ runtime boost compared with [23].

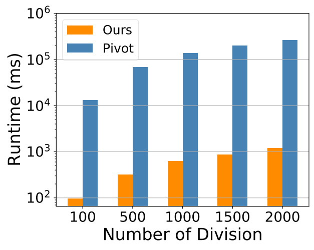
In addition, we report the detailed secure training and inference overheads of PriVDT under LAN and WAN on these three continuous datasets. As shown in Table VI, we observe that PriVDT introduces low runtime cost under LAN since our underlying protocols only contains efficient symmetric



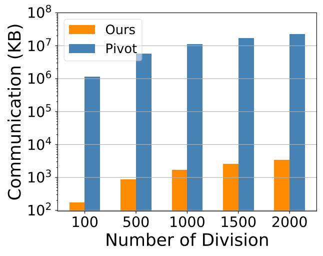
(a) Runtime of comparison



(b) Communication of comparison



(c) Runtime of division



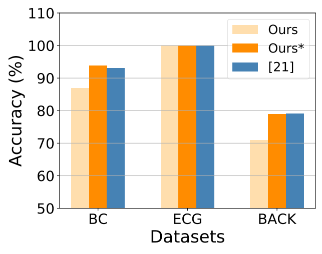
(d) Communication of division

Fig. 3: Comparing the online runtime and communication overhead of comparison and division operations with Pivot.

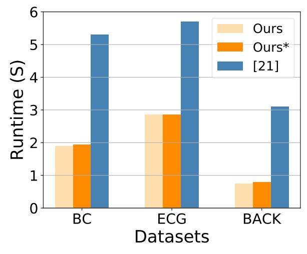
TABLE V: Comparing the runtime of `PriVDT` with Pivot [11]. Note that the offline benchmark cannot be measured in Pivot.

	Datasets	**Runtme for training (s) and inference (ms) on LAN**						**Runtime for training (s) and inference (ms) on WAN**					
		Offline		**Online**		**Total**		**Offline**		**Online**		**Total**	
		Pivot	**Ours**	**Pivot**	**Ours**	**Pivot**	**Ours**	**Pivot**	**Ours**	**Pivot**	**Ours**	**Pivot**	**Ours**
Training	Iris	-	0.36	448.94	0.52	>448.94	0.88	-	10.28	2047.06	32.96	>2047.06	43.24
	Heart Disease	-	1.24	213.48	2.08	>213.48	3.32	-	35.02	10755.81	117.16	>10755.81	152.18
	Bank Marketing	-	7.28	455.60	3.50	>455.60	10.78	-	190.98	3946.87	47.72	>3946.87	238.70
	Credit Card	-	9.14	560.97	4.28	>560.97	13.42	-	225.74	7423.59	85.54	>7423.59	311.28
Inference	Iris	-	0.80	17.64	1.45	>17.64	2.25	-	24.60	1747.59	121.18	>1747.59	145.78
	Heart Disease	-	1.09	17.51	1.46	>17.51	2.55	-	28.28	1737.80	121.04	>1737.80	149.32
	Bank Marketing	-	0.98	26.35	1.44	>26.35	3.40	-	28.32	2927.22	120.86	>2927.22	149.18
	Credit Card	-	1.02	26.68	1.44	>26.68	2.46	-	28.25	2927.81	120.94	>2927.81	149.19

cryptographic operations. However, under WAN, the runtime increases significantly, due to high network latency from multiple rounds of communication. In addition, PriVDT offloads most of the communication to the offline phase. For example, the offline communication of secure training on BACK is 62.52 MB, but the online phase only introduces 5.12 MB communication.



(a) Accuracy



(b) Runtime

Fig. 4: Comparing the accuracy and runtime of `PriVDT` with [23] on continuous datasets. Ours uses the discretization method in [23], and Ours* uses our proposed discretization method in Section VI-E.

TABLE VI: Secure training and inference performance on continuous datasets under LAN/WAN.

Datasets	**Runtime on LAN**			**Runtime on WAN**			**Communication cost**		
	Offline	**Online**	**Total**	**Offline**	**Online**	**Total**	**Offline**	**Online**	**Total**
Secure Training (Runtime: s, Communication: MB)									
BC	2.27	2.03	4.3	62.98	109.34	172.32	142.70	25.07	167.77
BACk	0.97	0.81	1.78	27.22	43.63	70.85	62.65	5.12	67.86
ECG	2.63	2.66	5.29	62.19	33.39	95.58	122.30	96.33	218.63
Secure Inference (Runtime: ms, Communication: KB)									
BC	2.16	2.10	4.26	60.46	161.30	221.76	144.49	1.67	146.16
BACK	2.13	2.07	4.20	60.67	161.36	222.04	144.49	1.67	144.16
ECG	0.14	1.22	1.37	4.04	120.50	124.54	9.63	0.14	9.77

E. *Detailed Performance of* `PriVDT`

Table VII gives the secure training performance of `PriVDT` under LAN and WAN, where the tree depth is 3, the maximum number of splits is 14, and the number of pruning samples is 5. On LAN, `PriVDT` takes <48s to train trees on all datasets. On WAN, our scheme is still efficient, which takes about 21 minutes on the complex Handwritten Digits dataset and <9 minutes on others. Longer runtime is required on WAN since the training of trees requires multiple communication rounds. For the communication, the cost of the online phase is much less than that of the offline phase. This is because the designed protocols are communication-efficient and we further move the costly cryptographic operations into the offline phase.

TABLE VII: Secure training performance on LAN/WAN.

Datasets	**Runtime (s) on LAN**			**Runtime (s) on WAN**			**Communication cost (MB)**		
	Offline	**Online**	**Total**	**Offline**	**Online**	**Total**	**Offline**	**Online**	**Total**
Iris	0.36	0.52	0.88	10.28	33.6	43.24	49.06	2.27	51.60
H-Disease	2.72	4.92	7.64	72.66	286.36	359.02	345.22	73.71	418.92
B-Marking	7.71	3.69	11.40	205.08	86.86	291.94	965.04	125.54	1091.38
C-Card	11.18	6.72	17.90	270.98	210.58	481.48	1273.96	347.46	1621.42
H-Digits	22.05	25.12	47.64	362.42	896.82	1260.04	1712.36	1468.54	3180.90

TABLE VIII: Secure inference performance on LAN/WAN.

