Privet: A Privacy-Preserving Vertical Federated Learning Service for Gradient Boosted Decision Tables

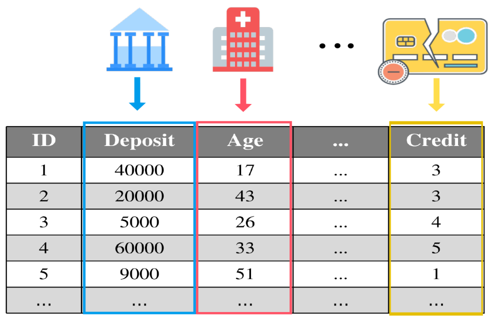
Yifeng Zheng, Shuangqing Xu, Songlei Wang, Yansong Gao, *Senior Member, IEEE*, and Zhongyun Hua, *Senior Member, IEEE*

***Abstract*—Vertical federated learning (VFL) has recently emerged as an appealing distributed paradigm empowering multi-party collaboration for training high-quality models over vertically partitioned datasets. Gradient boosting has been popularly adopted in VFL, which builds an ensemble of weak learners (typically decision trees) to achieve promising prediction performance. Recently there have been growing interests in using decision table as an intriguing alternative weak learner in gradient boosting, due to its simpler structure, good interpretability, and promising performance. In the literature, there have been works on privacy-preserving VFL for gradient boosted decision trees, but no prior work has been devoted to the emerging case of decision tables. Training and inference on decision tables are different from that in the case of generic decision trees, not to mention gradient boosting with decision tables in VFL. In light of this, we design, implement, and evaluate Privet, the first system framework enabling privacy-preserving VFL service for gradient boosted decision tables. Privet delicately builds on lightweight cryptography and allows an arbitrary number of participants holding vertically partitioned datasets to securely train gradient boosted decision tables. Extensive experiments over several real-world datasets and synthetic datasets demonstrate that Privet achieves promising performance, with utility comparable to plaintext centralized learning.**

***Index Terms*—Vertical federated learning service, multi-party collaboration, gradient boosting, decision table, privacy preservation.**

I. Introduction

FEDERATED learning (FL) has recently emerged as a fascinating distributed machine learning paradigm that greatly empowers multi-party collaboration for mining value over data federation [1], [2], [3], [4]. It allows distributed individual training datasets to be kept locally, and only intermediate outputs from the training algorithm are shared out for aggregation. According to how data is distributed among the participants in FL, there are two types of FL: horizontal federated learning (HFL) [5], [6] and vertical federated learning (VFL) [7], [8]. HFL addresses the scenario where the participants share the same feature space but hold disjoint sets of samples/instances, which generally suits the case that participants are individual customers. In contrast, VFL targets the scenario where each participant has the same set of samples/instances yet owns data for different features, which is more common when the participants are business organizations/enterprises. For example, as illustrated in Fig. 1, the participants hold datasets that have the same row indexes (corresponding to the same set of instances) but different non-overlapping column indexes (corresponding to different features). In this article, we focus on the VFL setting, which has received increasing attentions in the collaboration of different business organizations/enterprises in recent years [8], [9].



ID	Deposit	Age	...	Credit
1	40000	17	...	3
2	20000	43	...	3
3	5000	26	...	4
4	60000	33	...	5
5	9000	51	...	1
...	...	...	...	...

Fig. 1. Illustration of data partitioning in the VFL setting.

For model training in the VFL setting, the gradient boosting technique has received wide attentions [7], [8], [9], [10], [11] and has seen popular adoption for empowering a wide range of fields, such as web search ranking, online advertisement, and fraud detection [12], [13], [14]. Gradient boosting builds an ensemble of weak learners, which are typically (generic) decision trees, to achieve promising prediction performance. While decision tree is usually used as the weak learner in gradient boosting, in recent years there has been a fast-growing trend to use decision

Manuscript received 6 January 2023; revised 21 April 2023; accepted 22 May 2023. Date of publication 25 May 2023; date of current version 8 October 2023. This work was supported in part by the Guangdong Basic and Applied Basic Research Foundation under Grants 2021A1515110027, 2023A1515010714, and 2021A1515011406, in part by the Shenzhen Science and Technology Program under Grants RCBS20210609103056041 and JCYJ20220531095416037, in part by by the National Natural Science Foundation of China under Grants 62002167 and 62071142, and in part by the Natural Science Foundation of JiangSu Province under Grant BK20200461. Recommended for acceptance by H. Karatza. *(Corresponding author: Zhongyun Hua.)*

Yifeng Zheng, Shuangqing Xu, Songlei Wang, and Zhongyun Hua are with the School of Computer Science and Technology, Harbin Institute of Technology, Shenzhen, Guangdong 518055, China (e-mail: yifeng.zheng@hit.edu.cn; shuangqing.xu@outlook.com; songlei.wang@outlook.com; huazhongyun@hit.edu.cn).

Yansong Gao is with the Data61, CSIRO, Sydney, NSW 2720, Australia (e-mail: gao.yansong@hotmail.com).

Digital Object Identifier 10.1109/TSC.2023.3279839

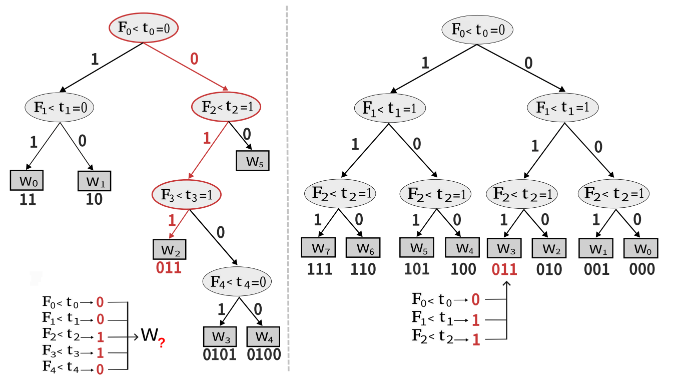


Fig. 2. Comparison of a generic decision tree and an oblivious tree in inference.

table [15] as an intriguing alternative [16], [17], [18], [19]. Many works [16], [17], [20], [21] have shown that gradient boosted *decision tables* yields promising performance on various tasks and achieves great inference efficiency over generic decision trees. In addition, some famous open-source gradient boosting libraries [17], [22] have also recently provided the support for using decision table as the weak learner in gradient boosting.

As demonstrated in Fig. 2, a D-dimensional decision table at a high level consists of D Boolean tests and 2^D output values. It can also be treated as a *special* full binary decision tree, called *oblivious tree*. In contrast with generic decision tree which has different Boolean tests at different internal nodes at the same level, the internal nodes at the same level of an oblivious tree share the same Boolean test defined with the same feature and threshold. Despite the similarly equivalent tree structure, it is worth noting that the algorithm for training oblivious tree is different from that for generic decision tree [23], [24]. Specifically, decision trees are typically trained through recursive algorithms [23], [24], while decision tables are trained through iterative algorithms following the top-down construction [15], [16]. Given tree depth D, the shape of a generic decision tree is uncertain because it needs to process samples associated with the current node to determine whether to split this node. In contrast, we cannot recursively build a decision table because all the samples in the dataset need to be processed to select the optimal split for each level of the decision table. Besides, given depth D, the shape of an oblivious tree is fixed and the number of operations like node splitting and output value calculation is also fixed. In addition, the inference process on an oblivious tree is also different from that on a generic decision tree [16], [18] (see Section III-A for more detailed discussion).

In the literature, while there have been several studies on privacy-preserving VFL with gradient boosted decision trees (GBDT) [7], [8], [10], no prior work has explored privacy-preserving VFL with gradient boosted decision tables. As mentioned above, even training and inference on decision table are different from the case of generic decision tree, not to mention gradient boosting with decision table as the weak learner in the VFL setting. Therefore, these prior works cannot be directly applied to support privacy-preserving training and inference of gradient boosted decision tables in VFL. In addition, it is noted that these prior works are also confronted with limitations such as exposing sensitive intermediate results (e.g., sum of gradients) [7], [10], supporting training only among two participants [8] (see Section II for more detailed discussion).

In light of the above, we propose Privet, which, to our best knowledge, is the first system framework enabling privacy-preserving VFL service for training *gradient boosted decision tables* over distributed datasets. Privet ambitiously supports an arbitrary number of participants to collaboratively train gradient boosted decision tables, while allowing them to keep their data locally and offering strong protection on the sensitive intermediate outputs throughout the training process. Privet builds on lightweight secret sharing techniques to develop customized protocols securely realizing the key components required by training gradient boosted decision tables in the VFL setting.

Specifically, through an in-depth examination on the training process of gradient boosted decision tables, we manage to decompose the holistic secure design in the VFL setting into the design of a series of secure components run in a distributed manner among the participants, including secure node splitting, secure Sigmoid evaluation, secure discretization, and secure distributed decision table inference. The delicate synergy of these secure components leads to the holistic protocol of Privet for privately training gradient boosted decision tables in the VFL setting. Through the customized secure protocol, Privet outputs gradient boosted decision tables that are distributed among the involved participants, where each participant only holds a part of the model. Subsequently, secure inference on the ensemble of learned decision tables can also be well supported in a distributed manner among the participants. We implement and evaluate Privet's protocols extensively over several real-world datasets as well as synthetic datasets. The results demonstrate that Privet presents promising performance in computation and communication. Meanwhile, the utility of the trained models in Privet is comparable to that in the plaintext centralized learning setting.

We highlight our contributions as follows.

- We present Privet, which, to our best knowledge, is the first system framework enabling privacy-preserving VFL service for gradient boosted decision tables. Privet allows an arbitrary number of participants holding vertically partitioned distributed datasets to securely train gradient boosted decision tables in a distributed manner, offering strong protection for sensitive individual data as well as for intermediate outputs.
- We devise a series of tailored secure components based on lightweight secret sharing techniques that run in a distributed manner among multiple participants with promising efficiency and utility, catering for the computation required by securely training gradient boosted decision tables in the VFL setting.
- We make an implementation of the proposed protocols and conduct an extensive evaluation over three real-world public datasets and three synthetic datasets. The experiment results demonstrate that Privet has promising performance, achieving model utility comparable to plaintext centralized learning.

The rest of this article is organized as follows. Section II discusses the related work. Section III introduces some preliminaries. Section IV gives a system overview. Section V presents the design of Privet. The security analysis is presented in Section VI, followed by the experiments in Section VII. Section VIII concludes the whole article.

II. Related Work

Securely Learning Gradient Boosted Decision Trees Under HFL: Due to the problems of data isolation and data privacy, FL has emerged as a new privacy-preserving machine learning paradigm. Several existing works [5], [6], [25] have been focused on privacy-preserving gradient boosted decision trees (GBDT) under the HFL setting, which assume that data are horizontally partitioned between participants. Among them, the work [6] rely on use of secure aggregation and differential privacy to provide a privacy guarantee. The work [25] leverages secure hardware [26] to build private GBDT under HFL. Different from these works, our work targets privacy-preserving gradient boosting systems under the VFL setting.

Securely Learning Gradient Boosted Decision Trees Under VFL: To cater for the need to collaboratively build models between different organizations that hold data on the same set of samples but for different features, VFL has received increasing attention in recent years. The works [7], [8], [10] consider vertical federated gradient boosted decision trees, which are most related to ours. In particular, SecureBoost [10] is the first work on privacy-preserving GBDT over vertically partitioned data, which uses homomorphic encryption to preserve data privacy. However, it has limited security guarantee because intermediate information (e.g., the sum of gradients in a bucket) is revealed during the training process. Moreover, homomorphic encryption involves heavy cryptographic operations and requires large memory, which results in low training efficiency. The works [7], [8] improve SecureBoost [10] in terms of efficiency via multi-party computation (MPC) techniques. Specifically, the work [8] proposes a secure GBDT system leveraging the additive secret sharing technique [27]. However, their proposed system is only designed for the two-party setting. Xie et al. [7] deal with the issue to support secure multi-party training. However, since they adopt large-scale matrix multiplication in the secret sharing domain to discretize secret-shared gradients into buckets, their scheme requires more communication and computation overhead compared to [8]. Moreover, the design in [7] has notable privacy leakages, e.g., the intermediate inference results of all training samples are leaked to the participant who holds the label set because it relies on this participant to conduct inference.

We also note that all these works [7], [8], [10] are aimed at supporting secure training and inference for gradient boosting with generic decision trees under VFL. In recent years, the gradient boosted decision table technique has seen rapidly growing adoption in various applications, such as learning to rank (LTR) [18], [21], [28], recommendation systems [16], [29], and medical diagnosis [30], [31]. Although the training of decision tree and decision table has some similarities, e.g., both of them need permutation protocols, their learning algorithms are different inherently. Thus the works [7], [8], [10] cannot directly support secure gradient boosting over decision tables under VFL. In comparison with them, Privet focuses on securely supporting privacy-preserving VFL for gradient boosted decision tables. In addition, Privet departs from them by achieving comparable utility to plaintext, concealing intermediate information for strong privacy, and supporting an arbitrary number of participants.

