Privacy Preserving Vertical Federated Learning for Tree-based Models

Yuncheng Wu[†], Shaofeng Cai[†], Xiaokui Xiao[†], Gang Chen[‡], Beng Chin Ooi[†]
[†]National University of Singapore [‡]Zhejiang University
{wuyc, shaofeng, xiaoxk, ooibc}@comp.nus.edu.sg cg@zju.edu.cn

ABSTRACT

Federated learning (FL) is an emerging paradigm that enables multiple organizations to jointly train a model without revealing their private data to each other. This paper studies *vertical* federated learning, which tackles the scenarios where (i) collaborating organizations own data of the same set of users but with disjoint features, and (ii) only one organization holds the labels. We propose Pivot, a novel solution for privacy preserving vertical decision tree training and prediction, ensuring that no intermediate information is disclosed other than those the clients have agreed to release (i.e., the final tree model and the prediction output). Pivot does not rely on any trusted third party and provides protection against a semi-honest adversary that may compromise $m-1$ out of m clients. We further identify two privacy leakages when the trained decision tree model is released in plaintext and propose an enhanced protocol to mitigate them. The proposed solution can also be extended to tree ensemble models, e.g., random forest (RF) and gradient boosting decision tree (GBDT) by treating single decision trees as building blocks. Theoretical and experimental analysis suggest that Pivot is efficient for the privacy achieved.

PVLDB Reference Format:
Yuncheng Wu, Shaofeng Cai, Xiaokui Xiao, Gang Chen, Beng Chin Ooi. Privacy Preserving Vertical Federated Learning for Tree-based Models. *PVLDB*, 13(11): xxxx-yyyy, 2020.
DOI: https://doi.org/10.14778/3407790.3407811

1. INTRODUCTION

There has been a growing interest in exploiting data from distributed databases of multiple organizations, for providing better customer service and acquisition. *Federated learning* (FL) [47,48] (or *collaborative learning* [39]) is an emerging paradigm for machine learning that enables multiple data owners (i.e., *clients*) to jointly train a model without revealing their private data to each other. The basic idea of FL is to iteratively let each client (i) perform some local computations on her data to derive certain intermediate results, and then (ii) exchange these results with other clients in a secure manner to advance the training process, until a final model is obtained. The advantage of FL is that it helps each client protect her data assets, so as to abide by privacy regulations (e.g., GDPR [2] and CCPA [1]) or to maintain a competitive advantage from proprietary data.

Proceedings of the VLDB Endowment, Vol. 13, No. 11
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3407790.3407811

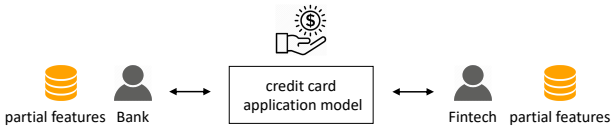


Figure 1: Example of vertical federated learning

Existing work on FL has mainly focused on the *horizontal* setting [7,8,47,48,50,53,54,60,75], which assumes that each client's data have the same schema, but no tuple is shared by multiple clients. In practice, however, there is often a need for *vertical federated learning*, where all clients hold the same set of records, while each client only has a disjoint subset of features. For example, Figure 1 illustrates a digital banking scenario, where a bank and a Fintech company aim to jointly build a machine learning model that evaluates credit card applications. The bank has some partial information about the users (e.g., account balances), while the Fintech company has some other information (e.g., the users' online transactions). In this scenario, vertical FL could enable the bank to derive a more accurate model, while the Fintech company could benefit from a pay-per-use model [67] for its contribution to the training and prediction.

To our knowledge, there exist only a few solutions [19,40,45,54,61–63,65] for privacy preserving vertical FL. These solutions, however, are insufficient in terms of either efficiency or data privacy. In particular, [40,65] assume that the labels in the training data could be shared with all participating clients in plaintext, whereas in practice, the labels often exist in one client's data only and could not be revealed to other clients without violating privacy. For instance, in the scenario illustrated in Figure 1, the training data could be a set of historical credit card applications, and each label would be a ground truth that indicates whether the application should have been approved. In this case, the labels are only available to the bank and could not be directly shared with the Fintech company. As a consequence, the solutions in [40,65] are inapplicable. Meanwhile, [19,45,61–63] assume that some intermediate results during the execution could be revealed in plaintext; nevertheless, such intermediate results could be exploited by an adversarial client to infer the sensitive information in other clients' data. The solution in [54], on the other hand, relies on secure hardware [46] for privacy protection, but such secure hardware may not be trusted by all parties [75] and could be vulnerable to

side channel attacks [70]. The method in [51] utilizes secure multiparty computation (MPC) [72], but assumes that each client's data could be outsourced to a number of non-colluding servers. This assumption is rather strong, as it is often challenging in practice to ensure that those servers do not collude and to convince all clients about it.

To address the above issues, we propose *Pivot*, a novel and efficient solution for vertical FL that does not rely on any trusted third party and provides protection against a semi-honest adversary that may compromise $m-1$ out of m clients. Pivot is a part of our *Falcon*[1] (federated learning with privacy protection) system, and it ensures that no intermediate information is disclosed during the training or prediction process. Specifically, Pivot is designed for training decision tree (DT) models, which are well adopted for financial risk management [19, 45], healthcare analytics [5], and fraud detection [14] due to their good interpretability. The core of Pivot is a hybrid framework that utilizes both threshold partially homomorphic encryption (TPHE) and MPC, which are two cryptographic techniques that complement each other especially in the vertical FL setting: TPHE is relatively efficient in terms of communication cost but can only support a restrictive set of computations, whereas MPC could support an arbitrary computation but incurs expensive communication overheads. Pivot employs TPHE as much as possible to facilitate clients' local computation, and only invokes MPC in places where TPHE is inadequate in terms of functionality. This leads to a solution that is not only secure but also highly efficient for vertical tree models, as demonstrated in Section 7. Specifically, we make the following contributions:

- We propose a basic protocol of Pivot that supports the training of both classification trees and regression trees, as well as distributed prediction using the tree models obtained. This basic protocol guarantees that each client only learns the final tree model but nothing else. To our knowledge, Pivot is the first vertical FL solution that achieves such a guarantee.
- We enhance the basic protocol of Pivot to handle a more stringent case where parts of the final tree model need to be concealed for better privacy protection. In addition, we propose extensions of Pivot for training several *ensemble* tree-based models, including random forest (RF) and gradient boosting decision trees (GBDT).
- We implement DT, RF, and GBDT models based on Pivot and conduct extensive evaluations on both real and synthetic datasets. The results demonstrate that Pivot offers accuracy comparable to non-private algorithms and provides high efficiency. The basic and enhanced protocols of Pivot achieve up to 37.5x and 4.5x speedup (*w.r.t.* training time) over an MPC baseline.

2. PRELIMINARIES

2.1 Partially Homomorphic Encryption

A partially homomorphic encryption (PHE) scheme is a probabilistic asymmetric encryption scheme for restricted computation over the ciphertexts. In this paper, we utilize the Paillier cryptosystem [25, 55], which consists of three algorithms (**Gen, Enc, Dec**):

- The *key generation* algorithm $(sk, pk) = \textbf{Gen}(keysize)$ which returns secret key sk and public key pk, given a security parameter *keysize*.
- The *encryption* algorithm $c = \textbf{Enc}(x, pk)$, which maps a plaintext x to a ciphertext c using pk.
- The *decryption* algorithm $x = \textbf{Dec}(c, sk)$, which reverses the encryption by sk and outputs the plaintext x.

For simplicity, we omit the public key pk in the **Enc** algorithm and write $\textbf{Enc}(x)$ as $[x]$ in the rest of the paper. Let x_1, x_2 be two plaintexts. We utilize the following properties:

- **Homomorphic addition:** given $[x_1]$, $[x_2]$, the ciphertext of the sum is $[x_1] \oplus [x_2] = [x_1 + x_2]$.
- **Homomorphic multiplication:** given x_1, $[x_2]$, the ciphertext of the product is $x_1 \otimes [x_2] = [x_1 x_2]$.
- **Homomorphic dot product:** given a ciphertext vector $[\boldsymbol{v}] = ([v_1], \cdots, [v_m])^T$ and a plaintext vector $\boldsymbol{x} = (x_1, \cdots, x_m)$, the ciphertext of the dot product is $\boldsymbol{x} \odot [\boldsymbol{v}] = (x_1 \otimes [v_1]) \oplus \cdots \oplus (x_m \otimes [v_m]) = [\boldsymbol{x} \cdot \boldsymbol{v}]$.

We utilize a *threshold variant* of the PHE scheme (i.e., TPHE) with the following additional properties. First, the public key pk is known to everyone, while each client only holds a partial secret key. Second, the decryption of a ciphertext requires inputs from a certain number of clients. In this paper, we use a *full threshold structure*, which requires all clients to participate in order to decrypt a ciphertext.

2.2 Secure Multiparty Computation

Secure multiparty computation (MPC) allows participants to compute a function over their inputs while keeping the inputs private. In this paper, we utilize the SPDZ [26] additive secret sharing scheme for MPC. We refer to a value $a \in \mathbb{Z}_q$ that is additively shared among clients as a *secretly shared value*, and denote it as $\langle a \rangle = (\langle a \rangle_1, \cdots, \langle a \rangle_m)$, where $\langle a \rangle_i$ is a random *share* of a hold by client i. To reconstruct a, every client can send her own share to a specific client who computes $a = (\sum_{i=1}^{m} \langle a \rangle_i) \mod q$. For ease of exposition, we omit the modular operation in the rest of the paper.

Given secretly shared values, we mainly use the following *secure computation* primitives in SPDZ as building blocks: secure addition, secure multiplication [6], secure division [16], secure exponential [36], and secure comparison [15, 16]. The output of any secure computation is also a secretly shared value unless it is reconstructed. We refer interested readers to [4, 26] for the detailed constructions. The secret sharing based MPC has two phases: an offline phase that is independent of the function and generates pre-computed Beaver's triplets [6], and an online phase that computes the designated function using these triplets.

2.3 Tree-based Models

In this paper, we consider the classification and regression trees (CART) algorithm [11] with binary structure, while we note that other variants (e.g., ID3 [58], C4.5 [59]) can be easily generalized. We assume there is a training dataset D with n data points $\{\mathbf{x}_1, \cdots, \mathbf{x}_n\}$ each containing d features and the corresponding output label set $Y = \{y_1, \cdots, y_n\}$.

The CART algorithm builds a tree recursively. For each tree node, it first decides whether some pruning conditions are satisfied, e.g., feature set is empty, tree reaches the maximum depth, the number of samples is less than a threshold. If any pruning condition is satisfied, then it returns a leaf node with the class of majority samples for classification or

[1] https://www.comp.nus.edu.sg/~dbsystem/fintech/project/falcon/

the mean label value for regression. Otherwise, it determines the best split to construct two sub-trees that are built recursively. To find the best split, CART uses Gini impurity [11] as a metric in classification. Let c be the number of classes and $K = \{1, \cdots, c\}$ be the class set. Let D be sample set on a given node, the Gini impurity is:

$$I_G(D) = 1 - \sum_{k \in K} (p_k)^2 \quad (1)$$

where p_k is the fraction of samples in D labeled with class k. Let F be the set of available features, given any split feature $j \in F$ and split value $\tau \in \text{Domain}(j)$, the sample set D can be split into two partitions D_l and D_r. Then, the *impurity gain* of this split is as follows:

$$\begin{aligned} gain &= I_G(D) - (w_l \cdot I_G(D_l) + w_r \cdot I_G(D_r)) \\ &= w_l \sum_{k \in K} (p_{l,k})^2 + w_r \sum_{k \in K} (p_{r,k})^2 - \sum_{k \in K} (p_k)^2 \end{aligned} \quad (2)$$

where $w_l = |D_l|/|D|$ and $w_r = |D_r|/|D|$, and $p_{l,k}$ (resp. $p_{r,k}$) is the fraction of samples in D_l (resp. D_r) that are labeled with class $k \in K$. The split with the maximum impurity gain is considered the best split of the node.