Datasets	**Runtime (ms) on LAN**			**Runtime (ms) on WAN**			**Communication cost (KB)**		
	Offline	**Online**	**Total**	**Offline**	**Online**	**Total**	**Offline**	**Online**	**Total**
Iris	0.81	1.45	2.26	24.60	121.18	145.78	115.74	0.68	116.42
H-Disease	0.98	1.44	2.42	28.16	120.82	148.98	135.03	0.79	135.82
B-Marketing	0.99	1.44	2.43	28.19	120.83	149.02	135.03	0.79	135.82
C-Card	0.98	1.43	2.41	28.19	120.80	148.99	135.03	0.79	135.82
H-Digits	1.03	1.59	2.62	28.21	120.92	149.13	135.03	0.79	135.82

Table VIII shows the performance of the secure inference under LAN and WAN, where the online phase takes about 1.5ms and 120ms respectively with 0.8KB communication on all datasets. The inference overhead on these datasets is similar, since complete binary trees with a depth of 3 are built for these datasets (except Iris), and hence the number of cryptographic operations is roughly equal. Besides, the online communication is much lower than that in the offline phase, whereas the online runtime is the opposite. The main reason is that in the offline phase, `PriVDT` just needs one communication round to generate correlated randomness. However, the online evaluation is carried out layer-by-layer, where communication cannot be parallelized, resulting in the increased round-trip latency.

F. *Performance over Varied Parameters*

We explore the performance of `PriVDT` over varied parameters based on the Credit Card dataset. By default, the number of training data and features are $5,000$ and 23 respectively. The maximum number of splits is 14 and the tree depth is 3.

1) Training: Figures 5 and 6 give the performance of the secure training phase under various parameters.

Varying the number of training data. The runtime and communication cost grows roughly linearly as the number of training samples increases as shown in Figures 5(a) and 6(a), because PriVDT needs more multiplication operations to compute the impurity gain. In Figure 5(a), the runtime on WAN is more expensive than that on LAN due to the bandwidth and latency constraints. In the same network setting, the offline overhead is much heavier than that in the online phase. The same phenomenon also appears in Figure 6(a). Note that the communication cost is independent of the network setting.

Varying the number of features. As shown in Figure 5(b), the runtime grows with the increased features. On WAN the online runtime varies more widely because the secure computations need more communication rounds, which introduces larger latency. In contrast, no matter how the parameters are changed, only one communication round is required in the offline phase. In Figure 6(b), the communication overhead of offline and online phases roughly increases linearly with the similar trend.

Varying the maximum tree depth. The well-trained tree tends to a complete binary tree, where about $2^h - 1$ internal nodes are constructed given the depth h. Therefore, as shown in Figures 5(c) and 6(c), the runtime and communication overhead tends to increase logarithmically with the tree depth.

2) Inference: Figures 5(d) and 6(d) give the performance of the secure inference phase under different tree depths.

Varying the maximum tree depth. Results in Figure 5(d) show that the online runtime increases logarithmically with the tree depth. In other words, the runtime grows linearly with the increased intermediate nodes. This is because the communication latency is relatively low on LAN and local cryptographic operations dominate the runtime. On WAN, the communication latency becomes the main performance bottleneck. Thus the online runtime grows roughly linearly as the tree depth increases. Since the communication overhead is directly related to the number of intermediate nodes, it grows logarithmically with the tree depth in Figure 6(d).

Other parameters such as the numbers of features have no direct effect on the inference performance. As shown in Section VI-C, the inference phase in PriVDT only contains secure comparison and multiplication operations, where the number of comparisons is equal to the number of tree nodes, and the number of multiplication rounds is equal to the maximum tree depth.

VIII. Discussion

A. Distribute STP in a 2PC protocol

Similar as most privacy-preserving works based on FSS [55], [19], [45], PriVDT generates correlated randomness (especially for FSS) via a third party. However, the role of the third party can be jointly emulated by the two parties via generic two-party secure protocols such as Garbled Circuits (GCs) [30] and GMW [56] or specific techniques [57], [58]. Specifically, (1) one can use generic GCs or GMW-style protocols to generate the required correlated randomness during the offline phase. Despite desirable versatility, these protocols require to privately evaluate underlying pseudorandom generators (PRGs) during the key generation step of FSS. (2) As a customized case, [57] proposed a novel solution that offers the significant efficiency advantage since the evaluation of PRGs only takes place locally, and does not need to be securely emulated. However, it's restricted to moderate domain size and hence hard to extend to general and large cases. Thus, it's an interesting future work to design efficient and general techniques for distributing the third party in FSS techniques, and then extend our PriVDT framework into a full two-party setting without the third party.

B. Extend PriVDT to the multi-party setting

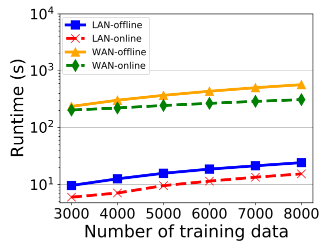
We explore the possibility of extending PriVDT to the multi-party setting, and provide technical ideas. The required building blocks and existing techniques are illustrated as follows. (1) Multi-party secret sharing: This is a trivial requirement in n-out-of-n additive secret sharing, which supports addition and multiplication operations like the 2-out-of-2 variant we used. (2) Multi-party truncation protocol: In multi-party settings where our two-party truncation protocol from [16] does not work, we can use the truncation protocol of [59] for scaling down the output after each fixed-point multiplication. This protocol slightly increases the communication complexity and requires 1 communication round. (3) Multi-party FSS: Dodis et al. [60] proposed a multi-party FSS scheme for general functions (more precisely, fixed-depth circuits) under the Learning with Errors assumption, which can be implemented by making use of multi-key FHE [61], [62]. This is sufficient for constructing the FSS protocols of comparison and division. With these building blocks, the multi-party protocol extension can be easily obtained by replacing the cryptographic operations in Algorithms 8 and 9 with multi-party protocols.