Secure Decision Tree Learning Supporting Both Horizontally and Vertically Partitioned Data: There are some works [32], [33], [34] which can support secure decision tree learning on both horizontally and vertically partitioned data in an *outsourcing* setting. Specifically, the work [32] considers a setting where data owners secret-share all their data among three servers and designs a protocol to enable the three servers to securely perform an adapted C4.5 decision tree learning algorithm. The work [34] proposes protocols to train decision trees for the *Random Forest* model, which similarly considers a setting where the data owners secret-share all their data among two extra non-colluding computing parties. In [33], Deforth et al. focus on building private gradient boosted decision trees and consider a scenario where data owners secret-share their data among a set of computing parties which may also be an extra set of servers. In contrast with these works that outsource the data and computation, Privet does not require such an extra set of non-colluding servers which may not be an easy assumption to meet in practice. Meanwhile, Privet allows the raw data of each participant to stay local throughout the whole training process, fitting the salient feature of FL.

III. Preliminaries

A. Decision Table

Consider a dataset $\mathcal{D}$ consisting of N samples $\{\mathbf{x}_i, y_i\}$ for $i = 0, \ldots, N-1$, where $\mathbf{x}_i = (x_{i1}, \ldots, x_{iJ})$ is a J-dimensional tuple and y_i is the label of the i-th sample. The j-th element of $\mathbf{x}_i$ is the value of an input attribute X_j. A D-dimensional decision table consists of D Boolean tests and 2^D output values. A Boolean test is of the form $X_j < t$, which outputs 1 if the j-th element in a given input tuple is less than a threshold t and 0 otherwise.

As illustrated in Fig. 2, a D-dimensional decision table is equivalent to a full binary tree with $D+1$ levels, where each internal node from the 0-th level (for the root node) to the $(D-1)$-th level has a Boolean test; each edge is assigned the outcome of its source node's test and each leaf node at the D-th level is associated with an output value. Such equivalent tree is called *oblivious tree*, because all internal nodes at the same level share the same test, as opposed to generic decision trees that have different tests at the same level. More specifically, the test at the d-th level of an oblivious tree could be represented as $F_d < t_d$, where $d \in [0, D-1]$, the split feature $F_d \in \{X_1, \ldots, X_J\}$, and t_d is the split threshold. The special structure of oblivious tree results in its different training and inference methods from non-oblivious trees like CART [23]. In [15], Kohavi et al. first introduce a top-down construction of oblivious trees and use information gain as the evaluation metric to find the optimal test at each level. Different evaluation metrics

Algorithm 1: Training an Oblivious Tree.

```
Input: A training dataset $\mathcal{D}$.
Output: An oblivious decision tree having $D$ tests and $2^D$
  output values.
 1: $\mathcal{S}^0 = \{\mathcal{D}\}$.
 2: for $d \in [0, D-1]$ do
 3:   $\mathcal{S}^{d+1} = \{\}$.
 4:   Optimal test $F_d < t_d \leftarrow$ find_split.
 5:   for $\mathcal{V}$ in $\mathcal{S}^d$ do
 6:     Split $\mathcal{V}$ into $\mathcal{V}_{F_d<t_d}$, $\mathcal{V}_{F_d\geq t_d}$ according to the
        optimal test and add these two sets to $\mathcal{S}^{d+1}$.
 7:     Create a node for each set in $\mathcal{S}^{d+1}$ and connect it
        to its parent node.
 8:   end for
 9: end for
10: Calculate output values for the $2^D$ leaf nodes at the
    $D$-th level, respectively.
```

are used in later studies, like mean squared error (MSE) [16] and Newton's method [17].

We follow the top-down construction in [15], [16] to train oblivious trees. Algorithm 1 shows the process of training an oblivious tree, which produces D tests and 2^D output values. The learning algorithm starts from the 0-th level and builds an oblivious tree level by level iteratively. Given a test $X_j < t$, we define $\mathcal{D}_{X_j<t} = \{(\mathbf{x}, y) \in \mathcal{D} \mid \mathbf{x}(X_j) < t\}$, $\mathcal{D}_{X_j \geq t} = \mathcal{D} \backslash \mathcal{D}_{X_j<t}$. We also apply this notation to subsets $\mathcal{V} \subseteq \mathcal{D}$. Let $\mathcal{S}^l$ denote the set of $\mathcal{D}$'s subsets at the l-th level, where $l \in [0, D]$. At level 0, the training dataset $\mathcal{D}$ is associated with the root node and $\mathcal{S}^0 = \{\mathcal{D}\}$ (line 1). Once an optimal test $F_0 < t_0$ at this level is found through the routine find_split (line 4), $\mathcal{D}$ is partitioned into two subsets $\mathcal{D}_{F_0<t_0}, \mathcal{D}_{F_0\geq t_0}$ according to it. After that, $\mathcal{D}_{F_0<t_0}, \mathcal{D}_{F_0\geq t_0}$ are added to $\mathcal{S}^1$ ($\mathcal{S}^1 = \{\mathcal{D}_{F_0<t_0}, \mathcal{D}_{F_0\geq t_0}\}$) and a new level is created (lines 6-7).

At level 1, an optimal test $F_1 < t_1$ is found and $\mathcal{D}_{F_0<t_0}, \mathcal{D}_{F_0\geq t_0}$ are each partitioned into two subsets according to $F_1 < t_1$. The same procedure is repeated until all the D tests are learned. In this way, the tree structure is kept full and symmetric, and we have $|\mathcal{S}^l| = 2^l$ at level l, where each set in $\mathcal{S}^l$ is associated with a node at this level. When reaching the D-th level, the output values will be calculated for the leaf nodes. Finally, an oblivious tree composed of D tests and 2^D output values is learned.

The optimal test at each level is found via the routine find_split by evaluating the candidate tests. Evaluation of the candidate tests can be made through different metrics. In Privet, we follow the popular second-order approximation method [17], [35] to evaluate tests because the decision tables in our work are trained sequentially for a gradient boosting system. Besides, the output values of decision tables can also be calculated following the gradient boosting theory, which will be introduced shortly in Section III-B.

As presented in Algorithm 1, training a decision table (oblivious tree) is an iterative process, while decision trees are typically trained through recursive algorithms [23], [24]. Given tree depth D, the shape of a generic decision tree is uncertain because it needs to process samples associated with the current node to judge whether to split this node. However, the shape of an oblivious tree is predetermined at a given dimension D. To select the optimal split at each level, all the samples in the dataset are required to be processed. Besides, the number of operations involved in training an oblivious tree, such as find_split and output value calculation, is fixed.

Decision table outperforms generic decision tree in inference efficiency significantly. As illustrated in Fig. 2, each leaf node of an oblivious tree (the right sub-figure in Fig. 2) corresponds to a Boolean sequence and the comparisons required by D tests could be parallelized. In contrast, inference in a regular decision tree is made by traversing the tree from the root node to a leaf node, which means the direction of the inference path after the current node depends on the test result of this node. Note that while the evaluation of each decision node in generic decision tree inference can be parallelized, it is still necessary to traverse the tree from the root node *sequentially* so as to identify the correct leaf node that produces the inference result. For example, as shown in the left sub-figure in Fig. 2, even if we parallelize the evaluation of each decision node, i.e., we obtain the sequence of test results [0,0,1,1,0] by evaluating the 0-th split to the 4-th split simultaneously, we cannot directly identify which leaf node is finally chosen using [0,0,1,1,0]. On the contrary, decision table inference is free of such sequential traversal [16], [17], [18]. As illustrated in the right sub-figure in Fig. 2, once the sequence comprised of Boolean test result at each level is obtained, the inference result can be immediately obtained because this Boolean sequence is also the identifier of a leaf node.

Additionally, it is noted that in gradient boosting systems, the number, size, and depth of generic decision trees are not necessarily smaller than decision tables when achieving the same accuracy because they are both weak learners and only require weak predictability. As reported in prior work [16], compared with gradient boosted decision trees with the number of trees $T = 50$ and tree depth $D = 7$, gradient boosted decision tables only requires depth $D = 6$ given the same number of oblivious trees $T = 50$ to achieve similar accuracy performance. Furthermore, it is noted that with the same depth D, a decision table only needs storage of D decision nodes (one for each level), while a generic decision tree may require storage of up to $2^D - 1$ decision nodes [16], [18].

B. Gradient Boosted Decision Tables

A gradient boosting system is built by training a set of weak learners sequentially based on the boosting algorithm [35], [36]. For the given dataset $\mathcal{D} = \{\mathbf{x}_i, y_i\}_{i=0}^{N-1}$, a gradient boosting system sums the inference results of T weak learners to produce the ultimate inference result for the i-th sample [36]: $\hat{y}_i^{(T)} = \sum_{t=1}^{T} f_t(\mathbf{x}_i)$, where f_t corresponds to the model of the t-th weak learner. In gradient boosted decision tables [16], [17], f_t corresponds to a decision table. A given sample will be classified into the leaf nodes in the decision tables according to the tests in them. Its ultimate inference result is calculated by summing up the output values associated with the corresponding leaf nodes.

The essence of gradient boosting algorithm comes from how it boosts the weak learners sequentially. After training $t-1$ weak learners, the t-th model f_t is needed to be trained and added to minimize the following objective function [35]:

$$\begin{aligned}\mathcal{L}^{(t)} &= \sum_{i=0}^{N-1} l(y_i, \hat{y}_i^{(t)}) + \Omega(f_t) \\ &= \sum_{i=0}^{N-1} l(y_i, \hat{y}_i^{(t-1)} + f_t(\mathbf{x}_i)) + \Omega(f_t),\end{aligned}$$

where l is a twice differentiable convex loss function that takes y_i, $\hat{y}_i^{(t)}$ as input, and outputs the loss. The regularization term $\Omega(f_t)$ is set following [35]. Friedman et al. [36] use second-order approximation to quickly approximate the objective function:

$$\tilde{\mathcal{L}}^{(t)} \simeq \sum_{i=0}^{N-1} \left[(l(y_i, \hat{y}^{(t-1)}) + g_i f_t(\mathbf{x}_i) + \frac{1}{2} h_i f_t^2(\mathbf{x}_i) \right] + \Omega(f_t), \tag{1}$$

where $g_i = \partial_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)})$, $h_i = \partial^2_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)})$ are the first and second-order gradients of the i-th sample. Typically, for regression problems, MSE is used as the loss function and the gradients are calculated as follows: $g_i = \hat{y}_i - y_i$ and $h_i = 1$ [7], [16]. When the problem is classification, a common choice is logistic loss and the gradients are calculated as follows: $g_i = p_i - y_i$ and $h_i = p_i \times (1-p_i)$, where $p_i = \text{Sigmoid}(\hat{y}_i)$ [37]. For a value $x \in \mathbb{R}$, the Sigmoid function is: $\text{Sigmoid}(x) = 1/(1+e^{-x})$. For the leaf node k, which is associated with a subset $\mathcal{V}^k \subset \mathcal{D}$, we define $\mathcal{I}^k = \{i | (\mathbf{x}_i, y_i) \in \mathcal{V}^k\}$ as its index set. This notation is also used to denote the index set associated with the internal node, e.g., we write $\mathcal{I}^q$ for node q. Then, after removing the constant terms, (1) can be rewritten as [35]:

$$\tilde{\mathcal{L}}^{(t)} = \sum_{k=0}^{L-1} \left[\left(\sum_{i \in \mathcal{I}^k} g_i \right) w_k + \frac{1}{2} \left(\sum_{i \in \mathcal{I}^k} h_i + \lambda \right) w_k^2 \right] + \gamma L \tag{2}$$

where w_k is the output value associated with the leaf node k, L is the number of leaf nodes in the tree, and λ, γ are hyper-parameters to control the regularization. When the tree stops growing, w_k and the minimum loss of the current tree are calculated by [35]:

$$w_k = -\frac{\sum_{i \in \mathcal{I}^k} g_i}{\sum_{i \in \mathcal{I}^k} h_i + \lambda}, \tag{3}$$

$$\tilde{\mathcal{L}}^{(t)} = -\frac{1}{2} \sum_{k=0}^{L-1} \frac{(\sum_{i \in \mathcal{I}^k} g_i)^2}{\sum_{i \in \mathcal{I}^k} h_i + \lambda} + \gamma L, \tag{4}$$

Eq. (4) can be used as the impurity function for evaluating the tests. In Privet, we follow the above theory to find optimal tests in decision table. Suppose we have learned b tests from the level 0 to the level $(b-1)$ of a D-dimensional decision table and we need to find an optimal test at level b. The nodes at the b-th level are numbered from 0 to $2^b - 1$ and the q-th node is associated with an index set $\mathcal{I}^q$. A candidate test $X_j < t$ will split the 2^b nodes at this level into 2^{b+1} nodes. Among all the candidate tests, the optimal test is the test that has the minimum score. The definition of score is defined as [38]:

$$Score = \sum_{q=0}^{2^b-1} \left(\mathcal{P}_{\mathcal{I}_L^q} + \mathcal{P}_{\mathcal{I}_R^q} \right), \tag{5}$$

where

$$\mathcal{P}_{\mathcal{I}} = -\frac{1}{2} \frac{\left(\sum_{i \in \mathcal{I}} g_i\right)^2}{\sum_{i \in \mathcal{I}} h_i + \lambda} \tag{6}$$

is the impurity of a node and $\mathcal{I}_L^q, \mathcal{I}_R^q$ are the index sets associated with the q-th node's left and right child nodes after the split respectively.