For regression, CART uses the label variance as a metric. Let Y be the set of labels of D, then the label variance is:

$$I_V(D) = E(Y^2) - (E(Y))^2 = \frac{1}{n} \sum_{i=1}^{n} y_i^2 - \left(\frac{1}{n} \sum_{i=1}^{n} y_i \right)^2 \quad (3)$$

Similarly, the best split is determined by maximizing the variance gain. With CART, ensemble models can be trained for better accuracy, e.g., random forest (RF) [10], gradient boosting decision tree (GBDT) [31, 32], XGBoost [18], etc.

3. SOLUTION OVERVIEW

3.1 System Model

We consider a set of m distributed clients (or data owners) $\{u_1, \cdots, u_m\}$ who want to train a decision tree model by consolidating their respective datasets $\{D_1, \cdots, D_m\}$. Each row in the datasets corresponds to a sample, and each column corresponds to a feature. Let n be the number of samples and d_i be the number of features in D_i, where $i \in \{1, \cdots, m\}$. We denote $D_i = \{\mathbf{x}_{it}\}_{t=1}^{n}$ where $\mathbf{x}_{it}$ represents the t-th sample of D_i. Let $Y = \{y_t\}_{t=1}^{n}$ be the set of sample labels.

Pivot focuses on the vertical federated learning scenario [71], where $\{D_1, \cdots, D_m\}$ share the same sample ids while with different features. In particular, we assume that the clients have determined and aligned their common samples using private set intersection [17, 49, 56, 57]. Besides, we assume that the label set Y is held by only one client (i.e., *super client*) and cannot be directly shared with other clients.

3.2 Threat Model

We consider the semi-honest model [20, 21, 33, 51, 68, 69] where every client follows the protocol exactly as specified, but may try to infer other clients' private information based on the messages received. Like any other client, no additional trust is assumed of the super client. We assume that an adversary $\mathcal{A}$ can corrupt up to $m - 1$ clients and the adversary's corruption strategy is static, such that the set of corrupted clients is fixed before the protocol execution and remains unchanged during the execution.

3.3 Problem Formulation

To protect the private data of honest clients, we require that an adversary learns nothing more than the data of the clients he has corrupted and the final output. Similar to previous work [21, 51, 75], we formalize our problem under the ideal/real paradigm. Let $\mathcal{F}$ be an ideal functionality such that the clients send their data to a trusted third party for computation and receive the final output from that party. Let π be a real world protocol executed by the clients. We say a real protocol π behaviors indistinguishably as the ideal functionality $\mathcal{F}$ if the following formal definition is satisfied.

Definition 1. ([13, 23, 51]). A protocol π securely realizes an ideal functionality $\mathcal{F}$ if for every adversary $\mathcal{A}$ attacking the real interaction, there exists a simulator $\mathcal{S}$ attacking the ideal interaction, such that for all environments $\mathcal{Z}$, the following quantity is negligible (in λ):

$$\left| \Pr[\text{REAL}(\mathcal{Z}, \mathcal{A}, \pi, \lambda) = 1] - \Pr[\text{IDEAL}(\mathcal{Z}, \mathcal{S}, \mathcal{F}, \lambda) = 1] \right|. \square$$

In this paper, we identify two ideal functionalities $\mathcal{F}_{\text{DTT}}$ and $\mathcal{F}_{\text{DTP}}$ for the model training and model prediction, respectively. In $\mathcal{F}_{\text{DTT}}$, the input is every client's dataset while the output is the trained model that all clients have agreed to release. In $\mathcal{F}_{\text{DTP}}$, the input is the released model and a sample while the output is the predicted label of that sample. The output of $\mathcal{F}_{\text{DTT}}$ is part of the input of $\mathcal{F}_{\text{DTP}}$. Specifically, in our basic protocol (Section 4), we assume that the output of $\mathcal{F}_{\text{DTT}}$ is the plaintext tree model, including the split feature and the split threshold on each internal node, and the label for prediction on each leaf node. While in our enhanced protocol (Section 5), the released plaintext information is assumed to include only the split feature on each internal node, whereas the split threshold and the leaf label are concealed for better privacy protection.

3.4 Protocol Overview

We now provide the protocol overview of Pivot. The protocols are composed of three stages: initialization, model training, and model prediction.

Initialization stage. In this stage, the m clients agree to run a designated algorithm (i.e., the decision tree model) over their joint data and release the pre-defined information (e.g., the trained model) among themselves. The clients collaboratively determine and align the joint samples. The clients also build consensus on some hyper-parameters, such as security parameters (e.g., key size), pruning thresholds, and so on. The m clients jointly generate the keys of threshold homomorphic encryption and every client u_i receives the public key pk and a partial secret key sk_i.

Model training stage. The m clients build the designated tree model iteratively. In each iteration, the super client first broadcasts some encrypted information to facilitate the other clients to compute encrypted necessary statistics at local. After that, the clients jointly convert those statistics into MPC-compatible inputs, i.e., secretly shared values, to determine the best split of the current tree node using secure computations. Finally, the secretly shared best split is revealed (in the Pivot basic protocol) or is converted back into an encrypted form (in the Pivot enhanced protocol), for clients to update the model. Throughout the whole process, no intermediate information is disclosed to any client.

Model prediction stage. After model training, the clients obtain a tree model. In the basic protocol of Pivot (Section 4), the whole tree is released in plaintext. In the Pivot enhanced protocol (Section 5), the split threshold on each internal node and the prediction label on each leaf node are

concealed from all clients, in secretly shared form. Given an input sample with distributed feature values, the clients can jointly produce a prediction. Pivot guarantees that no information except for the predicted label is revealed during the prediction process.

4. BASIC PROTOCOL

In this section, we present our basic protocol of Pivot. The output of the model training stage is assumed to be the whole plaintext tree model. Note that prior work [19, 40, 45, 61–63, 65] is not applicable to our problem since they simplify the problem by revealing either the training labels or intermediate results in plaintext, which discloses too much information regarding the client's private data.

To satisfy Definition 1 for vertical tree training, a straightforward solution is to directly use the MPC framework. For example, the clients can apply the additive secret sharing scheme (see Section 2.2) to convert private datasets and labels into secretly shared data, and train the model by secure computations. However, this solution incurs high communication complexity because it involves $O(nd)$ secretly shared values and most secure computations are communication intensive. On the other hand, while TPHE could enable each client to compute encrypted split statistics at local by providing the super client's encrypted label information, it does not support some operations (e.g., comparison), which are needed in best split determination. Based on these observations and inspired by [75], we design our basic protocol using a hybrid framework of TPHE and MPC for vertical tree training. The basic idea is that each client executes as many local computations (e.g., computing split statistics) as possible with the help of TPHE and uses MPC only when TPHE is insufficient (e.g., deciding the best split). As a consequence, most computations are executed at local and the secretly shared values involved in MPC are reduced to $O(db)$, where b denotes the maximum number of split values for any feature and db is the number of total splits.

Section 4.1 and Section 4.2 present our training protocol for classification tree and regression tree, respectively. Section 4.3 proposes our tree model prediction method. The security analysis is provided in Section 4.4.

4.1 Classification Tree Training

In our training protocol, the clients use an *mask vector* of size n to indicate which samples are available on a tree node, but keep the vector in an encrypted form to avoid disclosing the sample set. Specifically, let $\boldsymbol{\alpha} = (\alpha_1, \cdots, \alpha_n)$ be an indicator vector for a tree node ρ. Then, for any $i \in \{1, \cdots, n\}$, $\alpha_i = 1$ indicates that the i-th sample is available on ρ, and $\alpha_i = 0$ otherwise. We use $[\boldsymbol{\alpha}] = ([\alpha_1], \cdots, [\alpha_n])$ to denote the encrypted version of $\boldsymbol{\alpha}$, where $[\cdot]$ represents homomorphic encrypted values (see Section 2.1).

Before the training starts, each client initializes a decision tree with only a root node, and associates the root node with an encrypted indicator vector $[\boldsymbol{\alpha}]$ where all elements are $[1]$ (since all samples are available on the root node). Then, the clients work together to recursively split the root node. In what follows, we will use an example to illustrate how our protocol decides the best split for a given tree node based on Gini impurity.

Consider the example in Figure 2, where we have three clients u_1, u_2, and u_3. Among them, u_1 is the super client, and she owns the labels with two classes, 1 and 2. There are five training samples with three features (i.e., income, age,

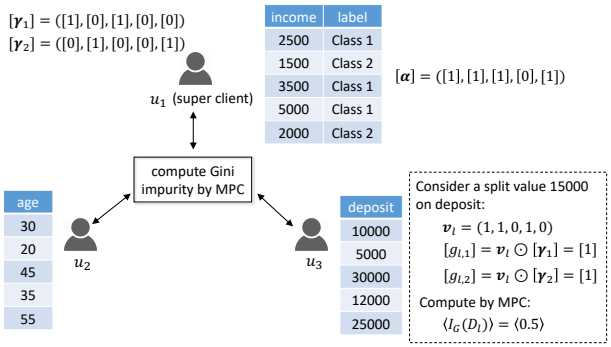


Figure 2: Classification tree training example

and deposit), and each client holds one feature. Suppose that the clients are to split a tree node ρ whose encrypted mask vector is $[\boldsymbol{\alpha}] = ([1], [1], [1], [0], [1])$, i.e., Samples 1, 2, 3, and 5 are on the node. Then, u_1 computes an encrypted indicator vector for each class, based on $[\boldsymbol{\alpha}]$ and her local labels. For example, for Class 1, u_1 derives an temporary indicator vector $(1, 0, 1, 1, 0)$, which indicates that Samples 1, 3, and 4 belong to Class 1. Next, u_1 uses the indicator vector to perform an element-wise homomorphic multiplication with $[\boldsymbol{\alpha}]$, which results in an encrypted indicator vector $[\boldsymbol{\gamma}_1] = ([1], [0], [1], [0], [0])$. This vector indicates that Samples 1 and 3 are on the node to be split, and they belong to Class 1. Similarly, u_1 also generates an encrypted indicator vector $[\boldsymbol{\gamma}_2]$ for Class 2. After that, u_1 broadcasts $[\boldsymbol{\gamma}_1]$ and $[\boldsymbol{\gamma}_2]$ to all clients.

After receiving $[\boldsymbol{\gamma}_1]$ and $[\boldsymbol{\gamma}_2]$, each client combines them with her local training data to compute several statistics that are required to choose the best split of the current node. In particular, to evaluate the quality of a split based on Gini impurity (see Section 2.3), each client needs to examine the two child nodes that would result from the split, and then compute the following statistics for each child node: (i) the total number of samples that belong to the child node, and (ii) the number of samples among them that are associated with label class k, for each $k \in K$.