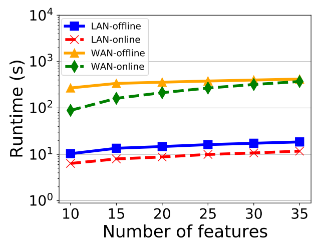
IX. Conclusion

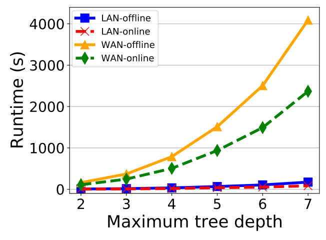
In this paper, we propose PriVDT, an efficient two-party cryptographic framework for vertical decision trees. Specifically, we develop new building blocks, including private comparison and division, based on function secret sharing. We further reduce the overhead of linear operations via employing lightweight PRF-based Beaver's triple technique. Extensive experiments show that PriVDT outperforms prior art by up to orders of magnitude with a slight accuracy loss. In the future, we will extend PriVDT to malicious settings [63], [14].

Acknowledgment

This work is supported by the National Key R&D Program of China under Grant 2022YFB3103500, the Key-Area Research and Development Program of Guangdong Province under Grant 2020B0101360001, the National Natural Science Foundation of China under Grants 62020106013 and 61972454, the Sichuan Science and Technology Program under Grants 2020JDTD0007, the Fundamental Research Funds for Chinese Central Universities under Grant ZYGX2020ZB027, and Singapore Ministry of Education (MOE) AcRF Tier 2 MOE-T2EP20121- 0006.


(a) Training: training data


(b) Training: features


(c) Training: tree depth

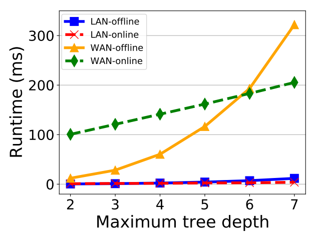
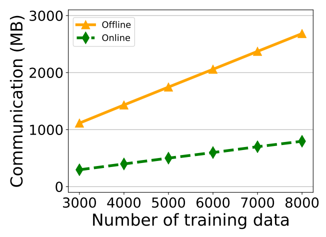
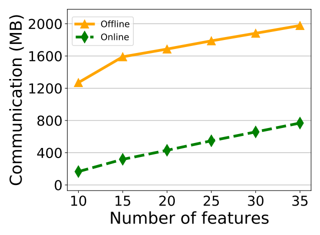
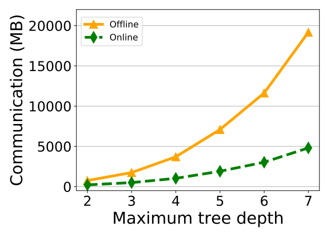

(d) Inference: tree depth

Fig. 5: The runtime of the secure training (s) and inference (ms) process in `PriVDT` under various parameters.


(a) Training: training data


(b) Training: features


(c) Training: tree depth

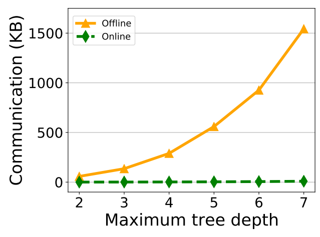

(d) Inference: tree depth

Fig. 6: The communication cost of the secure training (MB) and inference (KB) process in `PriVDT` under various parameters.

REFERENCES

[1] K. Cheng, T. Fan, Y. Jin, Y. Liu, T. Chen, D. Papadopoulos, and Q. Yang, "Secureboost: A lossless federated learning framework," *IEEE Intelligent Systems*, vol. 36, no. 6, pp. 87–98, 2021.

[2] A. T. Azar and S. M. El-Metwally, "Decision tree classifiers for automated medical diagnosis," *Neural Computing and Applications*, vol. 23, no. 7, pp. 2387–2403, 2013.

[3] M.-C. Wu, S.-Y. Lin, and C.-H. Lin, "An effective application of decision tree to stock trading," *Expert Systems with applications*, vol. 31, no. 2, pp. 270–274, 2006.

[4] C. Huang, L. Xue, D. Liu, X. Shen, W. Zhuang, R. Sun, and B. Ying, "Blockchain-assisted transparent cross-domain authorization and authentication for smart city," *IEEE Internet of Things Journal*, 2022.

[5] C. Huang, W. Wang, D. Liu, R. Lu, and X. Shen, "Blockchain-assisted personalized car insurance with privacy preservation and fraud resistance," *IEEE Transactions on Vehicular Technology*, 2022.

[6] W. Du and Z. Zhan, "Building decision tree classifier on private data," in *Proceedings of the IEEE international conference on Privacy, security and data mining*, 2002, pp. 1–8.

[7] K. Wang, Y. Xu, R. She, and P. S. Yu, "Classification spanning private databases," in *Proceedings of the National Conference on Artificial Intelligence*, vol. 21, no. 1, 2006, p. 293.

[8] J. Vaidya, C. Clifton, M. Kantarcioglu, and A. S. Patterson, "Privacy-preserving decision trees over vertically partitioned data," *TKDD*, vol. 2, no. 3, pp. 1–27, 2008.

[9] J. Vaidya, B. Shafiq, W. Fan, D. Mehmood, and D. Lorenzi, "A random decision tree framework for privacy-preserving data mining," *IEEE TDSC*, vol. 11, no. 5, pp. 399–411, 2013.

[10] J. Vaidya and C. Clifton, "Privacy-preserving decision trees over vertically partitioned data," in *IFIP Annual Conference on Data and Applications Security and Privacy*, 2005, pp. 139–152.

[11] Y. Wu, S. Cai, X. Xiao, G. Chen, and B. C. Ooi, "Privacy preserving vertical federated learning for tree-based models," in *Proceedings of VLDB Endowment*, 2020.

[12] W.-Y. Loh, "Classification and regression trees," *Wiley interdisciplinary reviews: data mining and knowledge discovery*, pp. 14–23, 2011.

[13] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Proceedings of EUROCRYPT*, 1999, pp. 223–238.

[14] I. Damgård, V. Pastro, N. Smart, and S. Zakarias, "Multiparty computation from somewhat homomorphic encryption," in *Proceedings of CRYPTO*, 2012, pp. 643–662.