C. Additive Secret Sharing

In Privet, we use n-out-of-n additive secret sharing over $\mathbb{Z}_{2^Q}$, where Q denotes the number of bits for value representation. In such secret sharing, a secret value $x \in \mathbb{Z}_{2^Q}$ is additively split into n secret shares $\langle x \rangle_1, \langle x \rangle_2, \ldots, \langle x \rangle_n \in \mathbb{Z}_{2^Q}$ such that $[\![x]\!] = \langle x \rangle_1 + \langle x \rangle_2 + \cdots + \langle x \rangle_n \bmod 2^Q$. The n shares are held by n parties respectively to be engaged in a secure computation. For simplicity, we denote such additive secret sharing of x by $[\![x]\!]$. Below we introduce the basic operations related to additive secret sharing in the n-party setting.

• Sharing: To additively share a private value x of party P_l, P_l needs to generate $n-1$ random numbers $\{x_{m \neq l}\}, m \in [1, n]$ over $\mathbb{Z}_{2^Q}$ and sends $x_{m \neq l}$ to $P_{m \neq l}$, respectively. Then P_l holds $\langle x \rangle_l = (x - \sum_{p=1, p \neq l}^n x_p) \bmod 2^Q$ and $P_{m \neq l}$ holds $\langle x \rangle_{m \neq l} = x_{m \neq l}$, respectively, as a share of x. For conciseness, the modulo operation will be henceforth omitted in the following protocols.

• Reconstruction: To reconstruct ($\text{Rec}(\cdot)$) a shared value $[\![x]\!]$ on P_l, $P_{m \neq l}$ sends its share $\langle x \rangle_{m \neq l}$ to P_l and P_l computes $\sum_{p=1}^n \langle x \rangle_p$.

• Addition: For the two secret-shared values $[\![x]\!]$ and $[\![y]\!]$, to securely compute addition ($[\![z]\!] = [\![x]\!] + [\![y]\!]$), each participant P_m locally computes $\langle z \rangle_m = \langle x \rangle_m + \langle y \rangle_m$. Similarly, to compute subtraction ($[\![z]\!] = [\![x]\!] - [\![y]\!]$), each participant subtracts its local share of y from that of x.

• Multiplication: To multiply a secret-shared value $[\![x]\!]$ with a constant c ($[\![z]\!] = c \times [\![x]\!]$), each participant multiplies its local share of x by c. To multiply two secret-shared values $[\![x]\!], [\![y]\!]$ (denoted by $[\![z]\!] = [\![x]\!] \times [\![y]\!]$ where $z = xy$), the multiplication triple technique can be used [39]. In an offline phase, all parties obtain a secret-shared multiplication triple ($[\![a]\!], [\![b]\!], [\![c]\!]$), where a, b are uniformly random numbers in $\mathbb{Z}_{2^Q}$ and $c = ab$. The secret-shared triples are data-independent and can be prepared and distributed offline by an independent third-party [40], so hereafter we assume the triples are available for use in online secure computation among the parties. The secret-shared multiplication proceeds as follows. Each party P_m locally computes $\langle e \rangle_m = \langle x \rangle_m - \langle a \rangle_m$ and $\langle f \rangle_m = \langle y \rangle_m - \langle b \rangle_m$. After that, the parties run $\text{Rec}([\![e]\!]), \text{Rec}([\![f]\!])$. Next, P_m computes $\langle z \rangle_m = j \times e \times f + f \times \langle a \rangle_m + e \times \langle b \rangle_m + \langle c \rangle_m$, where $j = 1$ if $m = 1$ and $j = 0$ if $m \neq 1$. Table I summarizes the key notations in this article.

TABLE I
SUMMARY OF NOTATIONS

Notation	Description
P_m	Participant m
n	Number of participants
N	Number of samples owned by each participant
J	Number of total features
J_m	Number of features owned by participant m
$\mathcal{D}_m^{N\times J_m}$	Vertically partitioned dataset owned by participant m
$\mathbf{y}$	Label set
D	Dimension of decision table
$\mathcal{T}$	Decision table model
T	Number of decision tables to be trained
$[\![\mathbf{x}]\!]$	Secret-shared vector
$\langle \mathbf{x} \rangle_m$	One share of a vector held by participant m

IV. SYSTEM OVERVIEW

A. System Architecture

Fig. 3 illustrates the system architecture of Privet, which targets the vertical federated learning scenario. In Privet, multiple participants (e.g., business organizations and institutions) want to collaboratively train gradient boosted decision tables over vertically partitioned data. Under such setting, a dataset consisting of N samples (each is associated with a feature vector and a label) is vertically partitioned among n participants $P_1, P_2, \ldots, P_n$. Each participant P_m holds its respective dataset $\mathcal{D}_m^{N\times J_m} = \{\mathbf{x}_i^{J_m}\}_{i=0}^{N-1}$, where J_m denotes the number of features owned by P_m and is subject to $\sum_{m=1}^{n} J_m = J$, $\mathbf{x}_i^{J_m}$ represents the i-th sample of $\mathcal{D}_m^{N\times J_m}$. Let $\mathbf{y} = \{y_i\}_{i=0}^{N-1}$ be the set of sample labels. Following prior works on VFL [7], [8], [10], we consider two roles for the participants: active participant (AP) and passive participant (PP). In particular, there is one AP that holds a local dataset as well as the label set $\mathbf{y}$; and the remaining participants are PPs, each only holding a local dataset. For simplicity, in Privet, we assume the participant P_n is the AP.

Throughout the secure training process in Privet, each participant keeps its feature data locally. The Boolean tests and output values of the decision tables are securely learned in Privet, in such a manner that no participant knows the complete models. In particular, Privet follows a setting similar to the works [7], [8], [10], [11] under VFL, where each participant learns partial information of the learned models. Specifically, in Privet, all the participants know the split feature of each test in a decision table of the ensemble and who owns this feature, but only the participant owning this feature knows the split threshold of the test. Formally, for the learned test $F_d < t_d$ at the d-th level ($d \in [0, D-1]$) of the t-th decision table ($t \in [1, T]$) in the ensemble, the split feature F_d is revealed to all participants but the threshold t_d is only known by the participant owning the feature data corresponding to F_d. In addition, all the output values of leaf nodes are produced in secret-shared form among all participants.

B. Threat Model

Privet is designed under the semi-honest adversary model, as is common in state-of-the-art security designs on vertical federated learning [8], [41]. Specifically, in Privet, each participant is assumed to faithfully follow the protocol specification but may try to deduce other participants' private information from the messages they receive. It is noted that though we consider two roles AP and PP for the participants, no additional trust is assumed regarding the AP. The semi-honest adversary model should be reasonable in practice because VFL aims at breaking down the data silos between business organizations, where the behavior of each organization is strictly enforced by privacy regulations [42]. We also consider that a static adversary may corrupt a subset of τ participants ($\tau \leq n-1$). That is, a static adversary may choose a subset of the participants to corrupt before the VFL procedure and the chosen participants remain corrupted during the VFL procedure.

Under the above threat model, Privet aims to guarantee that a semi-honest participant individually cannot learn any other participant's local data and learned partial model (tests and output values of each decision table in the ensemble) throughout the VFL procedure. In case of collusion among a subset of the participants, Privet strives to still ensure that the honest participants' private information is protected against the corrupted participants. Like prior works [8], [10], [32], Privet does not hide the data-independent generic parameters, such as the dimension D and the number of decision tables T.

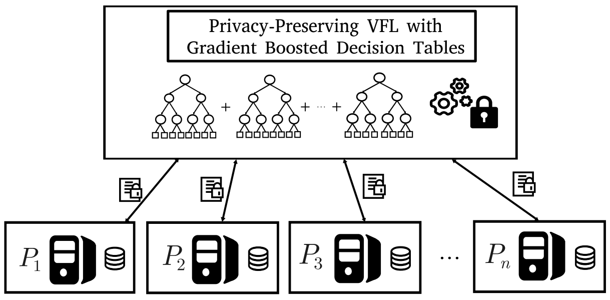

Privacy-Preserving VFL with Gradient Boosted Decision Tables
+
+ ··· +
P_1
P_2
P_3
···
P_n

Fig. 3. Privet's system architecture.

V. THE DESIGN OF PRIVET

A. Overview

We provide in Algorithm 2 an overview of the secure training framework in Privet, which inputs the vertically partitioned datasets $\{\mathcal{D}_m^{N\times J_m}\}_{m=1}^{n}$ and the label set $\mathbf{y}$ from the participants, and outputs an ensemble $\mathcal{E}$ of distributed decision tables among the participants. At the beginning, the secret-shared inference result $[\![\hat{\mathbf{y}}^{(0)}]\!]$ is initialized as the secret sharing $[\![\mathbf{0}_N]\!]$ (N denotes the length of the secret-shared vector), and the AP distributes the secret shares of its label set to other participants. After that, T distributed decision tables are securely built sequentially in T rounds (lines 4–8).

We develop a secure decision table learning algorithm SecTable to support the secure training of a single (distributed) decision table in each round. SecTable consists of several secure components, including (i) secure node splitting SecSplit, (ii) secure Sigmoid evaluation SecSigmoid, and (iii) secure discretization SecDisc. The secure node splitting component SecSplit (Section V-B1) is to securely split the nodes at a certain

Algorithm 2: Overview of Our Secure Training Framework.

```
Input: $P_1, P_2, \ldots, P_n$ hold local datasets $\{\mathcal{D}_m^{N\times J_m}\}_{m=1}^n$
  and one AP holds the label set $\mathbf{y}$.
Output: $P_1, P_2, \ldots, P_n$ obtain an ensemble $\mathcal{E}$ of $T$
  distributed decision tables $\{\mathcal{T}_t\}_{t=1}^T$, each containing $D$
  tests and $2^D$ secret-shared output values.
1:  Initialize ensemble $\mathcal{E} = \{\}$.
2:  $[\![\hat{\mathbf{y}}^{(0)}]\!] \leftarrow [\![\mathbf{0}_N]\!]$ .
3:  AP secret-shares $\mathbf{y}$ to other participants to produce
    $[\![\mathbf{y}]\!]$.
4:  for $t \in [1, T]$ do
5:    $\mathcal{T}_t \leftarrow$SecTable($\{\{\mathcal{D}_m^{N\times J_m}\}_{m=1}^n, [\![\mathbf{y}]\!], [\![\hat{\mathbf{y}}^{(t-1)}]\!]$).
6:    $[\![\hat{\mathbf{y}}^{(t)}]\!] \leftarrow$SecInfer($\mathcal{T}_t$, $\{\{\mathcal{D}_m^{N\times J_m}\}_{m=1}^n$).
7:    $[\![\hat{\mathbf{y}}^{(t)}]\!] \leftarrow [\![\hat{\mathbf{y}}^{(t-1)}]\!] + [\![\hat{\mathbf{y}}^{(t)}]\!]$.
8:  end for
9:  $\mathcal{E}.append(\mathcal{T}_t)$.
10: Output the ensemble $\mathcal{E}$ held by $P_1, P_2, \ldots, P_n$.
```

level and partition the index sets associated with these nodes without revealing the partitioned index sets. The secure Sigmoid evaluation component SecSigmoid (Section V-B2) inputs a secret-shared value and calculates the Sigmoid function in the secret sharing domain. The secure discretization component SecDisc (Section V-B3) is to securely rearrange the secret-shared gradients according to the local permutations owned by each participant and then group them into buckets. Through the synergy of these components, SecTable allows the participants to securely train a distributed decision table in each round, for which we will give the details in Section V-B4.

After securely training a distributed decision table in a certain round, secure inference needs to be conducted, of which the result will be added to previous inference results (line 7) for use in SecTable in the next round. To this end, we develop a secure distributed decision table inference protocol SecInfer (Section V-C), which inputs each participant's local data and partial model to produce secret-shared inference results without leaking their data and partial model. It is worth noting that SecInfer can also be used to support secure inference for new data after the completion of the whole training process.