For example, suppose that u_3 considers a split that divides the current node based on whether the deposit values are larger than 15000. Then, u_3 first examines her local samples, and divide them into two partitions. The first partition (referred to as the *left partition*) consists of Samples 1, 2, and 4, i.e., the local samples whose deposit values are no more than 15000. Meanwhile, the second partition (referred to as the *right partition*) contains Samples 3 and 5. Accordingly, for the left (resp. right) partition, u_3 constructs an indicator vector $\boldsymbol{v}_l = (1, 1, 0, 1, 0)$ (resp. $\boldsymbol{v}_r = (0, 0, 1, 0, 1)$) to specify the samples that it contains. After that, u_3 performs a homomorphic dot product between $\boldsymbol{v}_l$ and $[\boldsymbol{\gamma}_1]$ to obtain an encrypted number $[g_{l,1}]$. Observe that $g_{l,1}$ equals the exact number of Class 1 samples that belong to the left child node of the split. Similarly, u_3 uses $\boldsymbol{v}_l$ and $[\boldsymbol{\gamma}_2]$ to generate $[g_{l,2}]$, an encrypted version of the number of Class 2 samples that belong to the left child node. Using the same approach, u_3 also computes the encrypted numbers of Classes 1 and 2 samples associated with the right child node. Further, u_3 derives an encrypted total number of samples in the left (resp. right) child node, using a homomorphic dot product between $\boldsymbol{v}_l$ and $[\boldsymbol{\alpha}]$ (resp. $\boldsymbol{v}_r$ and $[\boldsymbol{\alpha}]$).

Suppose that each client computes the encrypted numbers associated with each possible split of the current node, us-

Algorithm 1: Conversion to secretly shared value

Input: $[x]$: a ciphertext, $\mathbb{Z}_q$: secret sharing scheme space
pk: the public key, $\{sk_i\}_{i=1}^m$: partial secret keys
Output: $\langle x\rangle = (\langle x\rangle_1, \cdots, \langle x\rangle_m)$: secretly shared x

for $i \in [1, m]$ **do**
  $[r_i] \leftarrow u_i$ randomly chooses $r_i \in \mathbb{Z}_q$ and encrypts it
  u_i sends $[r_i]$ to u_1
u_1 computes $[e] = [x] \oplus [r_1] \oplus \cdots \oplus [r_m]$
$e \leftarrow$ clients jointly decrypt $[e]$
u_1 sets $\langle x\rangle_1 = e - r_1 \mod q$
for $i \in [2, m]$ **do**
  u_i sets $\langle x\rangle_i = -r_i \mod q$

ing the approach illustrated for u_3 above. Then, they can convert them into MPC-compatible inputs, and then invoke an MPC protocol to securely identify the best split of the current node. We will elaborate on the details shortly.

In general, the clients split each node in three steps: local computation, MPC computation, and model update. In the following, we discuss the details of each step.

Local computation. Suppose that the clients are to split a node that is associated with an encrypted mask vector $[\boldsymbol{\alpha}] = ([\alpha_1], \cdots, [\alpha_n])$, indicating the available samples on the node. First, for each class label $k \in K$, the super client constructs an auxiliary indicator vector $\boldsymbol{\beta}_k = (\beta_{k,1}, \cdots, \beta_{k,n})$, such that $\beta_{k,t} = 1$ if Sample t's label is k, and $\beta_{k,t} = 0$ otherwise. After that, the super client performs an element-wise homomorphic multiplication between $\boldsymbol{\beta}_k$ and $[\boldsymbol{\alpha}]$, obtaining an encrypted indicator vector $[\boldsymbol{\gamma}_k]$. Then, the super client broadcasts $[\mathbb{\Gamma}] = \bigcup_{k\in K}\{[\boldsymbol{\gamma}_k]\}$ to other clients.

Upon receiving $[\mathbb{\Gamma}]$, each client u_i uses it along with her local data to derive several statistics for identifying the tree node's best split, as previously illustrated in our example. In particular, let F_i be the set of features that u_i has, and S_{ij} be the set of split values for a feature $j \in F_i$. Then, for any split value $\tau \in S_{ij}$, u_i first constructs two size-n indicator vectors $\boldsymbol{v}_l$ and $\boldsymbol{v}_r$, such that (i) the t-th element in $\boldsymbol{v}_l$ equals 1 if Sample t's feature j is no more than τ, and 0 otherwise, and (ii) $\boldsymbol{v}_r$ complements $\boldsymbol{v}_l$. Consider the two possible child nodes induced by τ. For each class $k \in K$, let $g_{l,k}$ (resp. $g_{r,k}$) be the number of samples labeled with class k that belongs to the left (resp. right) child node. u_i computes the encrypted versions of $g_{l,k}$ and $g_{r,k}$ using homomorphic dot products (see Section 2.1) as follows:

$$[g_{l,k}] = \boldsymbol{v}_l \odot [\boldsymbol{\gamma}_k], \quad [g_{r,k}] = \boldsymbol{v}_r \odot [\boldsymbol{\gamma}_k]. \tag{4}$$

Let n_l (resp. n_r) be the number of samples in the left (resp. right) child node. u_i computes $[n_l] = \boldsymbol{v}_l \odot [\boldsymbol{\alpha}]$ and $[n_r] = \boldsymbol{v}_r \odot [\boldsymbol{\alpha}]$. In total, for each split value τ, u_i generates $2 \cdot |K| + 2$ encrypted numbers, where $|K| = c$ is the number of classes.

MPC computation. After the clients generate the encrypted statistics mentioned above (i.e., $[g_{l,k}]$, $[g_{r,k}]$, $[n_l]$, $[n_r]$), they execute an MPC protocol to identify the best split of the current node. Towards this end, the clients first invoke Algorithm 1 to convert each encrypted number $[x]$ into a set of shares $\{\langle x\rangle_i\}_{i=1}^m$, where $\langle x\rangle_i$ is given to u_i. The general idea of Algorithm 1 is from [22, 26, 75]. We use $\langle x\rangle$ to denote that the x is secretly shared among the clients.

After the above conversion, the clients obtain secretly shared statistics $\langle n_l\rangle$, $\langle n_r\rangle$, $\langle g_{l,k}\rangle$, and $\langle g_{r,k}\rangle$ (for each class $k \in K$) for each possible split τ of the current node. Using these statistics, the clients identify the best split of the current node as follows.

Algorithm 2: Pivot DT training (basic protocol)

Input: $\{D_i\}_{i=1}^m$: local datasets, $\{F_i\}_{i=1}^m$: local features
Y: label set, $[\boldsymbol{\alpha}]$: encrypted mask vector
pk: the public key, $\{sk_i\}_{i=1}^m$: partial secret keys
Output: T: decision tree model

if *prune conditions satisfied* **then**
  classification: return leaf node with majority class
  regression: return leaf node with mean label value
else
  the super client computes $[\mathbb{\Gamma}]$ and broadcasts it
  for $i \in [1, m]$ **do**
    for $j \in F_i$ **do**
      for $s \in [1, |S_{ij}|]$ **do**
        u_i computes encrypted statistics by $[\mathbb{\Gamma}]$
  clients convert encrypted statistics to shares
  determine the best split identifier (i^*, j^*, s^*)
  client i^* computes $[\boldsymbol{\alpha}_l], [\boldsymbol{\alpha}_r]$ and broadcasts them
  return a tree with j^*-th feature and s^*-th split value that has two edges, build tree recursively

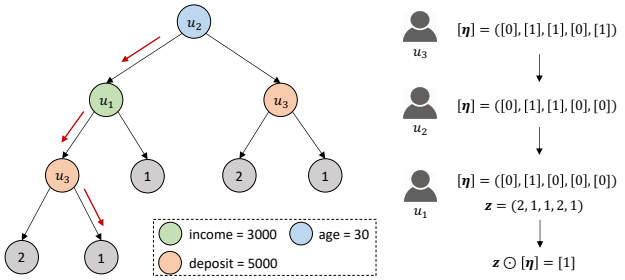
Consider a split τ and the two child nodes that it induces. To evaluate the Gini impurity of the left (resp. right) child node, the clients need to derive, for each class $k \in K$, the fraction $p_{l,k}$ (resp. $p_{r,k}$) of samples on the node that are labeled with k. Observe that

$$p_{l,k} = \frac{g_{l,k}}{\sum_{k'\in K} g_{l,k'}}, \quad p_{r,k} = \frac{g_{r,k}}{\sum_{k'\in K} g_{r,k'}}. \tag{5}$$

In addition, recall that the clients have obtained, for each class $k \in K$, the secretly shared values $\langle g_{l,k}\rangle$ and $\langle g_{r,k}\rangle$. Therefore, the clients can jointly compute $\langle p_{l,k}\rangle$ and $\langle p_{r,k}\rangle$ using the secure addition and secure division operators in SPDZ (see Section 2.2), without disclosing $p_{l,k}$ and $p_{r,k}$ to any client. With the same approach, the clients use $\langle n_l\rangle$ and $\langle n_r\rangle$ to securely compute $\langle w_l\rangle$ and $\langle w_r\rangle$, where $w_l = \frac{n_l}{n_l+n_r}$ and $w_r = \frac{n_r}{n_l+n_r}$. Given $\langle p_{l,k}\rangle$, $\langle p_{r,k}\rangle$, $\langle w_l\rangle$, and $\langle w_r\rangle$, the clients can then compute the impurity gain of each split τ (see Eqn. (2)) in the secretly shared form, using the secure addition and secure multiplication operators in SPDZ.

Finally, the clients jointly determine the best split using a secure maximum computation as follows. First, each client u_i assigns an identifier (i, j, s) to the s-th split on the j-th feature that she holds. Next, the clients initialize four secretly shared values $\langle gain_{max}\rangle$, $\langle i^*\rangle$, $\langle j^*\rangle$, $\langle s^*\rangle$, all with $\langle -1\rangle$. After that, they will compare the secretly shared impurity gains of all splits, and securely record the identifier and impurity gain of the best split in $\langle i^*\rangle$, $\langle j^*\rangle$, $\langle s^*\rangle$, and $\langle gain_{max}\rangle$, respectively. Specifically, for each split τ, the clients compare its impurity gain $\langle gain_\tau\rangle$ with $\langle gain_{max}\rangle$ using secure comparison (see Section 2.2). Let $\langle sign\rangle$ be the result of the secure comparison, i.e., $sign = 1$ if $gain_\tau > gain_{max}$, and $sign = 0$ otherwise. Then, the clients securely update $\langle gain_{max}\rangle$ using the secretly shared values, such that $gain_{max} = gain_{max} \cdot (1 - sign) + gain_\tau \cdot sign$. The best split identifier is updated in the same manner. After examining all splits, the clients obtain the secretly shared best split identifier $(\langle i^*\rangle, \langle j^*\rangle, \langle s^*\rangle)$.

Model update. Once the best split is computed, the clients reconstruct the identifier (i^*, j^*, s^*) of the best split. (Recall that the basic protocol releases the tree model in plaintext.) Then, the i^*-th client retrieves the two indicator vectors $\boldsymbol{v}_l$ and $\boldsymbol{v}_r$ for the s^*-th split of her j^*-th feature. After that, she executes an element-wise homomorphic multiplication



(a) Tree model (b) Prediction

Figure 3: Tree model prediction example with a sample (age = 25, income = 2500, deposit = 6000): (a) tree model: colored circles denote internal nodes and gray circles denote leaf nodes; (b) prediction: clients update a encrypted prediction vector in a round-robin manner.

between $\boldsymbol{v}_l$ (resp. $\boldsymbol{v}_r$) and $[\boldsymbol{\alpha}]$, obtaining an encrypted vector $[\boldsymbol{\alpha}_l]$ (resp. $[\boldsymbol{\alpha}_r]$), which she then broadcasts to other clients. Note that $[\boldsymbol{\alpha}_l]$ (resp. $[\boldsymbol{\alpha}_r]$) specifies the samples on the left (resp. right) child node resulting from the best split.