[15] C. Chen, J. Zhou, L. Wang, X. Wu, W. Fang, J. Tan, L. Wang, A. X. Liu, H. Wang, and C. Hong, "When homomorphic encryption marries secret sharing: Secure large-scale sparse logistic regression and applications in risk control," in *Proceedings of KDD*, 2021, pp. 2652–2662.

[16] P. Mohassel and Y. Zhang, "Secureml: A system for scalable privacy-preserving machine learning," in *Proceedings of IEEE S&P*, 2017.

[17] M. S. Riazi, C. Weinert, O. Tkachenko, E. M. Songhori, T. Schneider, and F. Koushanfar, "Chameleon: A hybrid secure computation framework for machine learning applications," in *Proceedings of AsiaCCS*, 2018.

[18] E. Boyle, N. Gilboa, and Y. Ishai, "Function secret sharing," in *Proceedings of EUROCRYPT*, 2015, pp. 337–367.

[19] E. Boyle, N. Chandran, N. Gilboa, D. Gupta, Y. Ishai, N. Kumar, and M. Rathee, "Function secret sharing for mixed-mode and fixed-point secure computation," in *Proceedings of EUROCRYPT*, 2021.

[20] R. E. Goldschmidt, "Applications of division by convergence," Ph.D. dissertation, Massachusetts Institute of Technology, 1964.

[21] S. Wagh, S. Tople, F. Benhamouda, E. Kushilevitz, P. Mittal, and T. Rabin, "Falcon: Honest-majority maliciously secure framework for private deep learning," *Proceedings on Privacy Enhancing Technologies*, vol. 2021, no. 1, pp. 188–208, 2021.

[22] S. Wagh, D. Gupta, and N. Chandran, "Securenn: 3-party secure computation for neural network training." *Proceedings on Privacy Enhancing Technologies*, vol. 2019, no. 3, pp. 26–49, 2019.

[23] S. Adams, C. Choudhary, M. De Cock, R. Dowsley, D. Melanson, A. Nascimento, D. Railsback, and J. Shen, "Privacy-preserving training of tree ensembles over continuous data," *Proceedings on Privacy Enhancing Technologies*, vol. 2, pp. 205–226, 2022.

[24] C. Giannella, K. Liu, T. Olsen, and H. Kargupta, "Communication efficient construction of decision trees over heterogeneously distributed data," in *Proceedings of IEEE ICDM*, 2004, pp. 67–74.

[25] S. Sharma and A. S. Rajawat, "A secure privacy preservation model for vertically partitioned distributed data," in *Proceedings of ICTBIG*, 2016.

[26] M. Hao, H. Li, G. Xu, H. Chen, and T. Zhang, "Efficient, private and robust federated learning," in *Proceedings of ACSAC*, 2021, pp. 45–60.

[27] A. Li, L. Zhang, J. Tan, Y. Qin, J. Wang, and X.-Y. Li, "Sample-level data selection for federated learning," in *Proceedings of IEEE INFOCOM*, 2021.

[28] Y. Liu, Y. Liu, Z. Liu, Y. Liang, C. Meng, J. Zhang, and Y. Zheng, "Federated forest," *IEEE Transactions on Big Data*, 2020.

[29] R. Agrawal, A. Evfimievski, and R. Srikant, "Information sharing across private databases," in *Proceedings of ACM SIGMOD*, 2003, pp. 86–97.

[30] A. C.-C. Yao, "How to generate and exchange secrets," in *Proceedings of FOCS*, 1986, pp. 162–167.

[31] J. R. Quinlan, "Induction of decision trees," *Machine learning*, vol. 1, no. 1, pp. 81–106, 1986.

[32] R. Cramer, I. Damgård, and J. B. Nielsen, "Multiparty computation from threshold homomorphic encryption," in *Proceedings of EUROCRYPT*, 2001, pp. 280–300.

[33] J. Friedman, T. Hastie, and R. Tibshirani, "Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors)," *The annals of statistics*, vol. 28, no. 2, pp. 337–407, 2000.

[34] M. Abspoel, D. Escudero, and N. Volgushev, "Secure training of decision trees with continuous attributes," *Proceedings on Privacy Enhancing Technologies*, vol. 2021, no. 1, pp. 167–187, 2021.

[35] R. K. Tai, J. P. Ma, Y. Zhao, and S. S. Chow, "Privacy-preserving decision trees evaluation via linear functions," in *Proceedings of ESORICS*, 2017.

[36] J. Liang, Z. Qin, S. Xiao, L. Ou, and X. Lin, "Efficient and secure decision tree classification for cloud-assisted online diagnosis services," *IEEE TDSC*, vol. 18, no. 4, pp. 1632–1644, 2019.

[37] J. Bai, X. Song, S. Cui, E.-C. Chang, and G. Russello, "Scalable private decision tree evaluation with sublinear communication," in *Proceedings of AsiaCCS*, 2022.

[38] Y. Zheng, H. Duan, C. Wang, R. Wang, and S. Nepal, "Securely and efficiently outsourcing decision tree inference," *IEEE TDSC*, 2020.

[39] B. Pinkas, T. Schneider, and M. Zohner, "Faster private set intersection based on ot extension," in *Proceedings of USENIX Security*, 2014.

[40] H. Chen, K. Laine, and P. Rindal, "Fast private set intersection from homomorphic encryption," in *Proceedings of ACM CCS*, 2017.

[41] J. Sun, G. Xu, T. Zhang, M. Alazab, and R. H. Deng, "A practical fog-based privacy-preserving online car-hailing service system," *IEEE TIFS*, vol. 17, pp. 2862–2877, 2022.

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

[43] G. Asharov, Y. Lindell, T. Schneider, and M. Zohner, "More efficient oblivious transfer and extensions for faster secure computation," in *Proceedings of ACM CCS*, 2013, pp. 535–548.

[44] G. Dessouky, F. Koushanfar, A.-R. Sadeghi, T. Schneider, S. Zeitouni, and M. Zohner, "Pushing the communication barrier in secure computation using lookup tables," in *Proceedings of NDSS*, 2012.

[45] E. Boyle, N. Gilboa, and Y. Ishai, "Secure computation with preprocessing via function secret sharing," in *Proceedings of TCC*, 2019.

[46] O. Catrina and A. Saxena, "Secure computation with fixed-point numbers," in *Proceedings of FC*, 2010.