B. Secure Distributed Decision Table Training

1) Secure Node Splitting: An oblivious tree grows to a new level by splitting each node at the current level into two child nodes. In plaintext centralized decision table training (Algorithm 1), node splitting is performed by partitioning the samples associated with the node to be split. However, in VFL, the partitioning of samples must be revealed to all participants because the training dataset is vertically partitioned and all participants hold the same samples. For instance, given a test "$Height < 180$", the participant owning feature data of "$Height$" needs to tell other participants which samples are less than 180 and which samples are greater than 180, which will leak each sample's range of "$Height$" and raise critical privacy concerns.

Algorithm 3: Secure Node Splitting (SecSplit).

```
Input: $P_1, P_2, \ldots, P_n$ hold local datasets $\{\mathcal{D}_m^{N\times J_m}\}_{m=1}^n$,
  the secret-shared first and second-order gradient vectors
  $\{[\![\mathbf{g}^{k,d}]\!]\}_{k=0}^{2^d-1}$, $\{[\![\mathbf{h}^{k,d}]\!]\}_{k=0}^{2^d-1}$ associated with $2^d$ nodes at
  the $d$-th level; $P_l$ holds the optimal test $F_d < t_d$ at the $d$-th
  level.
Output: $P_1, P_2, \ldots, P_n$ obtain the secret-shared gradient
  vectors $\{[\![\mathbf{g}^{k,d+1}]\!]\}_{k=0}^{2^{d+1}-1}$ and $\{[\![\mathbf{h}^{k,d+1}]\!]\}_{k=0}^{2^{d+1}-1}$
  associated with the $2^{d+1}$ split nodes.
// $P_l$ performs:
1:  $\mathbf{v}_l \leftarrow \mathbf{0}_N$.
2:  for $i \in [0, N-1]$ do
3:    if $\mathbf{x}_i^{J_l}(F_d) < t_d$ then
4:      $\mathbf{v}_{li} \leftarrow 1$ // Set the $i$-th element of $\mathbf{v}_l$ as 1.
5:    end if
6:  end for
7:  $\mathbf{v}_r \leftarrow \mathbf{1}_N - \mathbf{v}_l$.
8:  $P_l$ secret-shares $\mathbf{v}_l$ and $\mathbf{v}_r$ to other participants.
// $P_1, P_2, \ldots, P_n$ perform:
9:  for $k \in [0, 2^d - 1]$ do
10:   $[\![\mathbf{g}^{2k,d+1}]\!] \leftarrow [\![\mathbf{v}_l]\!] \times [\![\mathbf{g}^{k,d}]\!]$.
11:   $[\![\mathbf{h}^{2k,d+1}]\!] \leftarrow [\![\mathbf{v}_l]\!] \times [\![\mathbf{h}^{k,d}]\!]$.
12:   $[\![\mathbf{g}^{2k+1,d+1}]\!] \leftarrow [\![\mathbf{v}_r]\!] \times [\![\mathbf{g}^{k,d}]\!]$.
13:   $[\![\mathbf{h}^{2k+1,d+1}]\!] \leftarrow [\![\mathbf{v}_r]\!] \times [\![\mathbf{h}^{k,d}]\!]$.
14: end for
15: Output the secret-shared gradient vectors
    $\{[\![\mathbf{g}^{k,d+1}]\!]\}_{k=0}^{2^{d+1}-1}$ and $\{[\![\mathbf{h}^{k,d+1}]\!]\}_{k=0}^{2^{d+1}-1}$ associated
    with nodes at the $(d+1)$-th level.
```

To avoid this leakage, we design a secure node splitting component SecSplit. Inspired by existing works [7], [8], [41], we utilize indicator vectors to conduct secure node splitting for each level of the decision table. At a high level, Privet associates the k-th node ($k \in [0, 2^d - 1]$) at the d-th level ($d \in [0, D-1]$) of the oblivious decision tree with a first-order gradient vector $[\![\mathbf{g}^{k,d}]\!]$ and a second-order gradient vector $[\![\mathbf{h}^{k,d}]\!]$, each containing N elements that are secret-shared among all participants $P_1, P_2, \ldots, P_n$. If the i-th sample is partitioned into this node, the i-th element in $[\![\mathbf{g}^{k,d}]\!]$ and $[\![\mathbf{h}^{k,d}]\!]$ will be set as the i-th sample's first and second-order gradients, respectively, otherwise the i-th element will be set as $[\![0]\!]$.

Algorithm 3 gives the procedure of secure node splitting. First, participant P_l who owns the optimal test $F_d < t_d$ at the d-th level locally generates left indicator vector $\mathbf{v}_l$ and right indicator vector $\mathbf{v}_r$ and then distributes their secret sharings (denoted by $[\![\mathbf{v}_l]\!]$ and $[\![\mathbf{v}_r]\!]$) to other participants (i.e., lines 1-8 in Algorithm 3). Upon receiving $[\![\mathbf{v}_l]\!]$ and $[\![\mathbf{v}_r]\!]$, for the k-th node at this level, $P_1, P_2, \ldots, P_n$ update the first-order and second-order gradient vector of the k-th node's left and right child nodes. The update is achieved with secure element-wise multiplication between secret-shared indicator vectors and gradient vectors (i.e., lines 9–14 in Algorithm 3). In this way, the index set processed by each node is hidden and the number of the samples processed by each node remains constant as N, which means an adversary cannot deduce any information from node splitting.

2) Secure Sigmoid Evaluation: There are mainly two challenges in securely calculating the Sigmoid function in the secret sharing domain. First, how to compute the division $[\![\frac{x}{y}]\!]$ given two secret-shared values $[\![x]\!]$ and $[\![y]\!]$? Second, how to compute the exponentiation function $[\![e^x]\!]$ given a secret-shared value $[\![x]\!]$? Next, we introduce how Privet tackles the two challenges so as to allow the participants to securely calculate the Sigmoid function in the secret sharing domain. For the first challenge, we introduce a secure division component SDiv by transforming the division calculation into a numerical optimization problem. Specifically, we note that the core obstacle of calculating $[\![\frac{x}{y}]\!]$ given $[\![x]\!]$ and $[\![y]\!]$ is to calculate the secret-shared reciprocal $[\![\frac{1}{y}]\!]$. Therefore, we first approximate $\frac{1}{y}$ by the iterative Newton-Raphson algorithm [43], following previous works [44], [45]: $z_i \leftarrow 2z_{i-1} - yz_{i-1}^2$, which will converge to $z_n \approx \frac{1}{y}$. In Privet, we fix the initialization $z_0 = 1/Y$, where Y is a sufficiently large value. Note that the approximation consists of basic subtraction and multiplication operations which are naturally supported in the secret sharing domain, given $[\![y]\!]$, the secret-shared reciprocal $[\![\frac{1}{y}]\!]$ can be securely calculated. After securely calculating the reciprocal $[\![\frac{1}{y}]\!]$, Privet multiplies $[\![x]\!]$ by $[\![\frac{1}{y}]\!]$ to obtain $[\![\frac{x}{y}]\!]$, i.e., $[\![\frac{x}{y}]\!] = [\![x]\!] \cdot [\![\frac{1}{y}]\!]$.

For the second challenge, i.e., computing the exponentiation function $[\![e^x]\!]$ given a secret-shared value $[\![x]\!]$, we approximate e^x by limit characterization, inspired by [44]: $e^x \approx (1 + \frac{x}{2^n})^{2^n}$, which provides a good approximation of e^x. Note that since the approximation consists of basic addition and multiplication operations which are naturally supported in the secret sharing domain, given $[\![x]\!]$, the secret-shared exponentiation $[\![e^x]\!]$ can be securely calculated. However, we note that the approximation method requires 2^n chain multiplications, and thus requires 2^n rounds of online communication. The approximation method is inefficient in practice since the communication complexity grows exponentially. Therefore, Privet further reduces the exponential communication complexity to linear communication complexity. More specifically, we note that the computation in the approximation can be regarded as a^{2^n} where $a = 1 + \frac{x}{2^n}$. Therefore, given $[\![a]\!]$, Privet first securely calculates $[\![a^2]\!]$, which only requires one round of communication. After that, Privet regards the output as $[\![y]\!] = [\![a^2]\!]$ followed by securely calculating $[\![y^2]\!]$, which also only requires one round of communication. Therefore, in this way, we can securely calculate $[\![a^{2^n}]\!]$ in $\log 2^n = n$ rounds instead of 2^n rounds. Clearly, there is a trade-off between accuracy and efficiency in approximating e^x with $(1 + \frac{x}{2^n})^{2^n}$ for computation in the secret sharing domain. In principle increasing the value of n would lead to a more accurate approximation of the Sigmoid function. However, this also leads to increased computation and communication costs. Yet, as will be shown by our experiments, a small value of n (in our case, we set $n = 2$) suffices to enable Privet to achieve the accuracy comparable to plaintext centralized learning.

Remark: In the literature, there exist some methods for approximating the Sigmoid function so as to support secure Sigmoid evaluation, including Taylor expansion [46], piece-wise approximation [27], and function approximation like $f(x) = \frac{0.5x}{1+|x|} + 0.5$ [8]. For the Taylor expansion method, it requires a small input parameter (very close to 0), which is hard to satisfy in machine learning. The piece-wise approximation method has no such requirement but suffers from notable accuracy loss [8]. The work that is most closely related to ours is due to Fang et al. [8], who apply another function approximation method to approximate the Sigmoid function, i.e., $f(x) = \frac{0.5x}{1+|x|} + 0.5$. However, their method still experienced non-trivial loss in accuracy in their securely trained XGBoost model. As reported in their experiments, the Area Under the ROC Curve (AUC) value would go up to *0.84463* from 0.82945 if they replace the secure Sigmoid approximation with *plaintext* Sigmoid computation. In contrast, as will be shown by the experiments in Section VII-B, our proposed SecSigmoid can achieve AUC values that are highly close to those obtained using plaintext centralized learning (e.g., the gap can be as small as 0.0005).

3) Secure Discretization: Discretization, also called bucketing, is a commonly used grouping method in large-scale machine learning [22], [35]. Specifically, discretization groups the samples into a small number of buckets so as to allow the model training to scale on larger datasets. Let B denote the number of buckets in discretization, where $B \ll N$ and N is the number of samples. In gradient boosting, gradients are grouped into buckets and the sum of gradients in each bucket is calculated in the training stage [22], [35]. Typically, for each feature, the gradients are first permuted by a permutation π, which is obtained by sorting the values of this feature. Then the permuted gradients are partitioned into B buckets. Obviously, the cost of training on B buckets instead of N samples can be greatly reduced.

However, discretization is non-trivial in privacy-preserving machine learning. In existing MPC-based works [32], [33], [34], the training data is secret-shared among a fixed set of computing servers, and sorting the secret-shared training data for discretization requires a large number of secure comparison operations, which is expensive in the secret sharing domain. In Privet, the training data is vertically partitioned, and thus the sorting process can be achieved locally to reduce the overhead. However, it is still difficult to permute the secret-shared gradient vector by a permutation π held by a participant P_l without revealing π to other participants.

To tackle the challenge, we propose a secure permutation algorithm SecPerm (shown in Algorithm 4), which stems from the correlated randomness (CR) scheme in [8]. Our tailored design SecPerm enables our secure discretization component SecDisc to outperform that in [8] in supporting an arbitrary number of participants. Fang et al. [8] design two MPC-based secure discretization methods. Specifically, they first propose a basic discretization method based on multiplications between secret-shared lagre-scale matrices. Then they obtain significant speedup over the basic method by utilizing CR to efficiently permute secret-shared gradients, and then group them. However, both the basic and improved methods in [8] only work under the two-party setting in VFL. The work [7] is the first MPC-based work supporting more than two participants in VFL with GBDT, but it simply follows the basic discretization method in [8]. In contrast, Privet tailors the improved discretization method from [8] to support an arbitrary number of participants, which is

Algorithm 4: Secure Permutation (SecPerm).

```
Input: $P_1, P_2, \ldots, P_n$ hold the secret-shared vector $[\![\mathbf{x}]\!]$;
  $P_l$ inputs the permutation $\pi$.
Output: $P_1, P_2, \ldots, P_n$ obtain the permuted secret-shared
  vector $[\![\mathbf{u}]\!]$ subject to $\mathbf{u} = \pi(\mathbf{x})$.
// Initialization:
1: $P_1, P_2, \ldots, P_n$ hold in advance the secret shares of
   $\pi_p(\mathbf{r})$ and $\mathbf{r}$, and $P_l$ additionally holds $\pi_p$.
// Online computation:
Round 1:
2: $P_l$ generates $\pi_s$ subject to $\pi(\cdot) = \pi_s[\pi_p(\cdot)]$, and then
   sends $\pi_s$ to other participants.
3: Each participant $P_m$ locally calculates $\langle \mathbf{x} \rangle_m - \langle \mathbf{r} \rangle_m$.
Round 2:
4: $P_1, P_2, \ldots, P_n$ run Rec($[\![\mathbf{x} - \mathbf{r}]\!]$) to reveal $\mathbf{x} - \mathbf{r}$ to $P_l$.
5: $P_l$: $\langle \mathbf{u} \rangle_l \leftarrow \pi(\mathbf{x} - \mathbf{r}) + \pi_s(\langle \pi_p(\mathbf{r}) \rangle_l)$.
6: $P_{m \neq l}$: $\langle \mathbf{u} \rangle_{m \neq l} \leftarrow \pi_s(\langle \pi_p(\mathbf{r}) \rangle_{m \neq l})$.
7: Output the secret-shared vector $[\![\mathbf{u}]\!]$ held by
   $P_1, P_2, \ldots, P_n$.
```