4.2 Regression Tree Training

For the regression tree, the central part is to compute the label variance. The MPC computation and model update steps are similar to those of classification, and hence, we only present the difference in the local computation step.

By the label variance formula Eqn. (3), the super client can construct two auxiliary vectors $\boldsymbol{\beta}_1 = (y_1, \cdots, y_n)$ and $\boldsymbol{\beta}_2 = (y_1^2, \cdots, y_n^2)$, where the elements in $\boldsymbol{\beta}_1$ are the original training labels while the elements in $\boldsymbol{\beta}_2$ are the squares of the original training labels. Next, she computes element-wise homomorphic multiplication on $\boldsymbol{\beta}_1$ (resp. $\boldsymbol{\beta}_2$) by $[\boldsymbol{\alpha}]$, obtaining $[\boldsymbol{\gamma}_1]$ (resp. $[\boldsymbol{\gamma}_2]$). Then, she broadcasts $[\mathbb{\Gamma}] = \{[\boldsymbol{\gamma}_1], [\boldsymbol{\gamma}_2]\}$ to all clients. Similarly, each client computes the following encrypted statistics for any local split:

$$[n_l] = \boldsymbol{v}_l \odot [\boldsymbol{\alpha}], \quad [g_{l,1}] = \boldsymbol{v}_l \odot [\boldsymbol{\gamma}_1], \quad [g_{l,2}] = \boldsymbol{v}_l \odot [\boldsymbol{\gamma}_2] \quad (6)$$

where $[n_l], [g_{l,1}], [g_{l,2}]$ are the encrypted number of samples, and the encrypted sum of $[\boldsymbol{\gamma}_1]$ and $[\boldsymbol{\gamma}_2]$ of the available samples, for the left branch. Similarly, these encrypted statistics can be converted into secretly shared values and the best split identifier can be decided based on Eqn. (3).

Algorithm 2 describes the privacy preserving decision tree training protocol. Lines 1-3 check the pruning conditions and compute the leaf label if any condition is satisfied. Note that with the encrypted statistics, these executions can be easily achieved by secure computations. Lines 5-13 find the best split and build the tree recursively, where Lines 5-9 are the local computation step for computing encrypted split statistics; Line 10-11 are the MPC computation step that converts the encrypted statistics into secretly shared values and decides the best split identifier using MPC; and Line 12 is the model update step, which computes the encrypted indicator vectors for the child nodes given the best split.

4.3 Tree Model Prediction

After releasing the plaintext tree model, the clients can jointly make a prediction given a sample. In vertical FL, the features of a sample are distributed among the clients. Figure 3a shows an example of a released model, where each internal node represents a feature with a split threshold owned by a client, and each leaf node represents a predicted label on that path. To predict a sample, a naive method is to let the super client coordinate the prediction process [19]: starting from the root node, the client who has the node feature compares its value with the split threshold, and notifies super client the next branch; then the prediction is forwarded to the next node until a leaf node is reached. However, this method discloses the prediction path, from which a client can infer other client's feature value along that path.

Algorithm 3: Pivot DT prediction (basic protocol)

Input: T: decision tree model, $\{\mathbf{x}_i\}_{i=1}^m$: input sample
pk: the public key, $\{sk_i\}_{i=1}^m$: partial secret keys
Output: $\bar{k}$: predicted label

1 **for** $i \in [m, 1]$ **do**
2 **if** $i == m$ **then**
3 u_i initializes $[\boldsymbol{\eta}] = ([1], \cdots, [1])$ with size $t+1$
4 **if** $i > 1$ **then**
5 u_i updates $[\boldsymbol{\eta}]$ using $(T, \boldsymbol{x}_i)$
6 u_i sends $[\boldsymbol{\eta}]$ to u_{i-1}
7 **else**
8 u_i updates $[\boldsymbol{\eta}]$ using $(T, \boldsymbol{x}_i)$
9 u_i initializes label vector $\boldsymbol{z} = (z_1, \cdots, z_{t+1})$
10 u_i computes $[\bar{k}] = \boldsymbol{z} \odot [\boldsymbol{\eta}]$
11 clients jointly decrypt $[\bar{k}]$ by $\{sk_i\}_{i=1}^m$ and return $\bar{k}$

To ensure that no additional information other than the predicted output is leaked, we propose a distributed prediction method, as shown in Algorithm 3. Let $\boldsymbol{z} = (z_1, \cdots, z_{t+1})$ be the *leaf label vector* of the leaf nodes in the tree model, where t is the number of internal nodes. Note that all clients know $\boldsymbol{z}$ since the tree model is public in this protocol. Given a sample, clients collaborate to update an encrypted prediction vector $[\boldsymbol{\eta}] = ([1], \cdots, [1])$ with size $t+1$ in a round-robin manner. Each element in $[\boldsymbol{\eta}]$ indicates if a prediction path is possible with encrypted form.

Without loss of generality, we assume that the prediction starts with u_m and ends with u_1. If a prediction path is possible from the perspective of a client, then the client multiplies the designated element in $[\boldsymbol{\eta}]$ by 1 using homomorphic multiplication, otherwise by 0. Figure 3b illustrates an example of this method. Starting from u_3, given the feature value 'deposit = 6000', u_3 initializes $[\boldsymbol{\eta}]$ and updates it to $([0],[1],[1],[0],[1])$, since she can eliminate the first and fourth prediction paths after comparing her value with the split threshold 'deposit = 5000'. Then, u_3 sends $[\boldsymbol{\eta}]$ to the next client for updates. After all clients' updates, there is only one $[1]$ in $[\boldsymbol{\eta}]$, which indicates the true prediction path. Finally, u_1 computes $\boldsymbol{z} \odot [\boldsymbol{\eta}]$ to get the encrypted prediction output, and decrypts it jointly with all clients.

4.4 Security Guarantees

THEOREM 1. *The basic protocol of Pivot securely realizes the ideal functionalities $\mathcal{F}_{DTT}$ and $\mathcal{F}_{DTP}$ against a semi-honest adversary who can statically corrupt up to $m-1$ out of m clients.*

Proof Sketch. For model training, the proof can be reduced to the computations on one tree node because each node can be computed separately given that its output is public [42, 43]. Since the threshold Paillier scheme [55] is secure, the transmitted messages in the local computation and model update steps are secure. Meanwhile, the MPC conversion [22] and additive secret sharing scheme [26] are secure, thus, the MPC computation step is secure. Consequently, an adversary learns no additional information from the protocol execution, the security follows.

For model prediction, an adversary $\mathcal{A}$ only view an encrypted prediction vector $[\boldsymbol{\eta}]$ updated by the honest client(s), and the encrypted prediction output $[\bar{k}]$. Thus, no information is learned beyond the decrypted prediction output, because the threshold Paillier scheme is secure. □

5. ENHANCED PROTOCOL

The basic protocol guarantees that no intermediate information is disclosed. However, after obtaining the public model, colluding clients may extract private information of a target client's training dataset, with the help of their own datasets. We first present two possible privacy leakages in Section 5.1 and then propose an enhanced protocol that mitigates this problem by concealing some model information in Section 5.2. The security analysis is given in Section 5.3.

5.1 Privacy Leakages

We identify two possible privacy leakages: the training label leakage and the feature value leakage, regarding a target client's training dataset. The intuition behind the leakages is that the colluding clients are able to split the sample set based on the split information in the model and their own datasets. We illustrate them by the following two examples given the tree model in Figure 3.

Example 1. (Training label leakage). Assume that u_2 and u_3 collude, let us see the right branch of the root node. u_2 knows exactly the sample set in this branch, say $D_{\text{age} > 30}$, as all samples are available on the root node, and he can just split his local samples based on 'age = 30'. Then, u_3 can classify this set into two subsets given the 'deposit=5000' split, say $D_{\text{age} > 30 \wedge \text{deposit} \leq 5000}$ and $D_{\text{age} > 30 \wedge \text{deposit} > 5000}$, respectively. Consequently, according to the plaintext class labels on the two leaf nodes, colluding clients may infer that the samples in $D_{\text{age} > 30 \wedge \text{deposit} \leq 5000}$ are with class 2 and vise versa, with high probability.

Example 2. (Feature value leakage). Assume that u_1 and u_2 collude, let us see the path of $u_2 \rightarrow u_1 \rightarrow u_3$ (with red arrows). Similar to Example 1, u_1 and u_2 can know exactly the training sample set on the 'u_3' node before splitting, say D'. In addition, recall that u_1 is the super client who has all sample labels, thus, he can easily classify D' into two sets by class, say D'_1 and D'_2, respectively. Consequently, the colluding clients may infer that the samples in D'_2 have 'deposit $\leq$ 5000' and vise versa, with high probability.

Note that these two leakages happen when the clients (except the target client) along a path collude. Essentially, given the model, the colluding clients (without super client) may infer labels of some samples in the training dataset if there is no feature belongs to the super client along a tree path; similarly, if the super client involves in collusion, the feature values of some samples in the training dataset of a target client may be inferred.

5.2 Hiding Label and Split Threshold

Our observation is that these privacy leakages can be mitigated if the split thresholds on the internal nodes and the leaf labels on the leaf nodes in the model are concealed from all clients. Without such information, the colluding clients can neither determine how to split the sample set nor what leaf label a path owns. We now discuss how to hide such information in the model.

For the leaf label on each leaf node, the clients can convert it to an encrypted value, instead of reconstructing its plaintext. Specifically, after obtaining the secretly shared leaf label (e.g., $\langle k \rangle$) using secure computations (Lines 1-3 in Algorithm 2), each client encrypts her own share of $\langle k \rangle$ and broadcasts to all clients. Then, the encrypted leaf label can be computed by summing up these encrypted shares using homomorphic addition. As such, the leaf label is concealed.

For the split threshold on each internal node, the clients hide it by two additional computations in the model update step. Recall that in the basic protocol, the best split identifier $(\langle i^* \rangle, \langle j^* \rangle, \langle s^* \rangle)$ is revealed to all clients after the MPC computation in each iteration. In the enhanced protocol, we assume that $\langle s^* \rangle$ is not revealed, and thus the split threshold can be concealed. To support the tree model update without disclosing s^* to the i^*-th client, we first use the private information retrieval (PIR) [68, 69] technique to privately select the split indicator vectors of s^*.

Private split selection. Let $n' = |S_{ij}|$ denote the number of splits of the j^*-th feature of the i^*-th client. We assume n' is public for simplicity. Note that the clients can further protect n' by padding placeholders to a pre-defined threshold number. Instead of revealing $\langle s^* \rangle$ to the i^*-th client, the clients jointly convert $\langle s^* \rangle$ into an encrypted indicator vector $[\boldsymbol{\lambda}] = ([\lambda_1], \cdots, [\lambda_{n'}])^T$, such that $\lambda_t = 1$ when $t = s^*$ and $\lambda_t = 0$ otherwise, where $t \in \{1, \cdots, n'\}$. This vector is sent to the i^*-th client for private split selection at local. Let $\boldsymbol{V}^{n \times n'} = (\boldsymbol{v}_1, \cdots, \boldsymbol{v}_{n'})$ be the split indicator matrix, where $\boldsymbol{v}_t$ is the split indicator vector of the t-th split of the j^*-th feature (see Section 4.1). The following theorem [68] suggests that the i^*-th client can compute the encrypted indicator vector for the s^*-th split without disclosing s^*.