[47] P. Mishra, R. Lehmkuhl, A. Srinivasan, W. Zheng, and R. A. Popa, "Delphi: A cryptographic inference service for neural networks," in *Proceedings of USENIX Security*, 2020.

[48] W. Fang, D. Zhao, J. Tan, C. Chen, C. Yu, L. Wang, L. Wang, J. Zhou, and B. Zhang, "Large-scale secure xgb for vertical federated learning," in *Proceedings of CIKM*, 2021.

[49] C. Cachin, S. Micali, and M. Stadler, "Computationally private information retrieval with polylogarithmic communication," in *Proceedings of EUROCRYPT*, 1999, pp. 402–414.

[50] "Ezpc." [Online]. Available: https://github.com/mpc-msri/EzPC

[51] "Openssl." [Online]. Available: https://github.com/openssl/openssl

[52] "libfss." [Online]. Available: https://github.com/frankw2/libfss

[53] A. Asuncion and D. Newman, "Uci machine learning repository," 2007.

[54] "Pivot." [Online]. Available: https://github.com/nusdbsystem/pivot

[55] T. Ryffel, P. Tholoniat, D. Pointcheval, and F. Bach, "Ariann: Low-interaction privacy-preserving deep learning via function secret sharing," *Proceedings on Privacy Enhancing Technologies*, pp. 291–316, 2022.

[56] O. Goldreich, S. Goldwasser, and S. Micali, "How to construct random functions," *Journal of the ACM*, vol. 33, no. 4, pp. 792–807, 1986.

[57] J. Doerner and A. Shelat, "Scaling oram for secure computation," in *Proceedings of ACM CCS*, 2017, pp. 523–535.

[58] K. Yang, C. Weng, X. Lan, J. Zhang, and X. Wang, "Ferret: Fast extension for correlated ot with small communication," in *Proceedings of ACM CCS*, 2020.

[59] B. Knott, S. Venkataraman, A. Hannun, S. Sengupta, M. Ibrahim, and L. van der Maaten, "Crypten: Secure multi-party computation meets machine learning," in *Proceedings of NeurIPS*, 2021.

[60] Y. Dodis, S. Halevi, R. D. Rothblum, and D. Wichs, "Spooky encryption and its applications," in *Proceedings of CRYPTO*, 2016.

[61] P. Mukherjee and D. Wichs, "Two round multiparty computation via multi-key fhe," in *Proceedings of EUROCRYPT*, 2016.

[62] M. Clear and C. McGoldrick, "Multi-identity and multi-key leveled fhe from learning with errors," in *Proceedings of CRYPTO*, 2015.

[63] M. Keller, "Mp-spdz: A versatile framework for multi-party computation," in *Proceedings of ACM CCS*, 2020, pp. 1575–1590.

[64] M. D. Ercegovac and T. Lang, *Digital arithmetic*, 2004.

APPENDIX

A. Function Secret Sharing

We first consider the DCF with a general comparison function $f(x)$ that equals 1 if the input $x < 0$ and 0 otherwise. Following the construction of [19], our FSS solution for comparison is shown in Algorithms 10 and 11. At the high level, k_0 and k_1 respectively define a GGM-style binary tree [56] with 2^n leaves, where each leaf is uniquely labeled by an element $x \in \mathbb{Z}_{2^n}$. We denote a path from the root to the leaf labeled by $x = 0$ as the *special evaluation path*, and as the *evaluation path* otherwise. Each tree node is associated with a triple (s, ψ, t) of length $2k+1$, where the k-bit s is the seed of PRGs[5], k-bit ψ is the output element and 1-bit t is the control bit. The key insight of the FSS-based comparison is to maintain the invariant that 1) for each node on the special evaluation path, the two seeds s_0 and s_1 corresponding respectively to k_0 and k_1 are computationally indistinguishable, and the control bits t_0 and t_1 are opposite; 2) for each node on the evaluation path, the two seeds are identical, as are the two control bits; 3) $\sum_{j=0}^{n} \psi_0$ and $\sum_{j=0}^{n} \psi_1$ over all evaluation nodes of two trees are exactly the secret-shares of $f(x)$. Note that the triple of each node is fully determined by that of its parent. Since the root satisfies $s_0^0 \neq s_1^0$ and $t_0^0 \oplus t_1^0 = 1$, the invariant is auto-maintained in the special evaluation path. On the other hand, for the first node j outside the special evaluation path, if both seeds are corrected to $s_0^j = s_1^j$, and the two control bits meet $t_0^j \oplus t_1^j = 0$, then the invariant will be auto-maintained on the evaluation path. Besides, $\psi_0 + \psi_1$ along the evaluation path is 1 if the departure is to the left of the special evaluation path ($x{<}0$) and 0 otherwise ($x{>}0$). Thus, the task of $\mathsf{Gen}_{\alpha,\beta}^{<}$ where $\alpha = 0$ and $\beta = 1$ is to generate $k_i, i \in \{0,1\}$, which consists of s_i^0 of the root and the correct words CW to guarantee the invariant of the first node departure the special evaluation path.

The construction of DPF also follows the above ideas. Readers can refer to [19] for more details.