Algorithm 5: Secure Discretization (SecDisc).

```
Input: $P_1, P_2, \ldots, P_n$ hold the secret-shared first and
  second-order gradient vectors $[\![\mathbf{g}]\!]$ and $[\![\mathbf{h}]\!]$; $P_l$ holds the
  permutation $\pi$.
Output: $P_1, P_2, \ldots, P_n$ obtain two secret-shared vectors
  $[\![\boldsymbol{\alpha}]\!]$ and $[\![\boldsymbol{\beta}]\!]$ containing $B$ grouped first and second-order
  gradients, respectively.
1: $[\![\mathbf{g}']\!] \leftarrow$ SecPerm($\pi$, $[\![\mathbf{g}]\!]$).
2: $[\![\mathbf{h}']\!] \leftarrow$ SecPerm($\pi$, $[\![\mathbf{h}]\!]$).
3: $[\![\boldsymbol{\alpha}]\!] \leftarrow [\![\mathbf{0}_B]\!]$, $[\![\boldsymbol{\beta}]\!] \leftarrow [\![\mathbf{0}_B]\!]$.
4: $M \leftarrow N/B$
5: for $b \in [0, B-1]$ do
6:   $[\![\boldsymbol{\alpha}_b]\!] \leftarrow \sum_{i=b\times M}^{(b+1)\times M-1} [\![\mathbf{g}'_i]\!]$.
7:   $[\![\boldsymbol{\beta}_b]\!] \leftarrow \sum_{i=b\times M}^{(b+1)\times M-1} [\![\mathbf{h}'_i]\!]$.
8: end for
9: Output the secret-shared vectors $[\![\boldsymbol{\alpha}]\!]$ and $[\![\boldsymbol{\beta}]\!]$ held by
   $P_1, P_2, \ldots, P_n$.
```

more efficient than the straightforward secret-shared lagre-scale matrix multiplication-based method from [7].

As shown in Algorithm 4, at the beginning of SecPerm, participants $P_1, P_2, \ldots, P_n$ hold a secret-shared vector $[\![\mathbf{x}]\!]$ and P_l holds a permutation π. At the end of SecPerm, participants $P_1, P_2, \ldots, P_n$ hold a secret-shared vector $[\![\mathbf{u}]\!]$ where $\mathbf{u} = \pi(\mathbf{x})$. SecPerm guarantees that except for P_l, other participants cannot know the permutation π. In the initialization of Algorithm 4, all participants hold in advance the secret shares of $\pi_p(\mathbf{r})$ and $\mathbf{r}$, and P_l additionally holds π_p. After the initialization, all participants collaboratively permute $[\![\mathbf{x}]\!]$ in 2 rounds. In the first round, P_l generates the permutation π_s, which subjects to $\pi(\cdot) = \pi_s[\pi_p(\cdot)]$, and then P_l sends π_s to all other participants. After that, each participant P_m locally calculates $\langle \mathbf{x} \rangle_m - \langle \mathbf{r} \rangle_m$. In the second round, $\mathbf{x} - \mathbf{r}$ is revealed to P_l. Finally, the participants output $[\![\mathbf{u}]\!] = [\![\pi(\mathbf{x})]\!]$ (i.e., lines 5–6 in Algorithm 4).

The correctness analysis of SecPerm is as follows:

$$\begin{aligned}\mathbf{u} &= \langle \mathbf{u} \rangle_1 + \langle \mathbf{u} \rangle_2 + \cdots + \langle \mathbf{u} \rangle_l + \cdots + \langle \mathbf{u} \rangle_n \\ &= \pi_s(\langle \pi_p(\mathbf{r}) \rangle_1) + \pi_s(\langle \pi_p(\mathbf{r}) \rangle_2) + \cdots \\ &\quad + \pi(\mathbf{x} - \mathbf{r}) + \pi_s(\langle \pi_p(\mathbf{r}) \rangle_l) + \cdots + \pi_s(\langle \pi_p(\mathbf{r}) \rangle_n) \\ &= \pi(\mathbf{x} - \mathbf{r}) + \pi_s(\pi_p(\mathbf{r})) \\ &= \pi(\mathbf{x} - \mathbf{r}) + \pi(\mathbf{r}) \\ &= \pi(\mathbf{x}).\end{aligned}$$

Then, we introduce how Privet securely realizes discretization protocol SecDisc based on SecPerm. At a high level, SecDisc first uses SecPerm to securely permute the secret-shared first and second-order gradients of $P_1, P_2, \ldots, P_n$ with a permutation π held by P_l, and then partitions the gradients into B buckets. Algorithm 5 describes the details of our secure discretization protocol.

At the beginning, the secret-shared first and second-order gradient vectors $[\![\mathbf{g}]\!]$ and $[\![\mathbf{h}]\!]$ are securely permuted by SecPerm, which outputs $[\![\mathbf{g}']\!]$ and $[\![\mathbf{h}']\!]$. After that, $P_1, P_2, \ldots, P_n$ first initialize two secret-shared vectors $[\![\boldsymbol{\alpha}]\!]$ and $[\![\boldsymbol{\beta}]\!]$ of length B to store the grouped first and second-order gradients, respectively. $[\![\boldsymbol{\alpha}]\!]$ and $[\![\boldsymbol{\beta}]\!]$ can be locally initialized as $\mathbf{0}_B$. After that, for $b \in [0, B-1]$, the b-th bucket's secret-shared grouped first and second-order gradients $[\![\boldsymbol{\alpha}_b]\!]$ and $[\![\boldsymbol{\beta}_b]\!]$ are calculated as follows:

$$[\![\boldsymbol{\alpha}_b]\!] = \sum_{i=b\times M}^{(b+1)\times M-1} [\![\mathbf{g}'_i]\!]; \quad [\![\boldsymbol{\beta}_b]\!] = \sum_{i=b\times M}^{(b+1)\times M-1} [\![\mathbf{h}'_i]\!],$$

where $M = N/B$ is the number of gradients in a bucket. For conciseness, we assume that N can divide B evenly. As introduced in Section V-B1, since the invalid gradients are set as $[\![0]\!]$, the sum of the secret-shared gradients in a bucket is equal to that in the plaintext.

4) Secure Decision Table Training Algorithm: In this section, we introduce how Privet combines the components introduced above to securely train a distributed decision table. Algorithm 6 (named as SecTable) describes this process. SecTable is the secure instantiation of Algorithm 1 and relies on the coordination of the secure components introduced above.

Algorithm 6 inputs the vertically partitioned datasets $\{\mathcal{D}_m^{N\times J_m}\}_{m=1}^n$, secret-shared label $[\![\mathbf{y}]\!]$, and aggregated inference results $[\![\hat{\mathbf{y}}]\!]$ from the previous round of training, and then outputs a distributed decision table. The Boolean tests at different levels of the decision table are held by different participants and the output values associated with each leaf node are stored in an secret-shared vector $[\![\mathbf{w}]\!]$.

At the beginning of Algorithm 6, $P_1, P_2, \ldots, P_n$ calculate the secret-shared first and second-order gradient vectors (for the root node) (i.e., $[\![\mathbf{g}]\!]$ and $[\![\mathbf{h}]\!]$ at lines 1–6). After calculating the secret-shared gradients, Privet initializes a secret-shared vector $[\![\mathbf{w}]\!] = [\![\mathbf{0}_{2^D}]\!]$ to store the 2^D secret-shared output values. After that, a decision table will be built level by level. Similar to the functionality of find_split in Algorithm 1, Privet securely selects the optimal test $F_d < t_d$ at the d-th level ($d \in [0, D-1]$).

Algorithm 6: Secure Training of a Distributed Decision Table (SecTable).

```
Input: $P_1, P_2, \ldots, P_n$ hold local datasets $\{\mathcal{D}_m^{N\times J_m}\}_{m=1}^n$,
  secret-shared label set $[\![\mathbf{y}]\!]$ and aggregated inference results of
  the current models $[\![\hat{\mathbf{y}}]\!]$.
Output: $P_1, P_2, \ldots, P_n$ obtain a distributed decision table $\mathcal{T}$
  with $D$ tests and a secret-shared vector $[\![\mathbf{w}]\!]$ of $2^D$ output values.
// Initialization:
1:  if the problem is regression then
2:      $[\![\mathbf{g}]\!] \leftarrow [\![\hat{\mathbf{y}}]\!] - [\![\mathbf{y}]\!], [\![\mathbf{h}]\!] \leftarrow [\![\mathbf{1}]\!]$.
3:  else
4:      $[\![\mathbf{p}]\!] \leftarrow \mathsf{SecSigmoid}([\![\hat{\mathbf{y}}]\!])$.
5:      $[\![\mathbf{g}]\!] \leftarrow [\![\mathbf{p}]\!] - [\![\mathbf{y}]\!], [\![\mathbf{h}]\!] \leftarrow [\![\mathbf{p}]\!] \times ([\![\mathbf{1}]\!] - [\![\mathbf{p}]\!])$.
6:  end if
7:  $[\![\mathbf{w}]\!] \leftarrow [\![\mathbf{0}_{2^D}]\!]$.
// Training:
8:  for $d \in [0, D-1]$ do
9:      $[\![\boldsymbol{\sigma}]\!] \leftarrow [\![\mathbf{0}_J]\!], \boldsymbol{\gamma} \leftarrow \mathbf{0}_J$.
10:     for $j \in [0, J-1]$ in parallel do
11:         Participant who holds $\mathcal{D}(j)$ sorts $\mathcal{D}(j)$ to produce the
            permutation $\pi_j$.
12:         $[\![\boldsymbol{\delta}]\!] \leftarrow [\![\mathbf{0}_{B-1}]\!]$.
13:         for $k \in [0, 2^d - 1]$ do
14:             $[\![\boldsymbol{\alpha}^{k,d}]\!], [\![\boldsymbol{\beta}^{k,d}]\!] \leftarrow \mathsf{SecDisc}(\pi_j, [\![\mathbf{g}^{k,d}]\!], [\![\mathbf{h}^{k,d}]\!])$.
15:             $[\![G]\!], [\![H]\!] \leftarrow \sum_{i=0}^{B-1} [\![\boldsymbol{\alpha}_i^{k,d}]\!], \sum_{i=0}^{B-1} [\![\boldsymbol{\beta}_i^{k,d}]\!]$.
16:             for $c \in [0, B-2]$ do
17:                 $[\![G_l]\!], [\![H_l]\!] \leftarrow \sum_{i=0}^{c} [\![\boldsymbol{\alpha}_i^{k,d}]\!], \sum_{i=0}^{c} [\![\boldsymbol{\beta}_i^{k,d}]\!]$.
18:                 $[\![G_r]\!], [\![H_r]\!] \leftarrow [\![G]\!] - [\![G_l]\!], [\![H]\!] - [\![H_l]\!]$.
19:                 $[\![\boldsymbol{\delta}_c]\!] \leftarrow [\![\boldsymbol{\delta}_c]\!] - \frac{1}{2}\frac{[\![G_l]\!]^2}{[\![H_l]\!]+[\![\lambda]\!]} - \frac{1}{2}\frac{[\![G_r]\!]^2}{[\![H_r]\!]+[\![\lambda]\!]}$.
20:             end for
21:         end for
22:         $q \leftarrow \mathsf{SecArgmin}([\![\boldsymbol{\delta}]\!])$.
23:         $[\![\boldsymbol{\sigma}_j]\!] \leftarrow [\![\boldsymbol{\delta}_q]\!]$.
24:         $\boldsymbol{\gamma}_j \leftarrow q$.
25:     end for
26:     $F_d \leftarrow \mathsf{SecArgmin}([\![\boldsymbol{\sigma}]\!])$ // Optimal split feature.
27:     $q_d \leftarrow \boldsymbol{\gamma}_{F_d}$ // Optimal bucket ID.
28:     $t_d \leftarrow \{\pi_{F_d}[\mathcal{D}(F_d)]\}_{(q_d+1)\times(N/B)}$.
29:     $\mathsf{SecSplit}(\{\mathcal{D}_m^{N\times J_m}\}_{m=1}^n, \{[\![\mathbf{g}^{k,d}]\!]\}_{k=0}^{2^d-1}, \{[\![\mathbf{h}^{k,d}]\!]\}_{k=0}^{2^d-1}, F_d <$
        $t_d)$.
30: end for
31: for $k \in [0, 2^D - 1]$ do
32:     $[\![G]\!], [\![H]\!] \leftarrow \sum_{i=0}^{N-1} [\![\mathbf{g}_i^{k,D}]\!], \sum_{i=0}^{N-1} [\![\mathbf{h}_i^{k,D}]\!]$.
33:     $[\![\mathbf{w}_k]\!] \leftarrow -\frac{[\![G]\!]}{[\![H]\!]+[\![\lambda]\!]}$.
34: end for
35: Output a decision table $\mathcal{T}$ with $D$ tests, each held by a
    participant, and a secret-shared vector $[\![\mathbf{w}]\!]$ held by
    $P_1, P_2, \ldots, P_n$.
```

Specifically, the selection is made greedily: the learning algorithm first selects the best test for each feature (i.e., lines 10–25) and then selects the optimal test among the J selected candidate tests (i.e., lines 26–28). It is noted that the selection here is made following that in the plaintext domain, which is introduced in Section III-B, and the operations in the selection are substituted with secure operations and proposed components. For the J candidate tests, we initialize a secret-shared vector $[\![\boldsymbol{\sigma}]\!] = [\![\mathbf{0}_J]\!]$ to store the score of each feature's best test. Besides, Privet uses a public vector $\boldsymbol{\gamma} = \mathbf{0}_J$ to record the bucket ID of each feature's best test.