THEOREM 2. *Given an encrypted indicator vector* $[\boldsymbol{\lambda}] = ([\lambda_1], \cdots, [\lambda_{n'}])^T$ *such that* $[\lambda_{s^*}] = [1]$ *and* $[\lambda_t] = [0]$ *for all* $t \neq s^*$, *and the indicator matrix* $\boldsymbol{V}^{n \times n'} = (\boldsymbol{v}_1, \cdots, \boldsymbol{v}_{n'})$, *then* $[\boldsymbol{v}_{s^*}] = \boldsymbol{V} \otimes [\boldsymbol{\lambda}]$. □

The notion $\otimes$ represents the homomorphic matrix multiplication, which executes homomorphic dot product operations between each row in $\boldsymbol{V}$ and $[\boldsymbol{\lambda}]$. We refer the interested readers to [68] for the details.

For simplicity, we denote the selected $[\boldsymbol{v}_{s^*}]$ as $[\boldsymbol{v}]$. The encrypted split threshold can also be obtained by homomorphic dot product between the encrypted indicator vector $[\boldsymbol{\lambda}]$ and the plaintext split value vector of the j^*-th feature.

Encrypted mask vector updating. After finding the encrypted split vector $[\boldsymbol{v}]$, we need to update the encrypted mask vector $[\boldsymbol{\alpha}]$ for protecting the sample set recursively. This requires element-wise multiplication between $[\boldsymbol{\alpha}]$ and $[\boldsymbol{v}]$. Thanks to the MPC conversion algorithm, we can compute $[\boldsymbol{\alpha}] \cdot [\boldsymbol{v}]$ as follows [22]. For each element pair $[\alpha_j]$ and $[v_j]$ where $j \in [1, n]$, we first convert $[\alpha_j]$ into $\langle \alpha_j \rangle = (\langle \alpha_j \rangle_1, \cdots, \langle \alpha_j \rangle_m)$ using Algorithm 1, where $\langle \alpha_j \rangle_i (i \in \{1, \cdots, m\})$ is the share hold by u_i; then each client u_i executes homomorphic multiplication $\langle \alpha_j \rangle_i \otimes [v_j] = [\langle \alpha_j \rangle_i \cdot v_j]$ and sends the result to the i^*-th client; finally, the i^*-th client can sum up the results using homomorphic addition:

$$[\alpha'_j] = [\langle \alpha_j \rangle_1 \cdot v_j] \oplus \cdots \oplus [\langle \alpha_j \rangle_m \cdot v_j] = [\alpha_j \cdot v_j] \qquad (7)$$

After updating $[\boldsymbol{\alpha}]$, the tree can also be built recursively, similar to the basic protocol.

Secret sharing based model prediction. The prediction method in the basic protocol is not applicable here as the clients cannot directly compare their feature values with the

encrypted split thresholds. Hence, the clients first convert the encrypted split thresholds and encrypted leaf labels into secretly shared form and make predictions on the secretly shared model using MPC. Let $\langle \boldsymbol{z} \rangle$ with size $(t+1)$ denote the secretly shared leaf label vector, where t is the number of internal nodes. Given a sample, the clients also provide the feature values in the secretly shared form. After that, the clients initialize a secretly shared prediction vector $\langle \boldsymbol{\eta} \rangle$ with size $(t+1)$, indicating if a prediction path is possible. Then, they compute this vector as follows.

The clients initialize a secretly shared marker $\langle 1 \rangle$ for the root node. Starting from the root node, the clients recursively compute the markers of its child nodes until all leaf nodes are reached. Then, the marker of each leaf node is assigned to the corresponding element in $\langle \boldsymbol{\eta} \rangle$. After examining all the nodes, there will be only one $\langle 1 \rangle$ element in $\langle \boldsymbol{\eta} \rangle$, specifying the real prediction path in a secret manner. Specifically, each marker is computed by secure multiplication between its parent node's marker and a secure comparison result (between the secretly shared feature value and split threshold on this node). For example, in Figure 3a, the split threshold on the root node will be $\langle 30 \rangle$ while the feature value will be $\langle 25 \rangle$, and then $\langle 1 \rangle$ is assigned to its left child and $\langle 0 \rangle$ to its right child. The clients know nothing about the assigned markers since the computations are secure. Finally, the secretly shared prediction output can be readily obtained by a dot product between $\langle \boldsymbol{z} \rangle$ and $\langle \boldsymbol{\eta} \rangle$, using secure computations.

Discussion. A noteworthy aspect is that the clients can also choose to hide the feature $\langle j^* \rangle$ by defining n' as the total number of splits on the i^*-th client, or even the client $\langle i^* \rangle$ that has the best feature by defining n' as the total number of splits among all clients. By doing so, the leakages could be further alleviated. However, the efficiency and interpretability would be degraded greatly. In fact, there is a trade-off between privacy and efficiency (interpretability) for the released model. The less information the model reveals, the higher privacy while the lower efficiency and less interpretability the clients obtain, and vise versa.

5.3 Security Guarantees

THEOREM 3. *The enhanced protocol of Pivot securely realizes the ideal functionalities $\mathcal{F}_{DTT}$ and $\mathcal{F}_{DTP}$ against a semi-honest adversary who can statically corrupt up to $m-1$ out of m clients.*

Proof Sketch. For model training, the only difference from the basic protocol is the two additional computations (private split selection and encrypted mask vector updating) in the model update step, which are computed using threshold Paillier scheme and MPC conversion. Thus, the security follows. For model prediction, since the additive secret sharing scheme is secure and the clients compute a secretly shared marker for every possible path, the adversary learns nothing except the final prediction output. □

6. EXTENSIONS TO OTHER ML MODELS

So far, Pivot supports a single tree model. Now we briefly present how to extend the basic protocol to ensemble tree models, including random forest (RF) [10] and gradient boosting decision tree (GBDT) [31, 32] in Section 6.1 and 6.2, respectively. Same as the basic protocol, we assume that all trees can be released in plaintext. The extension to other machine learning models is discussed in Section 6.3.

6.1 Random Forest

RF constructs a set of independent decision trees in the training stage, and outputs the class that is the mode of the predicted classes (for classification) or mean prediction (for regression) of those trees in the prediction stage.

For model training, the extension from a single decision tree is natural since each tree can be built (using Algorithm 2) and released separately. For model prediction, after obtaining the encrypted predicted label of each tree, the clients can easily convert these encrypted labels into secretly shared values for majority voting using secure computations (for classification) or compute the encrypted mean prediction by homomorphic computations (for regression).

6.2 Gradient Boosting Decision Trees

GBDT uses decision trees as weak learners and improves model quality with a boosting strategy [30]. The trees are built sequentially where the training labels for the next tree are the prediction losses between the ground truth labels and the prediction outputs of previous trees.

Model training. The extension to GBDT is non-trivial, since we need to prevent the super client from knowing the training labels of each tree except the first tree (i.e., intermediate information) while facilitating the training process.

We first consider GBDT regression. Let W be the number of rounds and a regression tree is built in each round. Let Y^w be the training label vector of the w-th tree. We aim to protect Y^w by keeping it in an encrypted form. After building the w-th tree where $w \in \{1, \cdots, W-1\}$, the clients jointly make predictions for all training samples to get an encrypted estimation vector $[\bar{Y}^w]$; then the clients can compute the encrypted training labels $[Y^{w+1}]$ of the $(w+1)$-th tree given $[Y^w]$ and $[\bar{Y}^w]$. Besides, note that in Section 4.2, an encrypted label square vector $[\boldsymbol{\gamma}_2^{w+1}]$ is needed, which is computed by element-wise homomorphic multiplication between $\boldsymbol{\beta}_2^{w+1}$ and $[\boldsymbol{\alpha}]$. However, $\boldsymbol{\beta}_2^{w+1}$ is not plaintext here since the training labels are ciphertexts. Thus, the clients need expensive element-wise ciphertext multiplications (see Section 5.2) between $[\boldsymbol{\beta}_2^{w+1}]$ and $[\boldsymbol{\alpha}]$ in each iteration. To optimize this computation, we slightly modify our basic protocol. Instead of letting the super client compute $[\boldsymbol{\gamma}_2^{w+1}]$ in each iteration, we now let the client who has the best split update $[\boldsymbol{\gamma}_2^{w+1}]$ along with $[\boldsymbol{\alpha}]$ using the same split indicator vector and broadcast them to all clients. In this way, the clients only need to compute $[\boldsymbol{\gamma}_2^{w+1}]$ using $[\boldsymbol{\beta}_2^{w+1}]$ and $[\boldsymbol{\alpha}]$ once at the beginning of each round, which reduces the cost.

For GBDT classification, we use the one-vs-the-rest technique by combining a set of binary classifiers. Essentially, the clients need to build a GBDT regression forest for each class, resulting in $W * c$ regression trees in total (c is the number of classes). After each round in the training stage, the clients obtain c trees; and for each training sample, the clients make a prediction on each tree, resulting in c encrypted prediction outputs. Then, the clients jointly convert them into secretly shared values for computing *secure softmax* (which can be constructed using secure exponential, secure addition, and secure division, as mentioned in Section 2.2), and convert them back into an encrypted form as encrypted estimations. The rest of the computation is the same as regression.

Model prediction. For GBDT regression, the prediction output can be decrypted after homomorphic aggregating the encrypted predictions of all trees. For GBDT classification,

Table 1: Model accuracy comparison with non-private baselines

Dataset	Pivot-DT	NP-DT	Pivot-RF	NP-RF	Pivot-GBDT	NP-GBDT
Bank market	0.886077	0.886188	0.888619	0.890497	0.891271	0.892044
Credit card	0.821526	0.821533	0.823056	0.823667	0.825167	0.827167
Appliances energy	212.05281	211.45229	211.55175	211.32113	211.35326	210.75291

the encrypted prediction for each class is the same as that for regression; then the clients jointly convert these encrypted results into secretly shared values for deciding the final prediction output by *secure softmax* function.

6.3 Other Machine Learning Models

Though we consider tree-based models in this paper, the proposed solution can be easily adopted in other vertical FL models, such as logistic regression (LR), neural networks, and so on. The rationale is that these models can often be partitioned into the three steps described in Section 4.1. As a result, the TPHE primitives, conversion algorithm, and secure computation operators can be re-used.

For example, the clients can train a vertical LR model as follows. To protect the intermediate weights of the LR model during the training, the clients initialize an encrypted weight vector, $[\boldsymbol{\theta}] = ([\boldsymbol{\theta}_1], \cdots, [\boldsymbol{\theta}_m])$, where $[\boldsymbol{\theta}_i]$ corresponds to the encrypted weights of features held by client i. In each iteration, for a Sample t, each client i first locally aggregates an encrypted partial sum, say $[\xi_{it}]$, by homomorphic dot product between $[\boldsymbol{\theta}_i]$ and Sample t's local features $\mathbf{x}_{it}$. Then the clients jointly convert $\{[\xi_{it}]\}_{i=1}^{m}$ into secretly shared values using Algorithm 1, and securely aggregate them before computing the secure logistic function. Meanwhile, the super client also provides Sample t's label as a secretly shared value, such that the clients can jointly compute the secretly shared loss of Sample t. After that, the clients convert the loss back into the encrypted form (see Section 5.2), and each client can update her encrypted weights $[\boldsymbol{\theta}_i]$ using homomorphic properties, without knowing the loss. Besides, the model prediction is a half component of one iteration in training, which can be easily computed.

7. EXPERIMENTS

We evaluate the performance of Pivot basic protocol (Section 4) and Pivot enhanced protocol (Section 5) on the decision tree model, as well as the ensemble extensions (Section 6). We present the accuracy evaluation in Section 7.2 and the efficiency evaluation in Section 7.3.