Algorithm 10 $\mathsf{Gen}_{\alpha,\beta}^{<}$ of Secure Comparison Protocol

```
1:  //offline
2:  Let $\alpha = \alpha_1, \cdots, \alpha_n \in \{0,1\}^n$ and $s_0^0$, $s_1^0 \in \{0,1\}^k$.
3:  Let $\Psi = 0$ and for $i \in \{0,1\}$, $t_i^0 = i$.
4:  for $j = 1$ to $n$ do
5:      $s_i^L \| \psi_i^L \| t_i^L \| s_i^R \| \psi_i^R \| t_i^R \leftarrow \mathsf{PRG}(s_i^{(j-1)})$, $i \in \{0,1\}$.
6:      let $\mathsf{Keep} = L$, $\mathsf{Lose} = R$ if $\alpha_j = 0$, reverse otherwise.
7:      $S_{cw} = s_0^{\mathsf{Lose}} \oplus s_1^{\mathsf{Lose}}$
8:      $\Psi_{cw} = (-1)^{t_1^{j-1}} \cdot [\psi_1^{\mathsf{Lose}} - \psi_0^{\mathsf{Lose}} - \Psi + 1\{\mathsf{Lose} = L\}]$
9:      $\Psi = \Psi - \psi_1^{\mathsf{Keep}} + \psi_0^{\mathsf{Keep}} + (-1)^{t_1^{j-1}} \cdot \Psi_{cw}$
10:     $T_{cw}^L = t_0^L \oplus t_1^L \oplus \alpha_j \oplus 1$ and $T_{cw}^R = t_0^R \oplus t_1^R \oplus \alpha_j$
11:     $CW^j = S_{cw} \| \Psi_{cw} \| T_{cw}^L \| T_{cw}^R$
12:     $s_i^j = s_i^{\mathsf{Keep}} \oplus t_i^{j-1} \cdot S_{cw}$ for $i \in \{0,1\}$
13:     $t_i^j = t_i^{\mathsf{Keep}} \oplus t_i^{j-1} \cdot T_{cw}$ for $i \in \{0,1\}$
14: end for
15: $CW^{n+1} = (-1)^{t_1^n} \cdot [s_1^n - s_0^n - \Psi]$
16: $k_i = s_i^0 \| CW^1 \| \cdots \| CW^{n+1}$ for $i \in \{0,1\}$
```

B. Secure Truncation Protocol

We recap the truncation method of [16]. We denote by $\mathrm{Tr}_s(x)$ truncating the last s bits of x, where s is the fractional length. The protocol in [16] is executed as follows: given shares $[x]_0$, $[x]_1$ of x, P_0 and P_1 compute $[y]_0 = \mathrm{Tr}_s([x]_0)$ and $[y]_1 = 2^n - \mathrm{Tr}_s(2^n - [x]_1)$, respectively. From Theorem

[5] A Pseudo-Random Generator (PRG) takes a uniformly random seed and a security parameter κ as inputs, and outputs a long pseudorandom string.

Algorithm 11 $\mathsf{Eval}^{<}_{\alpha,\beta}(i, k_i, x)$ of Secure Comparison Protocol

```
1: //online
2: Parse $k_i = s^0 \| CW^1 \| \cdots \| CW^{n+1}$, $x = x_1, \cdots x_n$
3: Let $\Psi = 0$ and $t^0 = i$
4: for $j = 1$ to $n$ do
5:     Parse $CW^j = S_{cw} \| \Psi_{cw} \| T^L_{cw} \| T^R_{cw}$
6:     $\tilde{s}^L \| \tilde{\psi}^L \| \tilde{t}^L \| \tilde{s}^R \| \tilde{\psi}^R \| \tilde{t}^R \leftarrow \mathsf{PRG}(s^{(j-1)})$
7:     $s^L \| t^L \parallel s^R \| t^R = (\tilde{s}^L \| \tilde{t}^L \parallel \tilde{s}^R \| \tilde{t}^R) \oplus [t^{j-1} \cdot (S_{cw} \| T^L_{cw} \parallel S_{cw} \| T^R_{cw})]$
8:     if $x_j = 0$ then
9:         $\Psi = \Psi + (-1)^i \cdot [\tilde{\psi}^L + t^{j-1} \cdot \Psi_{cw}]$
10:        $s^j = s^L, t^j = t^L$
11:    else
12:        $\Psi = \Psi + (-1)^i \cdot [\tilde{\psi}^R + t^{j-1} \cdot \Psi_{cw}]$
13:        $s^j = s^R, t^j = t^R$
14: end for
15: $\Psi = \Psi + (-1)^i \cdot [s^n + t^n \cdot CW^{n+1}]$
```

TABLE IX: The ideal functionality

Comparison functionality $\mathcal{F}_{\mathsf{Compare}}$:
- **Input**: P_0: $[x]_0$, $[x]_1$; P_1: $[x]_1$, $[y]_1$.
- **Output**: P_0: $[z]_0 \in \mathbb{Z}_{2^n}$; P_1: $[z]_1 = 1\{x<y\} - [z]_0 \in \mathbb{Z}_{2^n}$.

Digit decomposition functionality $\mathcal{F}_{\mathsf{DigDec}}$:
- **Input**: P_0: $[x]_0$; P_1: $[x]_1$.
- **Output**: P_0: $[x_{d-1}]_0 \| \cdots \| [x_0]_0$; P_1: $[x_{d-1}]_1 \| \cdots \| [x_0]_1$.

Range functionality $\mathcal{F}_{\mathsf{Range}}$:
- **Input**: P_0: $[x]_0$; P_1: $[x]_1$.
- **Output**: P_0: $[k]_0 \in \mathbb{Z}_{2^n}$; P_1: $[k]_1 = k - [k]_0 \in \mathbb{Z}_{2^n}$, where $2^k \leq x < 2^{k+1}$.

Division functionality $\mathcal{F}_{\mathsf{Div}}$:
- **Input**: P_0: $[x]_0$, $[y]_0$; P_1: $[x]_1$, $[y]_1$.
- **Output**: P_0: $[z]_0 \in \mathbb{Z}_{2^n}$; P_1: $[z]_1 = x/y - [z]_0 \in \mathbb{Z}_{2^n}$.

Sharing multiplication functionality $\mathcal{F}_{\mathsf{SMul}}$:
- **Input**: P_0: $[x]_0$, $[y]_0$; P_1: $[x]_1$, $[y]_1$.
- **Output**: P_0: $[z]_0 \in \mathbb{Z}_{2^n}$; P_1: $[z]_1 = xy - [z]_0 \in \mathbb{Z}_{2^n}$.

Multiplication functionality $\mathcal{F}_{\mathsf{Mul}}$:
- **Input**: P_0: $x \in \mathbb{Z}_{2^n}$; P_1: $y \in \mathbb{Z}_{2^n}$.
- **Output**: P_0: $[z]_0 \in \mathbb{Z}_{2^n}$; P_1: $[z]_1 = xy - [z]_0 \in \mathbb{Z}_{2^n}$.

1 of [16], with probability $1 - \frac{1}{2^{n-n_x-1}}$, $y \in \{\mathrm{Tr}_s(x) - 1, \mathrm{Tr}_s(x), \mathrm{Tr}_s(x)+1\}$, where $x \in [0, 2^{n_x}] \cup [2^n - 2^{n_x}, 2^n)$.