For simplicity, the total J features are numbered from 0 to $J-1$. At the beginning of the loop for the j-th feature in Algorithm 6, the participant who holds the j-th feature first generates a permutation π_j locally by sorting the values of the j-th feature (denoted by $\mathcal{D}(j)$) in the ascending order, which will be used to securely permute the secret-shared gradient vectors associated with each node at this level. After that, a naive method is to permute the gradient vectors and then adapt the Exact Greedy Algorithm [35] to enumerate each training sample to find the best test. However, enumerating all training samples incurs heavy computation overhead. Moreover, it will incur prohibitively expensive communication overhead in the distributed setting, degrading the efficiency of the system.

We propose a component SecDisc (shown in Algorithm 5) to tackle this challenge and enable Privet to scale on larger datasets. Specifically, SecDisc inputs secret-shared gradient vectors $[\![\mathbf{g}^{k,d}]\!]$ and $[\![\mathbf{h}^{k,d}]\!]$ associated with the k-th node at the d-th level. The secret-shared gradients in $[\![\mathbf{g}^{k,d}]\!]$ and $[\![\mathbf{h}^{k,d}]\!]$ are securely discretized into B buckets and stored in secret-shared vectors $[\![\boldsymbol{\alpha}^{k,d}]\!]$ and $[\![\boldsymbol{\beta}^{k,d}]\!]$, respectively. There are $B-1$ intervals among the B buckets and each corresponds to a candidate test. In this way, Privet only needs to select the best test from $B-1$ candidate tests for each feature, instead of enumerating N samples, so as to save considerable computation and communication cost. In Privet, we initialize a secret-shared vector $[\![\boldsymbol{\delta}]\!] = [\![\mathbf{0}_{B-1}]\!]$ for each feature to store the scores of the $B-1$ candidate tests.

After securely discretizing gradients into B buckets, the $B-1$ candidate tests are evaluated to select the best test of the j-th feature. For the c-th candidate test ($c \in [0, B-2]$), the first $c+1$ buckets are aggregated to get $[\![G_l]\!]$ and $[\![H_l]\!]$, which are the sum of gradients associated with the left child node, and the remaining $B-c-1$ buckets are aggregated to get $[\![G_r]\!]$ and $[\![H_r]\!]$, which are the sum of gradients associated with the right child node. The impurity of each node's two child nodes is securely computed following (6) and then aggregated together to produce the secret-shared score of the c-th candidate test following (5). The division needed in (6) can be securely calculated with our proposed secure component SDiv in Section V-B2. Then for the j-th feature, we will have $B-1$ secret-shared scores stored in $[\![\boldsymbol{\delta}]\!]$.

After getting the $B-1$ scores, we need to select the best test that achieves the minimum score, which requires a method to securely calculate the index of the minimum value in a secret-shared vector. To tackle this challenge, Privet introduces a component SecArgmin, which inputs a secret-shared vector and outputs the index of the minimum value of the vector. It is noted that the key operation in the function Argmin is comparison, which is not naturally supported in the secret sharing domain. The secure comparison operation in our Privet is introduced as follows. Given two secret-shared values $[\![A]\!]$ and $[\![B]\!]$, Privet first locally decomposes $[\![A-B]\!]$ into bits, and then inputs these bits into a parallel prefix adder (PPA) to securely compute the secret-shared most significant bit (MSB) of $[\![A-B]\!]$, inspired by [47], [48]. After that, we convert the secret-shared MSB into the arithmetic sharing domain by the method in [45], so as to get the secret-shared result of the secure comparison. Based on the secure comparison method introduced above, SecArgmin inputs the secret-shared vector $[\![\boldsymbol{\delta}]\!]$ and then outputs bucket ID $q \in [0, B-2]$ of the j-th feature's best test in the plaintext. In Privet, all participants can learn the produced bucket ID in the training stage, but only the participant who owns the j-th feature can get the threshold of the j-th feature's best test. Since the values of the j-th feature (denoted by $\mathcal{D}(j)$) is sorted in ascending order, the participant who owns the j-th feature can get the split threshold via looking up the sorted values (i.e., $\pi_j[\mathcal{D}(j)]$) with index $(q+1) \times (N/B)$. Other participants cannot deduce the split threshold because the j-th feature is kept locally by its owner and unavailable to them.

To select the optimal test of all features, Privet lets the participants record the j-th feature's best bucket ID q and secret-shared minimum split score $[\![\delta_q]\!]$ at the j-th position of $\boldsymbol{\gamma}$ and $[\![\boldsymbol{\sigma}]\!]$, respectively (i.e., lines 23–24 in Algorithm 6). Recall that for the J features, we use $\boldsymbol{\gamma}$ and $[\![\boldsymbol{\sigma}]\!]$ to store the bucket ID and split score of each feature's best test. The J indices of $\boldsymbol{\gamma}$ and $[\![\boldsymbol{\sigma}]\!]$ correspond to J features, respectively. After enumerating J features, J scores are stored in $[\![\boldsymbol{\sigma}]\!]$ and SecArgmin is needed to be called again on $[\![\boldsymbol{\sigma}]\!]$. The output is the split feature F_d of the optimal test, which is known by all participants. The optimal bucket ID q_d of the split feature can then be retrieved with F_d from $\boldsymbol{\gamma}$ (line 27). After that, the participant who owns the split feature F_d looks up the its sorted values $\pi_{F_d}[\mathcal{D}(F_d)]$ with index $(q_d+1) \times (N/B)$ to get the split threshold t_d.

After learning the d-th test $F_d < t_d$, the participant who owns $F_d < t_d$ cooperates with other participants to securely split all the nodes at the d-th level with SecSplit to create a new level (line 29). A decision table in Privet is learned level by level in this way. At the D-th level, Privet securely calculates output values for the 2^D leaf nodes following (3) (i.e., lines 31–34 in Algorithm 6), where the division is securely calculated with SDiv in Section V-B2. Finally, SecTable outputs a distributed decision table consisting of D tests and 2^D secret-shared output values. Specifically, all participants know the split feature F_d at the d-th level where $d \in [0, D-1]$, but each split threshold t_d is only available to the participant who owns the feature F_d.

C. Secure Distributed Decision Table Inference

In Privet, each decision table in the ensemble learned in the secure training phase is held by the participants in a distributed manner, where each participant holds a part of it. Recall that in our secure VFL framework (Algorithm 2), once a distributed decision table $\mathcal{T}$ is securely learned in a certain round, we need to perform secure inference over the training data using $\mathcal{T}$. The the produced inference results at this round will be securely aggreagted with previous inference results for use in securely training a new distributed decision table in the next round. To prevent the partial model and local data on each participant from leaking during the secure inference process, we propose a secure distributed decision table inference protocol SecInfer, which relies on secure multiplication of indicator vectors to conduct privacy-preserving inference, as shown in Algorithm 7. SecInfer allows the participants to cooperatively perform secure inference on their local data utilizing the distributed ensemble and produce secret-shared inference results while keeping the local data and partial model not unavailable to other participants throughout the inference process. We introduce the design of SecInfer as follows.

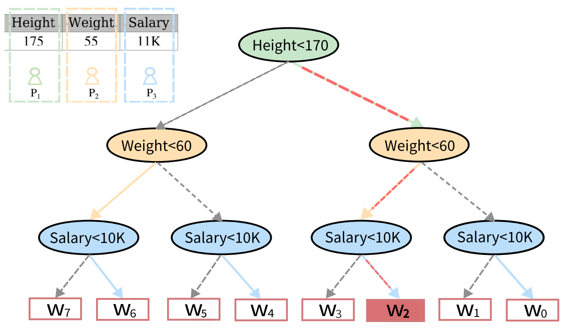


Fig. 4. A simple example of secure distributed decision table inference.

Algorithm 7: Secure Decision Table Inference (SecInfer).

```
Input: $P_1, P_2, \dots, P_n$ hold a vertically partitioned sample
  $\{\mathbf{x}^{J_1}, \mathbf{x}^{J_2}, \dots, \mathbf{x}^{J_n}\}$, a distributed decision table $\mathcal{T}$ of $D$ tests
  and a secret-shared vector $[\![\mathbf{w}]\!]$ containing $2^D$ output values;
Output: $P_1, P_2, \dots, P_n$ obtain the secret-shared inference result
  $[\![w]\!]$.
1: for $d \in [0, D-1]$ in parallel do
2:     $P_l$ who holds the $d$-th test $F_d < t_d$ locally compares
       $\mathbf{x}^{J_l}(F_d)$ with $t_d$.
3:     $P_l$ locally generates leaf indicator $\mathbf{u}_d$ at this level
       according to the outcome of the Boolean test.
4:     $P_l$ secret-shares $\mathbf{u}_d$ to other participants.
5: end for
//$P_1, P_2, \dots, P_n$ perform:
6: $[\![\mathbf{s}]\!] \leftarrow [\![\mathbf{u}_0]\!] \times [\![\mathbf{u}_1]\!] \times \cdots \times [\![\mathbf{u}_{D-1}]\!] \times [\![\mathbf{w}]\!]$.
7: $[\![w]\!] \leftarrow \sum_{i=0}^{2^D-1} [\![\mathbf{s}_i]\!]$.
8: Output the secret-shared inference result $[\![w]\!]$ held by
   $P_1, P_2, \dots, P_n$.
```

To securely produce the inference result of a vertically partitioned sample $\{\mathbf{x}^{J_1}, \mathbf{x}^{J_2}, \dots, \mathbf{x}^{J_n}\}$, the participant P_l who owns the test $F_d < t_d$ at the d-th level ($d \in [0, D-1]$) locally generates a leaf indicator (denoted by $\mathbf{u}_d$) by comparing the sample's feature value of F_d (represented as $\mathbf{x}^{J_l}(F_d)$) with the split threshold t_d. After that, the leaf indicator $\mathbf{u}_d$ is secret-shared to other participants. The inference result could then be obliviously calculated by secure element-wise multiplication between the D secret-shared leaf indicators $\{[\![\mathbf{u}_i]\!]\}_{i=0}^{D-1}$ and the secret-shared vector $[\![\mathbf{w}]\!]$ of decision table's output values. We take a 3-dimensional decision table to present the details of model distribution and SecInfer in Fig. 4. Without loss of generality, we assume that the tests are "$Height < 170$", "$Weight < 60$", and "$Salary < 10000$", held by participants P_1, P_2, P_3, respectively, and the eight leaves' output values are secret-shared among n participants. In Fig. 4, it is noted that the exact test is only visible to the participant owning the corresponding feature, e.g., only P_1 knows the test "$Height < 170$" at the 0-th level.

We take the inference of sample ("$Height = 175$", "$Weight = 55$", "$Salary = 11000$") as an example. The three features are vertically partitioned and held by P_1, P_2, P_3, respectively. For the first test "$Height < 170$", P_1 locally compares "$Height = 175$" with the threshold 170 and generate leaf indicator vector $\mathbf{u}_0 = (0,0,0,0,1,1,1,1)$ to guide the inference path because $175 > 170$. Similarly, we can get $\mathbf{u}_1 = (1,1,0,0,1,1,0,0)$ and $\mathbf{u}_2 = (0,1,0,1,0,1,0,1)$. Each leaf indicator vector is then secret-shared among all participants. Recall that in the basic decision table inference introduced in Section III, the comparisons required by different Boolean tests are parallelized to accelerate inference due to the oblivious tree structure. Although the learned decision tables in the ensemble in Privet are distributed and secret-shared, their oblivious structure remains unchanged, and thus operations at different levels can still be parallelized. After the sharing of leaf indicator vectors, the inference result of this sample can be obliviously calculated by element-wise multiplication as follows: $[\![\mathbf{u}_0]\!] \times [\![\mathbf{u}_1]\!] \times [\![\mathbf{u}_2]\!] \times [\![\mathbf{w}]\!]$. In this way, each participant will not know which path in the distributed decision table is used during the secure inference process.