We implement Pivot in C++ and employ the *GMP*[2] library for big integer computation and the *libhcs*[3] library for threshold Paillier scheme. We utilize the *SPDZ*[4] library for semi-honest additive secret sharing scheme. Besides, we use the *libscapi*[5] library for network communications among clients. Since the cryptographic primitives only support big integer computations, we convert the floating point datasets into fixed-point integer representation.

7.1 Experimental Setup

We conduct experiments on a cluster of machines in a local area network (LAN). Each machine is equipped with Intel (R) Xeon (R) CPU E5-1650 v3 @ 3.50GHz×12 and 32GB of RAM, running Ubuntu 16.04 LTS. Unless noted otherwise, the *keysize* of threshold Paillier scheme is 1024 bits and the security parameter of SPDZ configuration is 128 bits.

Datasets. We evaluate the model accuracy using three real-world datasets: credit card data (30000 samples with 25 features) [73], bank marketing data (4521 samples with 17 features) [52], and appliances energy prediction data (19735 samples with 29 features) [12]. The former two datasets are for classification while the third dataset is for regression.

We evaluate the efficiency using synthetic datasets, which are generated with *sklearn*[6] library. Specifically, we vary the number of samples (n) and the number of total features (d) to generate datasets, and then equally split these datasets *w.r.t.* features into m partitions, which are held by m clients, respectively. We denote $\bar{d} = d/m$ as the number of features each client holds. For classification tasks, the number of classes is set to 4, and only one client holds the labels.

Baselines. For accuracy evaluation, we adopt the non-private decision tree (NP-DT), non-private random forest (NP-RF), and non-private gradient-boosting decision tree (NP-GBDT) algorithms from *sklearn* for comparison. For a fair comparison, we adopt the same hyper-parameters for both our protocols and the baselines, e.g., the maximum tree depth, the pruning conditions, the number of trees, etc.

For efficiency evaluation, to our knowledge, there is no existing work providing the same privacy guarantee as Pivot. Therefore, we implement a secret sharing based decision tree algorithm using the SPDZ library (namely, SPDZ-DT) as a baseline. The security parameter of SPDZ-DT is also 128 bits. Besides, we also implement a non-private distributed decision tree (NPD-DT) algorithm as another baseline to illustrate the overhead of providing strong privacy.

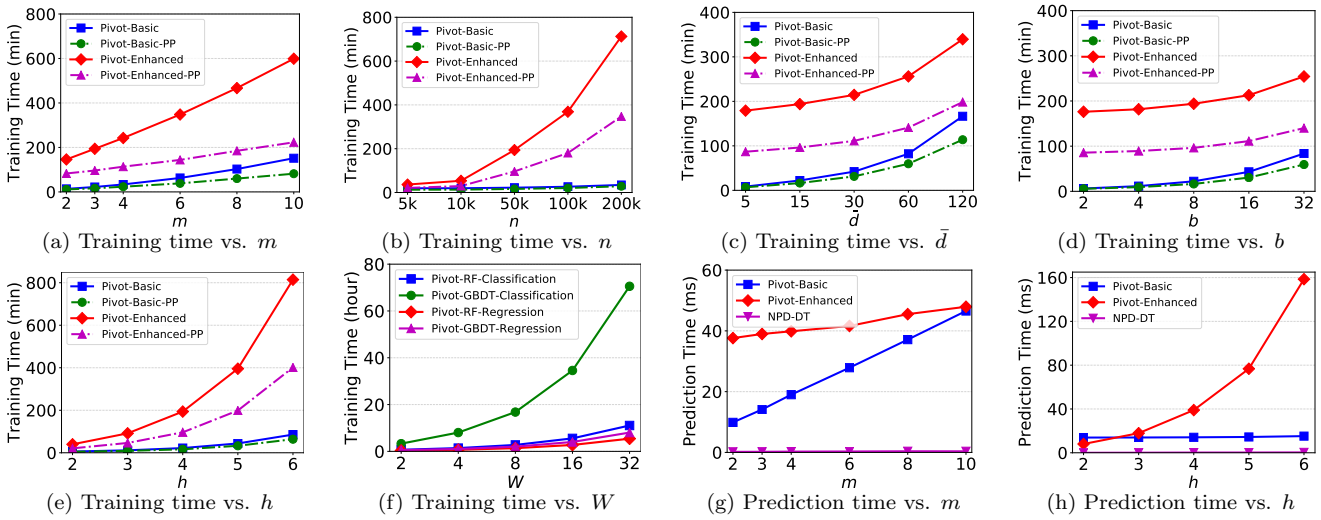
Metrics. For model accuracy, we measure the number of samples that are correctly classified over the total testing samples for classification; and the mean square error (MSE) between the predicted labels and the ground truth labels for regression. For efficiency, we measure the total running time of the model training stage and the prediction running time per sample of the model prediction stage. In all experiments, we report the running time of the online phase because SPDZ did not support the offline time benchmark for the semi-honest additive secret sharing protocol.

7.2 Evaluation of Accuracy

In terms of accuracy, we compare the performance of the proposed decision tree (Pivot-DT), random forest (Pivot-RF) and gradient boosting decision tree (Pivot-GBDT) algorithms with their non-private baselines on three real world datasets. In these experiments, the *keysize* of threshold Paillier scheme is set to 512 bits. We conduct 10 independent trials of each experiment and report the average result.

Table 1 summarizes the comparison results. We can notice that the Pivot algorithms achieve accuracy comparable to the non-private baselines. There are two reasons for the slight loss of accuracy. First, we use the fixed-point integer to represent float values, whose precision is thus truncated.

[2] `http://gmplib.org`
[3] `https://github.com/tiehuis/libhcs`
[4] `https://github.com/data61/MP-SPDZ`
[5] `https://github.com/cryptobiu/libscapi`
[6] `https://scikit-learn.org/stable/`



(a) Training time vs. m (b) Training time vs. n (c) Training time vs. $\bar{d}$ (d) Training time vs. b

(e) Training time vs. h (f) Training time vs. W (g) Prediction time vs. m (h) Prediction time vs. h

Figure 4: Effect of parameters in decision tree models

Second, Pivot has only implemented the basic algorithms, which are not optimized as the adopted baselines.

7.3 Evaluation of Efficiency

In terms of efficiency, we evaluate the training and prediction time of Pivot with the two protocols (namely Pivot-Basic and Pivot-Enhanced) in Section 7.3.1 and Section 7.3.2, by varying the number of clients (m), the number of samples (n), the number of features of each client ($\bar{d}$), the maximum number of splits (b), the maximum tree depth (h), and the number of trees (W) for ensemble methods.

We employ parallelism with 6 cores for threshold decryption of multiple ciphertexts, which is observed to be the most time-consuming part in Pivot. The secure computations using SPDZ are not parallelized because the current SPDZ cannot express parallelism effectively and flexibly. These partially parallelized versions are denoted as Pivot-Basic-PP and Pivot-Enhanced-PP, respectively. The comparison with the baselines is reported in Section 7.3.3. Table 2 describes the ranges and default settings of the evaluated parameters.

7.3.1 Evaluation on Training Efficiency

Varying m. Figure 4a shows the performance for varying m. The training time of all algorithms increases as m increases because the threshold decryptions and secure computations need more communication rounds. Pivot-Basic always performs better than Pivot-Enhanced since Pivot-Enhanced has two additional computations in the model update step, where the $O(n)$ ciphertexts multiplications dominate the cost. Besides, we can see that Pivot-Enhanced-PP that parallelizes only the threshold decryptions could reduce the total training time by up to 2.7 times.

Varying n. Figure 4b shows the performance for varying n. The comparison of the Pivot-Basic and Pivot-Enhanced is similar to Figure 4a, except that the training time of Pivot-Basic increases slightly when n goes up. The reason is that, in Pivot-Basic, the cost of encrypted statistics computation (proportional to $O(n)$) is only a small part of the total training time; the time-consuming parts are the MPC conversion that requires $O(cdb)$ threshold decryptions. The training time of Pivot-Enhanced scales linearly with n because of the threshold decryptions for encrypted mask vector updating are proportional to $O(n)$.

Table 2: Parameters adopted in the evaluation

Parameter	Description	Range	Default
m	number of clients	$[2, 10]$	3
n	number of samples	$[5K, 200K]$	$50K$
$\bar{d}$	number of features	$[5, 120]$	15
b	maximum splits	$[2, 32]$	8
h	maximum tree depth	$[2, 6]$	4
W	number of trees	$[2, 32]$	–

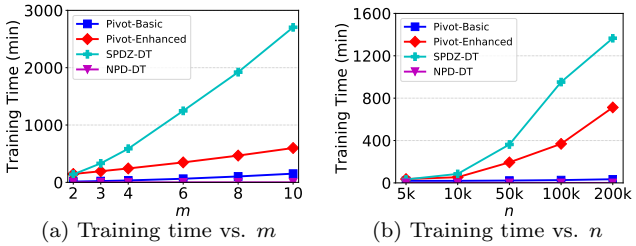
Varying $\bar{d}, b$. Figure 4c-4d show the performance for varying $\bar{d}$ and b, respectively. The trends of the four algorithms in these two experiments are similar, i.e., the training time all scales linearly with $\bar{d}$ or b since the number of total splits is $O(db)$. In addition, the gap between Pivot-Basic and Pivot-Enhanced is stable as $\bar{d}$ or b increases. This is because that $\bar{d}$ does not affect the additional costs in Pivot-Enhanced, and b only has small impact via private split selection (i.e., $O(nb)$ ciphertext computations) which is negligible comparing to the encrypted mask vector update.

Varying h. Figure 4e shows the performance for varying h. Since the generated synthetic datasets are sampled uniformly, the trained models tend to construct a full binary tree, where the number of internal nodes is $2^h - 1$ given the maximum tree depth h. Therefore, the training time of all algorithms approximately double when h increases by one.

Varying W. Figure 4f shows the performance for varying W in ensemble methods. RF classification is slightly slower than RF regression as the default c is 4 in classification comparing to 2 in regression. GBDT regression is slightly slower than RF regression, since additional computations are required by GBDT to protect intermediate training labels. Besides, GBDT classification is much longer than GBDT regression because of two overheads. One is the one-vs-the-rest strategy, which trains $W * c$ trees. The other is the *secure softmax* computation on c encrypted predictions for every sample in the training dataset, which needs additional MPC conversions and secure computations.

7.3.2 Evaluation on Prediction Efficiency

Varying m. Figure 4g compares the prediction time per sample for varying m. Results show that the prediction time of Pivot-Enhanced is higher than Pivot-Basic, because the cost of secure comparisons is higher than the homomorphic



(a) Training time vs. m (b) Training time vs. n

Figure 5: Comparison with baselines

computations. Besides, the prediction time of Pivot-Basic increases faster than that of Pivot-Enhanced as m increases. The reason is that the communication round for distributed prediction in Pivot-Basic scales linearly with m; while in Pivot-Enhanced, the number of secure comparisons remains the same, the increasing of m only incurs slight overhead.

Varying h. Figure 4h compares the prediction time per sample for varying h. When $h = 2$, Pivot-Enhanced takes less prediction time because the number of internal nodes (i.e. secure comparisons) is very small. Pivot-Basic outperforms Pivot-Enhanced when $h \geq 3$ and this advantage increases as h increases for two reasons. Firstly, the number of internal nodes is proportional to $2^h - 1$. Secondly, as described in Figure 4g, the number of clients dominates the prediction time of Pivot-Basic; although the size of the prediction vector also scales to h, its effect is insignificant since the size is still very small, leading to stable performance.