C. Proof of Theorem 1

Proof. The reformulated comparison incurs error when $1\{x < 2^{n-1}\} \neq 1\{x + r \mod 2^n \geq r\}$. We analyze in the following two cases. 1) Consider the case where x is positive, i.e., $x < 2^{n-1}$. The wrong result is assigned if $x + r \mod 2^n < r$. This is true when adding x and r incurs an overflow, i.e., $x + r \geq 2^n$. Since r is drawn at random in $\mathbb{Z}_{2^n}$, the probability of error $P = \frac{x}{2^n}$. 2) For the case where x is negative, the result can be obtained with similar analysis. In this case, the wrong result is assigned if $x + r \mod 2^n \geq r$. This is true when $x + r$ does not overflow, i.e., $x + r < 2^n$. Since r is drawn at random in $\mathbb{Z}_{2^n}$, it results in an error probability of $P = \frac{|x|}{2^n}$, where $x \geq 2^{n-1}$ and $|x| = 2^n - x$. □

D. Proof of Theorem 2

Proof. We prove the security of $\mathsf{Compare}([x],[y])$. STP receives no private information, hence this protocol is trivially secure against semi-honest corruption of STP. Now, we prove the security against corruption of either P_0 or P_1. P_i, $i \in \{0,1\}$, receives $[b]_{1-i} = [y]_{1-i} - [x]_{1-i} + [r]_{1-i}$ and k_i. Given the security of PRFs, $[r]_{1-i}$ is a random value unknown to P_i. Thus the distribution of $[b]_{1-i}$ is uniformly random from P_i's view. Then given the security of FSS, the information learned by P_i can be perfectly simulated. Hence our protocol is trivially secure against semi-honest corruption of P_i. □

E. Proof of Theorem 3

Proof. $\mathsf{DigDec}([x])$ is sequential combination of local computations and invocations of $\mathcal{F}_{\mathsf{Compare}}$, $\mathcal{F}_{\mathsf{LUT}}$, $\mathcal{F}_{\mathsf{FSS}}$, $\mathcal{F}_{\mathsf{SMul}}$ and $\mathcal{F}_{\mathsf{OT}}$. Simulation follows directly from composing the corresponding simulators. □

F. Proof of Theorem 4

Proof. We prove the security of $\mathsf{Range}([x])$. $\mathsf{Range}([x])$ is sequential combinations of local computations and invocations of $\mathcal{F}_{\mathsf{DigDec}}$, $\mathcal{F}_{\mathsf{LUT}}$, $\mathcal{F}_{\mathsf{FSS}}$, $\mathcal{F}_{\mathsf{OT}}$ and $\mathcal{F}_{\mathsf{SMul}}$. Simulation follows directly from composing the corresponding simulators. □

G. Proof of Theorem 5

Proof. We first prove the correctness of $\mathsf{Div}([x],[y])$. The initial approximation $\omega_0 = 2.9142 - 2\tilde{y}$ of $1/\tilde{y}$ introduces an error of $\epsilon_0 < 0.08578$ [46]. Then we iterate twice to get the approximation of $[x]/[y]$, where the final error is $\epsilon_0^{2^2} = 0.54 \times 10^{-4}$. This error is negligible in the ring $\mathbb{Z}_{2^{64}}$ [64]. Now we prove security of this protocol. $\mathsf{Div}([x],[y])$ is sequential combination of local computations and invocations of $\mathcal{F}_{\mathsf{Range}}$, $\mathcal{F}_{\mathsf{LUT}}$, $\mathcal{F}_{\mathsf{OT}}$ and $\mathcal{F}_{\mathsf{SMul}}$. Simulation follows directly from composing the corresponding simulators. □

H. Proof of Theorem 6

Proof. We prove the security of $\mathsf{SMul}([x],[y])$. STP receives no private information, hence this protocol is trivially secure against semi-honest corruption of STP. The messages P_0 received are $[e]_1 = [x]_1 - [a]_1$ and $[d]_1 = [y]_1 - [b]_1$. Given the security of PRFs, $[a]_1$ and $[b]_1$ are random values unknown to P_0. Thus, the distribution of $[e]_1$ and $[d]_1$ are uniformly random from P_0's view and the information learned by P_0 can be perfectly simulated. P_1 receives $[c]_1$, $[e]_0 = [x]_0 - [a]_0$ and $[d]_0 = [y]_0 - [b]_0$. Given the security of PRFs, the distribution of $[c]_1$, $[e]_1$ and $[d]_1$ are uniformly random from P_1's view, hence our protocol is trivially secure against semi-honest corruption of P_1. □

I. Proof of Theorem 7

Proof. We prove the security of $\mathsf{Mul}(x,y)$. STP receives no private information, hence this protocol is trivially secure against semi-honest corruption of STP. The message P_0 received is $e = x - a$. Given the security of PRFs, a is a random value unknown to P_0. Thus, the distribution of e is uniformly random from P_0's view and the information learned by P_0 can be perfectly simulated. P_1 receives $[c]_1$ and $d = y + b$. Given the security of PRFs, the distribution of $[c]_1$ and d are uniformly random from P_1's view, hence our protocol is trivially secure against semi-honest corruption of P_1. □

TABLE X: Communication complexity (bit) of `PriVDT` and Pivot [11] on the secure training process. Given a training dataset, d, k, f are the numbers of samples, classifications, and features, respectively. s' and s denote the numbers of splits of each feature and overall splits, respectively. An example is given (shaded entries), where $d = 4$, $k = 2$, $f = 2$, $s' = 2$, and $s = 4$.