Remark: We note that there are some existing secure distributed decision tree inference methods [7], [8] in the VFL setting. However, they are not well suited for the required secure distributed decision table inference in Privet. At a high level, these two works and Privet share the common approach of using indicator vectors to enable secure inference. However, the inherent structural differences between decision tables and decision trees result in different methods of generating and utilizing indicator vectors for guiding the inference paths during privacy-preserving inference. In secure distributed decision tree inference of [7], [8], each internal node is associated with an indicator vector, and the inference result is produced by secure multiplication of these indicator vectors. In contrast, for secure distributed decision table inference, each level in the decision table is associated with an indicator vector. As a result, existing secure distributed decision tree inference methods cannot be efficiently extended to secure distributed decision table inference. For instance, with our proposed SecInfer protocol and a decision table with a dimension of $D = 6$, only six indicator vectors and six secure multiplications are needed. However, applying the method from [7], [8] to our target problem would require $2^6 - 1$ indicator vectors and *63* secure multiplications, resulting in poor efficiency. Moreover, the secure distributed tree inference method in [8] is not applicable in our setting because it targets a two-party setting, while Privet aims to support an arbitrary number of participants.

VI. Security Analysis

Privet utilizes standard secret sharing techniques [49] to properly encrypt the intermediate information during both training and inference phases and the secret shares are uniformly distributed in a ring $\mathbb{Z}_{2^Q}$. In addition, throughout the VFL procedure, the feature data owned by each participant is kept locally. We follow the standard simulation-based paradigm [50] to analyze the security of Privet. We first define the ideal functionality of our target privacy-preserving VFL with gradient boosted decision tables as follows.

Definition 1: The ideal functionality $\mathcal{F}_{\mathrm{VDT}}$ of privacy-preserving VFL with gradient boosted decision tables is formulated as follows:

– *Input:* The input to the $\mathcal{F}_{\mathrm{VDT}}$ consists of datasets $\{\mathcal{D}_m^{N \times J_m}\}_{m=1}^n$ and the label set $\mathbf{y}$ from the participants $P_1, P_2, \ldots, P_n$.

– *Computation:* Upon receiving the above input, the ideal functionality $\mathcal{F}_{\mathrm{VDT}}$ performs training of gradient boosted decision tables and produces the trained model VDT, which consists of T decision tables.

– *Output:* The ideal functionality $\mathcal{F}_{\mathrm{VDT}}$ broadcasts the split feature in the decision table to all participants, but only sends the split threshold to the participant who holds the corresponding split feature. Additionally, the $\mathcal{F}_{\mathrm{VDT}}$ splits the output values into n secret shares and then distributes them to the participants $P_1, P_2, \ldots, P_n$.

Definition 2: A protocol Π securely realizes the ideal functionality $\mathcal{F}_{\mathrm{VDT}}$ in the semi-honest adversary setting if a semi-honest participant does not learn any information about other participants' private data and partial model. Formally, let $\mathsf{View}_{P_m}^{\Pi}$ represent participant P_m's view during the execution of Π. Formally, there should exist a PPT simulator, which can generate a simulated view $\mathsf{Sim}_{P_m}^{\Pi}$ such that $\mathsf{Sim}_{P_m}^{\Pi}$ is indistinguishable from $\mathsf{View}_{P_m}^{\Pi}$.

Theorem 1: Our Privet securely realizes the ideal functionality $\mathcal{F}_{\mathrm{VDT}}$ against a semi-honest adversary who can statically corrupt a subset of τ participants ($\tau \leq n-1$) according to Definition 2.

Proof: If the simulator for each sub-protocol exists, then our complete protocol is secure [51], [52], [53]. As presented before, Privet consists of several secure sub-protocols: 1) secure division SDiv; 2) secure Sigmoid $\mathsf{SecSigmoid}$; 3) secure node splitting $\mathsf{SecSplit}$; 4) secure distributed decision table inference $\mathsf{SecInfer}$; 5) secure discretization $\mathsf{SecDisc}$; 6) secure Argmin $\mathsf{SecArgmin}$. We use $\mathsf{Sim}_{P_m}^{\mathsf{X}}$ as the simulator which can generate P_m's view in sub-protocol X ($\mathsf{X} \in \{\mathsf{SDiv}, \mathsf{SecSigmoid}, \mathsf{SecSplit}, \mathsf{SecInfer}, \mathsf{SecDisc}, \mathsf{SecArgmin}\}$) on corresponding input and output. Obviously, the simulators for $\mathsf{X} \in \{\mathsf{SDiv}, \mathsf{SecSigmoid}, \mathsf{SecArgmin}\}$ must exist, because they are comprised of basic operations in the secret sharing domain [49]. In the execution of these three protocols, even if a subset of τ participants is corrupted, the honest participants' private data will not be leaked due to the security guarantee of additive secret sharing [45]. In addition, when revealing the index of the minimum value in a secret-shared vector to all participants in SecArgmin, the simulator can adjust the shares of the result such that the revealed index is indeed the value received from the ideal functionality, and thus the simulator of SecArgmin exists. Therefore, Privet is secure if the simulators for the remaining sub-protocols exist, i.e., SecSplit in Section V-B1, SecInfer in Section V-C, SecDisc in Section V-B3. We next provide the existence of the simulators for the remaining sub-protocols. $\square$

Theorem 2: The simulators for sub-protocols SecSplit and SecInfer exsit.

Proof: The sub-protocols SecSplit and SecInfer both require that the participant who holds the split generates indicator vectors locally and then secret-shares them to other participants. For simplicity, we assume that P_l holds the split threshold and collaborates with $P_{m\neq l}$ to split the node and conduct inference. We then prove the existence of the simulators $\mathsf{Sim}_{P_l}^{\mathsf{SecSplit}}$, $\mathsf{Sim}_{P_l}^{\mathsf{SecInfer}}$ for P_l and the simulators $\mathsf{Sim}_{P_{m\neq l}}^{\mathsf{SecSplit}}$, $\mathsf{Sim}_{P_{m\neq l}}^{\mathsf{SecInfer}}$ for $P_{m\neq l}$ due to their different computation. We also note that there is no difference in the simulation between AP and PP due to their role equivalence in the execution of SecSplit and SecInfer.

- $\mathsf{Sim}_{P_l}^{\mathsf{SecSplit}}$, $\mathsf{Sim}_{P_l}^{\mathsf{SecInfer}}$ for P_l: The simulator is simple since P_l only secret-shares its local indicator vectors and perform secure multiplications on the secret-shared vectors. In the execution of SecSplit and SecInfer, P_l receives nothing. Moreover, since secure multiplication is the basic operation in the secret sharing domain, the privacy of the honest participants' data is ensured even if P_l colludes with $\tau - 1$ participants. Therefore, it is clear that the simulated view is indistinguishable from the real view.
- $\mathsf{Sim}_{P_{m\neq l}}^{\mathsf{SecSplit}}$, $\mathsf{Sim}_{P_{m\neq l}}^{\mathsf{SecInfer}}$ for $P_{m\neq l}$: In the execution of SecSplit and SecInfer, the only information $P_{m\neq l}$ receives is the secret share (denoted by $\langle \mathbf{r} \rangle_{m\neq l}$) of P_l's indicator vector $\mathbf{r}$. The secret share $\langle \mathbf{r} \rangle_{m\neq l}$ is randomly generated at P_l and thus is uniformly random in $P_{m\neq l}$'s view. Therefore, the distribution over the real secret share $\langle \mathbf{r} \rangle_{m\neq l}$ received by $P_{m\neq l}$ in the execution and over the simulated $\langle \mathbf{r} \rangle_{m\neq l}$ generated by the simulator are identically distributed. Furthermore, the secure multiplication involved is the basic operation in the secret sharing domain, which means $P_{m\neq l}$ learns no additional information from the execution, even if $P_{m\neq l}$ colludes with $\tau - 1$ participants. Therefore, the simulated view is indistinguishable from the real view. □

Theorem 3: The simulator for the sub-protocol SecDisc exists.

Proof: The security of SecDisc relies on the security of its sub-protocol SecPerm because the operations in SecDisc besides SecPerm are basic operations in the secret sharing domain. Therefore, if the protocol SecPerm can be simulated, the existence of a simulator for the protocol SecDisc follows. In SecPerm, for simplicity, we assume that P_l sorts the values for a feature locally to generate a permutation π. We then prove the existence of the simulator $\mathsf{Sim}_{P_l}^{\mathsf{SecDisc}}$ for P_l and the simulator $\mathsf{Sim}_{P_{m\neq l}}^{\mathsf{SecDisc}}$ for $P_{m\neq l}$ due to their different computation. Similarly, since the role equivalence of AP and PP in the execution of SecDisc, there is no difference in the analysis for them.

- $\mathsf{Sim}_{P_l}^{\mathsf{SecDisc}}$ for P_l: In the execution of SecDisc, P_l only receives secret share $\langle \mathbf{x} - \mathbf{r} \rangle_{m\neq l}$ of $[\![\mathbf{x} - \mathbf{r}]\!]$ from $P_{m\neq l}$ to reconstruct $\mathbf{x} - \mathbf{r}$. In the simulated view, P_l receives $n - 1$ random vectors. Therefore, we need to prove that $\langle \mathbf{x} - \mathbf{r} \rangle_{m\neq l}$ is uniformly random in the view of P_l. Obviously, the above claim is valid, because $\langle \mathbf{x} - \mathbf{r} \rangle_{m\neq l}$ is a random vector generated at $P_{m\neq l}$, which must be uniformly random in the view of P_l. Therefore the distributions over the real $\langle \mathbf{x} - \mathbf{r} \rangle_{m\neq l}$ received by P_l in the protocol execution and over the simulated $\langle \mathbf{x} - \mathbf{r} \rangle_{m\neq l}$ are identically distributed. Moreover, even if P_l colludes with $\tau - 1$ participants, P_l still only learns a randomly masked version of $\mathbf{x}$, thus $\mathbf{x}$ would not be leaked to P_l. Thus, the simulator for $\mathsf{Sim}_{P_l}^{\mathsf{SecPerm}}$ exists, which indicates that the simulator for $\mathsf{Sim}_{P_l}^{\mathsf{SecDisc}}$ also exists.
- $\mathsf{Sim}_{P_{m\neq l}}^{\mathsf{SecDisc}}$ for $P_{m\neq l}$: In the execution of SecDisc, $P_{m\neq l}$ only receives permutation π_s from P_l. The permutation π_s is randomly generated at P_l and is uniformly random in $P_{m\neq l}$'s view. Therefore, the distribution over the real permutation π_s received by $P_{m\neq l}$ in the execution and over the simulated π_s generated by the simulator is identically distributed. In case of corruption, there are two situations: 1) If participant P_l is corrupted, nothing is revealed regarding the vector $\mathbf{x}$. 2) If participant P_l is not corrupted, the private permutation π on P_l and the private vector $\mathbf{x}$ are protected, even if all other participants collude. This security guarantee comes from the fact that the corrupted participants could learn nothing about π_p. Without the knowledge of π_p, π cannot be deduced because $\pi(\cdot) = \pi_s[\pi_p(\cdot)]$. Thus, the simulator for $\mathsf{Sim}_{P_{m\neq l}}^{\mathsf{SecPerm}}$ exists, which indicates that the simulator for $\mathsf{Sim}_{P_{m\neq l}}^{\mathsf{SecDisc}}$ also exists. □

TABLE II
STATISTICS OF DATASETS

Dataset Type	Name	Samples	Features	Task
Real-world	Cal Housing[1]	20,640	8	Regression
	Credit[2]	30,000	23	Classification
	Breast Cancer[3]	570	30	Classification
Synthetic	Syn^A	20,000	80	Regression
	Syn^B	20,000	60	Regression
	Syn^C	20,000	40	Classification

VII. EXPERIMENTS

A. Setup

We implement our protocols in Python. All experiments are performed on a workstation with 16 Intel I7- 10700 K cores, 64 GB RAM, and 1 TB SSD external storage, running Ubuntu 20.04.2 LTS. It is worth mentioning that the practice of evaluating VFL algorithms on a single machine also exists in prior works [7], [8], [10], [54]. We also note that in practice, it is not common to have VFL scenarios with more than four participants since it could be hard to bring together many enterprises [9], [55]. Therefore, in our experiments, we follow prior works [7], [9], [55], [56] to conduct experiments with four participants. The communication between participants on the workstation is emulated by the loopback filesystem, where the delay is set to 5 ms and bandwidth is set to 100 Mbps.