7.3.3 *Comparison with Baseline Solution*

We compare the Pivot protocols with the baselines SPDZ-DT and NPD-DT. For NPD-DT, we report the training time for varying m and n in Figure 5a-5b, and the prediction time per sample for varying m and h in Figure 4g-4h. In all the evaluated NPD-DT experiments, the training time is less than 1 minute, and the prediction time is less than 1 ms. Nevertheless, the efficiency of NPD-DT is at the cost of data privacy. For SPDZ-DT, since it is not parallelized, we adopt the non-parallelized versions. We omit the comparison of prediction time, because the model prediction of SPDZ-DT is similar to that of Pivot-Enhanced. We compare with SPDZ-DT for varying m and n.

Varying m. Figure 5a shows the comparison for varying m. When $m = 2$, Pivot-Enhanced and SPDZ-DT achieve similar performance. However, the training time of SPDZ-DT increases much faster as m increases because almost every secure computation in SPDZ-DT requires communication among all clients while most computations in Pivot protocols can be executed locally. We can notice that Pivot-Basic and Pivot-Enhanced can achieve up to about 19.8x and 4.5x speedup over SPDZ-DT, respectively.

Varying n. Figure 5b shows the comparison for varying n. Both Pivot-Enhanced and SPDZ-DT scale linearly to n and SPDZ-DT increases more quickly than Pivot-Enhanced. When n is small (e.g., $n = 5K$), the three algorithms achieve almost the same performance. While when $n = 200K$, Pivot-Basic and Pivot-Enhanced are able to achieve about 37.5x and 1.8x speedup over SPDZ-DT.

8. FURTHER PROTECTIONS

This section extends Pivot to account for malicious adversaries (in Section 8.1), and to incorporate differential privacy for enhanced protection (in Section 8.2).

8.1 Extension to Malicious Model

In the malicious model, an adversary may arbitrarily deviate from the protocol to launch attacks. To prevent the malicious behaviors, a typical solution is to let each client prove that she executes the specified protocol on the correct data step by step. If a client deviates from the protocol or uses incorrect data in any step, other clients should be able to detect it and abort the execution.

For this purpose, we extend Pivot using zero-knowledge proofs (ZKP) [22, 24] and SPDZ with authenticated shares [26, 41]. Specifically, ZKP enables a prover to prove to a verifier that a specific statement is true, without conveying any secret information. We adopt the the Σ-protocol [24] for ZKP, and mainly utilize three of its building blocks: proof of plaintext knowledge (POPK) [22], proof of plaintext-ciphertext multiplication (POPCM) [22], and proof of homomorphic dot product (POHDP) [75]. Meanwhile, SPDZ ensures malicious security using the information-theoretic message authentication code (MAC) [26,41], which also supports the secure computation building blocks that we mentioned in Section 2.2. Specifically, for a value $a \in \mathbb{Z}_q$ in SPDZ, its authenticated secretly shared form is denoted as $\langle a \rangle = (\langle a \rangle_1, \cdots, \langle a \rangle_m, \langle \delta \rangle_1, \cdots, \langle \delta \rangle_m, \langle \Delta \rangle_1, \cdots, \langle \Delta \rangle_m)$. Client i is given the random share $\langle a \rangle_i$, the random MAC share $\langle \delta \rangle_i$, and the fixed MAC key share $\langle \Delta \rangle_i$, such that the MAC relation $\delta = a \cdot \Delta$ holds. The MAC-related shares ensure that no client can modify $\langle a \rangle_i$ without being detected. In what follows, we discuss how we extend the classification tree training basic protocol (from Section 4.1) to handle malicious adversaries; the extensions of other protocols are similar.

Commitment. Before training, each client commits its local data by encrypting and broadcasting it to other clients, e.g., the pre-computed split indicator vectors $\boldsymbol{v}_l$ and $\boldsymbol{v}_r$ for each split, and the label indicator vector $\boldsymbol{\beta}_k$ of each class k that belongs to the super client. Each client uses POPK to prove that she knows the plaintext of the committed data (e.g., $[\boldsymbol{v}_l]$, $[\boldsymbol{v}_r]$, $[\boldsymbol{\beta}_k]$). These commitments will be used for ZKP verification during the whole process.

Model training. First, the super client initializes $[\boldsymbol{\alpha}]$ with $[1]$ and broadcasts it. Other clients can then verify its correctness by invoking threshold decryption since the initial $\boldsymbol{\alpha}$ is public. In the local computation step, the super client computes $[\boldsymbol{\gamma}_k]$ using an element-wise homomorphic multiplication between $\boldsymbol{\beta}_k$ and $[\boldsymbol{\alpha}]$ and broadcasts it, and proves its correctness using POPCM. In addition, each client proves that she computes the encrypted statistics (e.g., $[n_l] = \boldsymbol{v}_l \odot [\boldsymbol{\alpha}]$) correctly using POHDP. In the MPC computation step, we modify Algorithm 1 for malicious security by letting each client i do the following [22, 75]:

- Broadcast $[r_i]$ together with POPK (Line 3);
- Compute $[e]$ and invoke threshold decryption (Line 4);
- Broadcast $[x_i]$ together with POPK for committing her own share (Lines 6-8).

In addition, before computing with SPDZ, the clients verify that the authenticated shares are valid and match the converted encrypted value [75]. The remainder of SPDZ computations are malicious secure, and the best split identifier can be found and revealed. In the model update step, the client who has the best split computes $[\boldsymbol{\alpha}_l]$ (resp. $[\boldsymbol{\alpha}_r]$) using an element-wise homomorphic multiplication between $\boldsymbol{v}_l$ (resp. $\boldsymbol{v}_r$) and $[\boldsymbol{\alpha}]$, which can be proved by POPCM. The

pruning condition check and leaf label computation can be proved in a similar manner. If any ZKP or MAC is incorrect, the protocol is aborted.

8.2 Incorporating Differential Privacy

We can incorporate differential privacy (DP) [20, 28, 37, 38, 60, 66] to provide further protection, ensuring that the released model (even in the plaintext form) leaks limited information about each individual's private data in the training dataset. In a nutshell, a computation is differentially private if the probability of producing a given output is not highly dependent on whether a particular sample is included in the input dataset [28, 60]. Formally, a function f satisfies ϵ-DP, if for any two datasets D and D' differing in a single sample and any output O of f, we have

$$\Pr[f(D) \in O] \leq e^{\epsilon} \cdot \Pr[f(D') \in O]. \quad (8)$$

The parameter ϵ is the privacy budget that controls the tradeoff between the accuracy of f and the degree of privacy protection that f offers.

In our case, f trains a tree model with multiple iterations, where a node is built in each iteration. Existing work [29] has presented DP algorithms for such model training, and it works by making the following three steps noisy:

1. When checking whether the current node ρ should be split, it uses a noisy version of the sample number on ρ;
2. When choosing a split for ρ, it selects the split from a probability distribution over all possible splits, instead of choosing the one with the maximum impurity gain;
3. When deciding the leaf label for a leaf node ρ, for each label class $k \in K$, it uses a noisy version of the number of samples on ρ that are labeled k, instead of using the exact number.

We incorporate the above DP solution into Pivot by implementing the three noisy steps in an MPC manner. In what follows, we briefly explain the implementation of classification tree training in our basic protocol.

On the one hand, for Step 1 mentioned above, the clients first compute an exact encrypted sample number $[\bar{n}]$ by homomorphicly aggregating $[\boldsymbol{\alpha}]$, and convert it into secretly shared $\langle \bar{n} \rangle$ (see Section 4.1). Then, the clients jointly add a secretly shared noise to $\langle \bar{n} \rangle$, ensuring that Step 1 is noisy. The noise should be i.i.d. and follow a Laplace distribution whose parameter is public and related to ϵ, according to the *Laplace mechanism* [29]. To this end, the clients sample a secretly shared uniformly random value within a specific range using the primitive in SPDZ [3, 26]; and transform it into a secretly shared noise that follows a given Laplace distribution using *inverse transform sampling* [27, 64]. The transformation will be executed by SPDZ computations, such that the plaintext of the noise is not revealed to any client. Following the same approach, the clients can make Step 3 noisy.

On the other hand, for Step 2, the clients first compute the secretly shared impurity gains for the set S of all possible splits, as described in Section 4.1. Then, the clients need to securely sample a split from a probability distribution over S, such that each split $\tau \in S$ has a probability $\frac{\exp(gain_\tau \cdot \zeta)}{\sum_{\tau' \in S} \exp(gain_{\tau'} \cdot \zeta)}$ in the secretly shared form, where $gain_\tau$ is the impurity gain of τ, and ζ is a public parameter that relates to ϵ. This sampling approach is referred to as the *exponential mechanism* [29]. To securely perform this sampling procedure, the clients construct $|S|$ sub-intervals within $(\langle 0 \rangle, \langle 1 \rangle)$ according to the secretly shared cumulative distribution of the above probabilities. Next, they sample a secretly shared value uniformly at random from $(\langle 0 \rangle, \langle 1 \rangle)$, and find the sub-interval that the value falls into using SPDZ computations. The sampled sub-interval follows the above probability distribution [27, 35], and thus satisfies the exponential mechanism. After that, the split that corresponds to the sampled sub-interval is revealed as the best split.

The integration of DP with the enhanced protocol is the same as the basic protocol. The resulting protocol provides an additional layer of protection as an adversary needs to reverse the concealed model before obtaining the differentially private decision tree.

9. RELATED WORK

The works most related to ours are [19, 40, 45, 61–63, 65] for privacy preserving vertical tree models. None of these solutions, however, achieve the same privacy guarantee as our solution. They disclose either the super client's labels [40, 65] or intermediate results [19, 45, 61–63], which compromise the client's data privacy.

Meanwhile, several general techniques are applicable to our problem, but they may suffer from privacy deficiency or inefficiency. Secure hardware (e.g., Intel SGX [46]) protects client's private data using secure enclaves [54, 74]. However, it relies on trusted third party and is vulnerable to side channel attacks [70]. Secure multiparty computation (MPC) [72] could provide a strong privacy guarantee, but training machine learning models using generic MPC frameworks is extremely inefficient [75]. The tailored MPC solutions (e.g., [51, 53]) based on non-colluding servers are problematic in practice since it is difficult to convince the clients. [75] also uses a hybrid framework of TPHE and MPC, but it mainly focuses on linear models in horizontal FL, while our work addresses tree-based models in vertical FL and further considers the privacy leakages after releasing the tree model.

Finally, there are a number of works on collaborative prediction [9, 21, 34, 44] that consists of two parties where the server holds the private model and the client holds private data. However, these solutions cannot be directly adopted in our problem. Although [19, 45] consider the same scenario as ours, their methods disclose the prediction path.

10. CONCLUSIONS

We have proposed Pivot, a privacy preserving solution with two protocols for vertical tree-based models. With the basic protocol, Pivot guarantees that no intermediate information is disclosed during the execution. With the enhanced protocol, Pivot further mitigates the possible privacy leakages occurring in the basic protocol. To our best knowledge, this is the first work that provides strong privacy guarantees for vertical tree-based models. The experimental results demonstrate Pivot achieves accuracy comparable to non-private algorithms and is highly efficient.

ACKNOWLEDGEMENTS

We thank Xutao Sun for his early contribution to this work. This research is supported by Singapore Ministry of Education Academic Research Fund Tier 3 under MOEs official grant number MOE2017-T3-1-007.

11. REFERENCES

[1] California consumer privacy act. bill no. 375 privacy: personal information: businesses. https://leginfo.legislature.ca.gov/. 2018-06-28.

[2] Regulation (eu) 2016/679 of the european parliament and of the council of 27 april 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing directive 95/46/ec (general data protection regulation). oj, 2016-04-27.