	1. Leave label computation		**2. Gini impurity evaluation**		**3. Best split selection**		**4. Model update**	
	Offline	**Online**	**Offline**	**Online**	**Offline**	**Online**	**Offline**	**Online**
Pivot	$(k-1)(39n-4)$	$4kl+4(k-1)(9n-4)+2l$	$2sn[(k+1)(11n-4)+3(3k+2)]$	$skdl+8sl(k+1)+4sn(12kn+12n-5k-6)$	$68ns-16s$	$47ns-4s$	$-$	$5nl$
	$2,492$	$43,248$	$1,087,488$	$2,867,200$	$17,344$	$12,016$	$-$	$1,310,720$
`PriVDT`	$n(k-1)(2\lambda+2n+3)$	$10n(k-1)$	$nd(k+2s)+2ks[n(d+n+9)+2(d-1)(2n\lambda+2n^2+n+1)]$	$nd(1+k+2s)+ns+2ks[(8n+15)(d-1)+2n(d+n+20)+2]$	$n(s-1)(2\lambda+2n+1)+n(2s+f-3)$	$2n(4s-5+f)$	$ns'(2\lambda+2n+1)+dn$	$2n(1+s'+2d)$
	$24,768$	640	$2,446,944$	$208,624$	$74,368$	$1,664$	$49,536$	$1,408$

J. Proof of Theorem 8 and Theorem 9

We provide proofs of semi-honest simulation based security for the training and inference protocols in Algorithm 8 and 9. We focus on one tree node because each node performs the same operation separately. To proof the security, we describe a simulator Sim to simulate the view of corrupt party P_i by simulating the sequence of hybrid transcripts Hyb_j, where Hyb_0 is the real-world distribution. The view consists of its input/output and received messages. Given the functionality $\mathcal{F}$, $\mathsf{Sim}_{\mathcal{F}}$ simulates the operations in $\mathcal{F}$ and appends its output to the general view. Now, we should prove the indistinguishability of the produced transcript from the real execution against corrupted P_0. The simulator for P_1 is similar.

Proof of indistinguishability during the training phase.

- Hyb_1: Hyb_1 is same as Hyb_0, except $\mathcal{F}_{\mathsf{SMul}}$ is replaced with $\mathsf{Sim}_{\mathsf{SMul}}$ that runs the simulator for the PRF-assisted multiplication procedure. Since $\mathsf{Sim}_{\mathsf{SMul}}$ is guaranteed to produce output indistinguishable from the real-world, Hyb_1 is distributed identically to Hyb_0.
- Hyb_2: Hyb_2 is same as Hyb_1, except $\mathcal{F}_{\mathsf{Mul}}$ is replaced with $\mathsf{Sim}_{\mathsf{Mul}}$. $\mathcal{F}_{\mathsf{Mul}}$ takes the inputs ($\boldsymbol{C_k}$, $\boldsymbol{t_l}$) owned by P_1 and ($[\boldsymbol{\gamma}]_0$, $[\boldsymbol{\delta_k}]_0$) owned by P_0, which does not leak any information to each other. Therefore, the output of $\mathsf{Sim}_{\mathsf{Mul}}$ is indistinguishable from the real-world. Hyb_2 is distributed identically to Hyb_1.
- Hyb_3: Hyb_3 is same as Hyb_2, except $\mathcal{F}_{\mathsf{Compare}}$ is replaced with $\mathsf{Sim}_{\mathsf{Compare}}$ that runs the simulator for FSS generation and evaluation. Besides, P_1 sends $[x_1]_1 - [x_2]_1 + [r]_1$ for a uniformly chosen value, where r is the mask to hide the secret $x_1 - x_2$. According to the security of FSS, Hyb_3 is distributed identically to Hyb_2.
- Hyb_4: Hyb_4 is same as Hyb_3, except $\mathcal{F}_{\mathsf{DIV}}$ is replaced with $\mathsf{Sim}_{\mathsf{DIV}}$. According to the security of division procedure, Hyb_4 is distributed identically to Hyb_3. This concludes the proof.

Proof of indistinguishability during the inference phase.

- Hyb_1: Hyb_1 is same as Hyb_0, except $\mathcal{F}_{\mathsf{SMul}}$ is replaced with $\mathsf{Sim}_{\mathsf{SMul}}$ that runs the simulator for the PRF-assisted multiplication procedure. Since $\mathsf{Sim}_{\mathsf{SMul}}$ is guaranteed to produce output indistinguishable from the real-world, Hyb_1 is distributed identically to Hyb_0.
- Hyb_2: Hyb_2 is same as Hyb_1, except $\mathcal{F}_{\mathsf{Compare}}$ is replaced with $\mathsf{Sim}_{\mathsf{Compare}}$ that runs the simulator for FSS generation and evaluation procedures. Besides, P_1 sends $[x_j]_1 - [s_j]_1 + [r_j]_1$, $j \in [2, z]$ and $[y]_1$ for two uniformly chosen values. According to the security of FSS, Hyb_2 is distributed identically to Hyb_1. This concludes the proof.

TABLE XI: Communication complexity (bit) of `PriVDT` and Pivot [11] on the secure inference process. f' and z denotes the number of features of the inference sample and the number of leaves of the well-trained tree, respectively. An example is given (shaded entries) where $f' = 2$ and $z = 3$.

	Offline	**Online**
Pivot	$2(z-1)(21n-4)+6zn$	$4l(2z-1)+nf'+2(z-1)(4n-16)+4zn$
	$6,512$	$83,776$
`PriVDT`	$2n(z-1)(\lambda+n+1)+n(z+1)$	$4n(3z-2)$
	$49,664$	$1,792$

K. Communication complexity analysis

We analyze the theoretical communication complexity of `PriVDT`, and give the comparison with Pivot [11], where the security parameter $\lambda = 128$ and ring size $n = 64$ in `PriVDT`, and the length of an HE ciphertext in Pivot is $l = 4096$.

Secure training phase. As detailed in Section VI-B, the training phase in `PriVDT` can be divided into 4 steps: leave label computation, gini impurity evaluation, best split selection, and model update. In table X, we report the communication overhead for each step, where we only give the overhead of one recursion. For ease of understanding, we also give a concrete example. We can observe that for the online communication, `PriVDT` always outperforms Pivot, achieving a significant communication boost of about $20\times$. Although the offline communication of `PriVDT` is higher than that of Pivot, the overall communication cost of `PriVDT` still has a significant advantage.

Secure inference phase. Table XI provides the communication analysis for the inference phase of `PriVDT` and Pivot. Given an inference sample with 2 features and a well-trained tree with 3 leaves, `PriVDT` improves the communication overhead of Pivot by $46\times$ during the online phase. The communication bottleneck of Pivot mainly comes from the expensive HE-to-MPC conversion, and costly comparison and division protocols.