[3] data61/mp-spdz: Versatile framework for multiparty computation. https://github.com/data61/mp-spdz. Accessed: 2019-11-25.

[4] T. Araki, A. Barak, J. Furukawa, M. Keller, Y. Lindell, K. Ohara, and H. Tsuchida. Generalizing the SPDZ compiler for other protocols. In *CCS*, pages 880–895, 2018.

[5] A. T. Azar and S. M. El-Metwally. Decision tree classifiers for automated medical diagnosis. *Neural Computing and Applications*, 23(7-8):2387–2403, 2013.

[6] D. Beaver. Efficient multiparty protocols using circuit randomization. In *CRYPTO*, 1991.

[7] A. N. Bhagoji, S. Chakraborty, P. Mittal, and S. B. Calo. Analyzing federated learning through an adversarial lens. In *ICML*, pages 634–643, 2019.

[8] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth. Practical secure aggregation for privacy-preserving machine learning. In *CCS*, pages 1175–1191, 2017.

[9] R. Bost, R. A. Popa, S. Tu, and S. Goldwasser. Machine learning classification over encrypted data. In *NDSS*, 2015.

[10] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[11] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. Classification and regression trees. 1984.

[12] L. M. Candanedo, V. Feldheim, and D. Deramaix. Data driven prediction models of energy use of appliances in a low-energy house. *Energy and Buildings*, 140:81–97, 2017.

[13] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–145, 2001.

[14] S. Cao, X. Yang, C. Chen, J. Zhou, X. Li, and Y. Qi. Titant: Online real-time transaction fraud detection in ant financial. *PVLDB*, 12(12):2082–2093, 2019.

[15] O. Catrina and S. de Hoogh. Improved primitives for secure multiparty integer computation. In *SCN*, pages 182–199, 2010.

[16] O. Catrina and A. Saxena. Secure computation with fixed-point numbers. In *FC*, pages 35–50, 2010.

[17] H. Chen, K. Laine, and P. Rindal. Fast private set intersection from homomorphic encryption. In *CCS*, pages 1243–1255, 2017.

[18] T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In *SIGKDD*, pages 785–794, 2016.

[19] K. Cheng, T. Fan, Y. Jin, Y. Liu, T. Chen, and Q. Yang. Secureboost: A lossless federated learning framework. *CoRR*, abs/1901.08755, 2019.

[20] A. R. Chowdhury, C. Wang, X. He, A. Machanavajjhala, and S. Jha. Crypt?: Crypto-assisted differential privacy on untrusted servers. In *SIGMOD*, pages 603–619, 2020.

[21] M. D. Cock, R. Dowsley, C. Horst, R. S. Katti, A. C. A. Nascimento, W. Poon, and S. Truex. Efficient and private scoring of decision trees, support vector machines and logistic regression models based on pre-computation. *IEEE TDSC*, 16(2):217–230, 2019.

[22] R. Cramer, I. Damgård, and J. B. Nielsen. Multiparty computation from threshold homomorphic encryption. In *EUROCRYPT*, pages 280–299, 2001.

[23] R. Cramer, I. B. Damgrd, and J. B. Nielsen. Secure multiparty computation and secret sharing. 2015.

[24] I. Damgård. On σ-protocol. In *Lecture Notes*, 2010.

[25] I. Damgård and M. Jurik. A generalisation, a simplification and some applications of paillier's probabilistic public-key system. In *Public Key Cryptography*, pages 119–136, 2001.

[26] I. Damgård, V. Pastro, N. P. Smart, and S. Zakarias. Multiparty computation from somewhat homomorphic encryption. In *CRYPTO*, pages 643–662, 2012.

[27] L. Devroye. *Non-Uniform Random Variate Generation*. Springer, 1986.

[28] C. Dwork and A. Roth. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3-4):211–407, 2014.

[29] A. Friedman and A. Schuster. Data mining with differential privacy. In *SIGKDD*, pages 493–502, 2010.

[30] J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: a statistical view of boosting. *Annals of Statistics*, 28:2000, 1998.

[31] J. H. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29:1189–1232, 2000.

[32] F. Fu, J. Jiang, Y. Shao, and B. Cui. An experimental evaluation of large scale GBDT systems. *PVLDB*, 12(11):1357–1370, 2019.

[33] C. Ge, I. F. Ilyas, and F. Kerschbaum. Secure multi-party functional dependency discovery. *PVLDB*, 13(2):184–196, 2019.

[34] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. E. Lauter, M. Naehrig, and J. Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *ICML*, pages 201–210, 2016.

[35] J. Goldstick. Introduction to statistical computing, statistics 406, notes - lab 5. *Lecture Notes, Department of Statistics*, 2009.

[36] J. F. Hart. *Computer Approximations*. Krieger Publishing Co., Inc., Melbourne, FL, USA, 1978.

[37] M. Hay, V. Rastogi, G. Miklau, and D. Suciu. Boosting the accuracy of differentially private histograms through consistency. *PVLDB*, 3(1):1021–1032, 2010.

[38] X. He, A. Machanavajjhala, C. J. Flynn, and D. Srivastava. Composing differential privacy and secure computation: A case study on scaling private record linkage. In *CCS*, pages 1389–1406, 2017.

[39] B. Hitaj, G. Ateniese, and F. Pérez-Cruz. Deep models under the GAN: information leakage from collaborative deep learning. In *CCS*, pages 603–618, 2017.

[40] Y. Hu, D. Niu, J. Yang, and S. Zhou. FDML: A collaborative machine learning framework for distributed features. In *SIGKDD*, pages 2232–2240, 2019.

[41] M. Keller, E. Orsini, and P. Scholl. MASCOT: faster malicious arithmetic secure computation with oblivious transfer. In *CCS*, pages 830–842, 2016.

[42] Y. Lindell and B. Pinkas. Privacy preserving data mining. In *CRYPTO*, pages 36–54, 2000.

[43] Y. Lindell and B. Pinkas. Secure multiparty computation for privacy-preserving data mining. *J. Priv. Confidentiality*, 1(1), 2009.

[44] J. Liu, M. Juuti, Y. Lu, and N. Asokan. Oblivious neural network predictions via minionn transformations. In *CCS*, pages 619–631, 2017.

[45] Y. Liu, Y. Liu, Z. Liu, J. Zhang, C. Meng, and Y. Zheng. Federated forest. *CoRR*, abs/1905.10053, 2019.

[46] F. McKeen, I. Alexandrovich, A. Berenzon, C. V. Rozas, H. Shafi, V. Shanbhogue, and U. R. Savagaonkar. Innovative instructions and software model for isolated execution. In *HASP*, page 10, 2013.

[47] H. B. McMahan, E. Moore, D. Ramage, and B. A. y Arcas. Federated learning of deep networks using model averaging. *CoRR*, abs/1602.05629, 2016.

[48] H. B. McMahan, D. Ramage, K. Talwar, and L. Zhang. Learning differentially private recurrent language models. In *ICLR*, 2018.

[49] C. A. Meadows. A more efficient cryptographic matchmaking protocol for use in the absence of a continuously available third party. In *IEEE S&P*, pages 134–137, 1986.

[50] L. Melis, C. Song, E. D. Cristofaro, and V. Shmatikov. Exploiting unintended feature leakage in collaborative learning. In *IEEE S&P*, pages 691–706, 2019.

[51] P. Mohassel and Y. Zhang. Secureml: A system for scalable privacy-preserving machine learning. In *IEEE S&P*, pages 19–38, 2017.

[52] S. Moro, P. Cortez, and P. Rita. A data-driven approach to predict the success of bank telemarketing. *Decision Support Systems*, 62:22–31, 2014.

[53] V. Nikolaenko, U. Weinsberg, S. Ioannidis, M. Joye, D. Boneh, and N. Taft. Privacy-preserving ridge regression on hundreds of millions of records. In *IEEE S&P*, pages 334–348, 2013.

[54] O. Ohrimenko, F. Schuster, C. Fournet, A. Mehta, S. Nowozin, K. Vaswani, and M. Costa. Oblivious multi-party machine learning on trusted processors. In *USENIX Security Symposium*, pages 619–636, 2016.

[55] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT*, pages 223–238, 1999.

[56] B. Pinkas, T. Schneider, G. Segev, and M. Zohner. Phasing: Private set intersection using permutation-based hashing. In *USENIX Security Symposium*, pages 515–530, 2015.

[57] B. Pinkas, T. Schneider, and M. Zohner. Scalable private set intersection based on OT extension. *ACM Trans. Priv. Secur.*, 21(2):7:1–7:35, 2018.

[58] J. R. Quinlan. Induction of decision trees. *Mach. Learn.*, 1(1):81–106, Mar. 1986.

[59] J. R. Quinlan. C4.5: Programs for machine learning. Morgan Kaufmann Publishers Inc., 1993.

[60] R. Shokri and V. Shmatikov. Privacy-preserving deep learning. In *CCS*, pages 1310–1321, 2015.

[61] J. Vaidya and C. Clifton. Privacy-preserving decision trees over vertically partitioned data. In *DBSec*, pages 139–152, 2005.

[62] J. Vaidya, C. Clifton, M. Kantarcioglu, and A. S. Patterson. Privacy-preserving decision trees over vertically partitioned data. *TKDD*, 2(3):14:1–14:27, 2008.

[63] J. Vaidya, B. Shafiq, W. Fan, D. Mehmood, and D. Lorenzi. A random decision tree framework for privacy-preserving data mining. *IEEE TDSC*, 11(5):399–411, 2014.

[64] C. R. Vogel. *Computational Methods for Inverse Problems*, volume 23 of *Frontiers in Applied Mathematics*. SIAM, 2002.

[65] K. Wang, Y. Xu, R. She, and P. S. Yu. Classification spanning private databases. In *AAAI*, pages 293–298, 2006.

[66] N. Wang, X. Xiao, Y. Yang, J. Zhao, S. C. Hui, H. Shin, J. Shin, and G. Yu. Collecting and analyzing multidimensional data with local differential privacy. In *ICDE*, pages 638–649, 2019.

[67] D. J. Wu, J. Zimmerman, J. Planul, and J. C. Mitchell. Privacy-preserving shortest path computation. In *NDSS*, 2016.

[68] Y. Wu, K. Wang, R. Guo, Z. Zhang, D. Zhao, H. Chen, and C. Li. Enhanced privacy preserving group nearest neighbor search. *IEEE TKDE*, 2019.

[69] Y. Wu, K. Wang, Z. Zhang, W. Lin, H. Chen, and C. Li. Privacy preserving group nearest neighbor search. In *EDBT*, pages 277–288, 2018.

[70] Y. Xu, W. Cui, and M. Peinado. Controlled-channel attacks: Deterministic side channels for untrusted operating systems. In *IEEE S&P*, pages 640–656, 2015.

[71] Q. Yang, Y. Liu, T. Chen, and Y. Tong. Federated machine learning: Concept and applications. *ACM TIST*, 10(2):12:1–12:19, 2019.

[72] A. C. Yao. Protocols for secure computations (extended abstract). In *FOCS*, pages 160–164, 1982.

[73] I. Yeh and C. Lien. The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients. *Expert Syst. Appl.*, 36(2):2473–2480, 2009.

[74] W. Zheng, A. Dave, J. G. Beekman, R. A. Popa, J. E. Gonzalez, and I. Stoica. Opaque: An oblivious and encrypted distributed analytics platform. In *NSDI*, pages 283–298, 2017.

[75] W. Zheng, R. A. Popa, J. E. Gonzalez, and I. Stoica. Helen: Maliciously secure coopetitive learning for linear models. In *IEEE S&P*, pages 915–929, 2019.