Datasets: We use three real-world datasets to evaluate the accuracy and efficiency of Privet and three synthetic datasets to further evaluate the scalability of Privet. The synthetic datasets are generated with *sklearn*[4] library. Table II summarizes the

[1]https://scikit-learn.org/stable/modules/generated/sklearn.datasets.fetch_california_housing.html
[2]https://www.kaggle.com/datasets/uciml/default-of-credit-card-clients-dataset
[3]https://www.kaggle.com/datasets/uciml/breast-cancer-wisconsin-data
[4]https://scikit-learn.org/stable/

statistics of the six datasets. We divide each dataset into two parts for training and testing respectively according to the ratio of 8:2. In addition to the preprocessing in the above, each dataset is split and distributed to all participants vertically and evenly. Similar to previous works on VFL [8], [11], we assume that the records in each participant's database have been properly aligned beforehand.

Protocol instantiation: Our protocols are instantiated using the following parameter settings. We use the ring $\mathbb{Z}_{2^{64}}$ with the number of precision bits $l = 20$. The number of iterations for the reciprocal approximation is set to 20 (with $1/Y = 1/2^{20}$ as the initialization). For approximating the exponential function, we set $2^n = 4$, thus the approximation requires $\log 2^n = 2$ rounds of multiplication. The hyper-parameters are public to all participants. We fix $\lambda = 1$ of (6) and vary the public parameter T (the number of decision tables), B (the number of buckets), and D (the dimension of decision tables) in our experiments to demonstrate the utility, efficiency, and scalability of Privet.

It is noted that to handle real numbers for secure computation, we follow the common practice of fixed-point representation, where real numbers are scaled by a factor of 2^l (l represents the number of precision bits) and then rounded. As a result, when two scaled values are multiplied, the result is under a scaling factor of 2^{2l}. Therefore, truncation is required to scale down the multiplication result, making its scaling factor 2^l again. Privet resorts to the method in [45] to support secure truncation on a result $[\![z]\!]$ produced from secret-shared multiplication, which works as follows. First, the secret sharing of the number of wraps (denoted by $[\![\theta_z]\!]$) in z needs to be computed, which is subject to $\theta_z = (\sum_{m=1}^{n}\langle z\rangle_m - z)/2^Q$. It is noted that to count the number of wrap rounds in this truncation protocol, computing the sum of shares (represented in the form of $\sum_{m=1}^{n}\langle z\rangle_m$) does not involve modular arithmetic. The computation of $[\![\theta_z]\!]$ proceeds as follows. All parties hold in advance a secret-shared random value $[\![r]\!]$ and its secret-shared wrap count $[\![\theta_r]\!]$ subject to $\theta_r = (\sum_{m=1}^{n}\langle r\rangle_m - r)/2^Q$. For secure truncation, all parties first compute $[\![p]\!] = [\![z + r]\!]$. After that, each party P_m computes the differential wraps produced between its shares of $[\![z]\!]$, $[\![r]\!]$, and $[\![p]\!]$: $\langle\beta_{zr}\rangle_m = (\langle z\rangle_m + \langle r\rangle_m - \langle p\rangle_m)/2^Q$, where no modulo operation is required in calculating $\langle z\rangle_m + \langle r\rangle_m - \langle p\rangle_m$. Next, $P_{m\neq 1}$ sends $\langle p\rangle_{m\neq 1}$ to P_1 to reconstruct p and P_1 also computes $\theta_p = (\sum_{m=1}^{n}\langle p\rangle_m - p)/2^Q$. Finally, each P_m produces the secret share $\langle\theta_z\rangle_m = j \times \theta_p + \langle\beta_{zr}\rangle_m - \langle\theta_r\rangle_m$, where $j = 1$ if $m = 1$ and $j = 0$ if $m \neq 1$. Then $[\![\theta_z]\!]$ is used to correct the truncation: $[\![z]\!] = \frac{[\![z]\!] - [\![\theta_z]\!]2^Q}{2^l}$. The above method only needs 1 online communication round to compute the number of wraps $[\![\theta_z]\!]$, making it efficient and practical for use in Privet.

B. Utility Evaluation

We first compare the accuracy of two approaches: our Privet and plaintext centralized learning of gradient boosted decision tables. For the Cal Housing dataset, we set $T = 50$ to build 50 decision tables in the ensemble model and $D = 5$ to limit the dimension of each decision table. The number of buckets B is set as 32. For the Credit dataset, we set $T = 10$, $D = 4$ and $B = 32$. For the Breast Cancer dataset, we set $T = 10$, $D = 3$ and $B = 32$. Following other works on gradient boosting [8], more decision tables are trained for regression tasks to guarantee accuracy. We use the identical parameters in Privet and plaintext. For the regression tasks, we use the Root Mean Square Error (RMSE) as the evaluation metric. For the evaluation of classification tasks, we report the results using two commonly used metrics: Accuracy (ACC) and Area Under the ROC Curve (AUC). The accuracy of Privet and plaintext on both regression and classification tasks are reported in Table III.

TABLE III
ACCURACY COMPARISON

Dataset	Method	RMSE	ACC/AUC
Cal Housing	Plaintext	0.51	-
	Privet	0.51	-
Credit	Plaintext	-	80.3%/0.7438
	Privet	-	80.3%/0.7433
Breast Cancer	Plaintext	-	96.5%/0.999
	Privet	-	96.5%/0.998

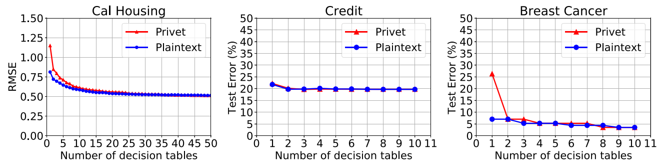


Fig. 5. RMSE/test error on the three real-world datasets, for different numbers of decision tables.

Fig. 5 shows the RMSE/test error on the three real-world datasets, for varying number of decision tables. Note that the test error is defined as the complement of the Accuracy (i.e., 1−ACC). It is observed that the difference in utility between Privet and plaintext is obvious at the very beginning, but the difference rapidly diminishes as the increase of number of decision tables in the ensemble. This indicates the similar convergence behavior of Privet and plaintext. From the above results, we can conclude that our Privet achieves compatible accuracy with plaintext centralized learning of gradient boosted decision tables on both classification and regression tasks.

TABLE IV
PRIVET'S COMPUTATION AND COMMUNICATION PERFORMANCE

Dataset	Online Secure Training	
	Time (seconds)	Comm. (GB)
Cal Housing	6333	41.1
Credit	1724	11.8
Breast Cancer	486	0.54

C. Efficiency Evaluation

We now report the computation and communication performance of Privet in secure training over the three public datasets, and present the results in Table IV. We also note that to evaluate the efficiency of Privet over larger datasets, we use three synthetic datasets, of which the results are reported in Section VII-D.

From the results in the first two records of Table IV, we can observe that the training time and communication cost of Privet

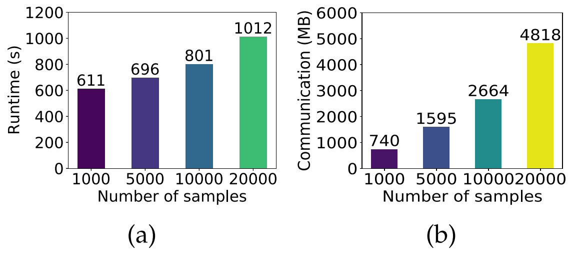


Fig. 6. Performance on Syn^A dataset with 80 features for different numbers of training samples (with the dimension of decision tables $D = 2$, the number of buckets $B = 32$ and the number of decision tables $T = 10$): (a) Runtime; (b) Communication.

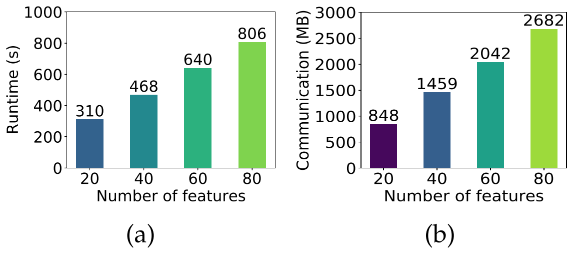


Fig. 7. Performance on Syn^A dataset with 10000 samples for different numbers of features (with the dimension of decision tables $D = 2$, the number of buckets $B = 32$ and the number of decision tables $T = 10$): (a) Runtime; (b) Communication.

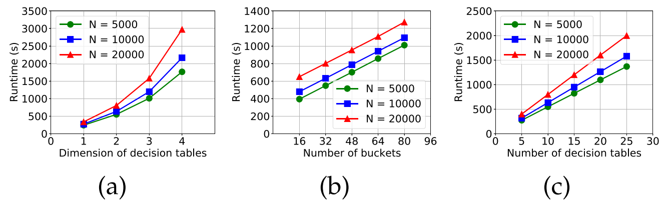


Fig. 8. Runtime performance on Syn^B with 60 features for different numbers of training samples N and (a) varying dimension of decision tables D, (b) varying number of buckets B, and (c) varying number of decision tables T, respectively.

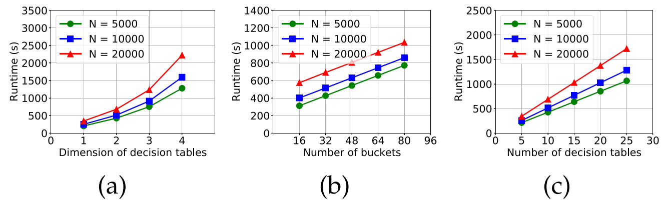


Fig. 9. Runtime performance on Syn^C with 40 features for different numbers of training samples N and (a) varying dimension of decision tables D, (b) varying number of buckets B, and (c) varying number of decision tables T, respectively.

on Cal Housing is significantly more than that on Credit, which is because we train 50 decision tables on Cal Housing but only 10 decision tables on Credit. However, Privet consumes an average of 127 seconds to train a 5-dimensional decision table on Cal Housing and 172 seconds to train a 4-dimensional decision table on Credit. There are mainly two reasons for this observation: (1) the number of features and the number of samples of Cal Housing are less than that of Credit. (2) dealing with classification tasks additionally needs secure Sigmoid evaluation compared with regression tasks, which will result in more computation and communication overhead.

From the results in the last two records of Table IV, we can observe that although the scale of Credit is nearly 50 times larger than that of Breast Cancer, the training time on Credit under similar parameters is only roughly 4 times that on Breast Cancer. This is because Privet securely discretizes training data into buckets, and all the time-consuming computations are conducted on the buckets. In this way, the training cost is highly correlated with the number of buckets, instead of the number of training samples. In fact, this also indicates the strong scalability of Privet, which will be demonstrated in detail in the next section.

D. Scalability Evaluation

We now evaluate the scalability of Privet. To examine how the number of training samples and features affect the cost of secure online training, we conduct an experiment using the Syn^A dataset with 20,000 samples and 80 features. We report the results in Figs. 6 and 7, which show the runtime and communication cost for varying numbers of samples and features, respectively. For the experiment related to Fig. 6, we fix the number of features as 80, the dimension D as 2, the number of buckets B as 32, and the number of decision tables T as 10, for varying number of training samples by randomly selecting samples from Syn^A. It is observed that the runtime is not much affected by the number of training samples from Fig. 6(a). At the same time, the communication cost grows linearly with the number of training samples as shown in Fig. 6(b). For the experiment related to Fig. 7, we randomly select 10,000 samples from Syn^A and fix the dimension D as 2, the number of buckets B as 32, and the number of decision tables T as 10, for varying number of features by randomly selecting features from Syn^A. From Fig. 7(a) and (b), we can observe that both the runtime and communication cost of Privet increase proportionally with the number of features, in line with the complexity of Algorithm 6, where the main loop in each level enumerates features.

Next, we examine the impact of dimension D, the number of buckets B, and the number of decision tables T on the runtime of secure training. We use the synthetic regression dataset Syn^B with 20,000 samples and 60 features for the regression task, and the synthetic classification dataset Syn^C with 20,000 samples and 40 features for the classification task. We first examine the relationship between dimension D and runtime. For both regression and classification tasks, we set $B = 32$, $T = 10$, and vary the dimension D, over varying number of training samples. The results are shown in Figs. 8(a) and 9(a), from which we can observe that the runtime grows exponentially with the increase of dimension. The results are consistent with the complexity of our secure training algorithm because the number of tree nodes is exponentially related to the dimension. The computation in each node accounts for the largest proportion of all calculations. We then examine the relationship between the number of buckets

B and runtime. We set $T = 10$, $D = 2$, and vary B, over varying number of training samples. The evaluation is also performed over both Syn^B and Syn^C. Figs. 8(b) and 9(b) show the results, which indicate the linear association between B and runtime. Finally, we evaluate the relationship between the number of decision tables T and runtime, and summarize the results in Figs. 8(c) and 9(c). We set T from 5 to 25, while keeping $B = 32$ and $D = 2$, on both Syn^B and Syn^C with varying number of training samples. Recall that in Privet an ensemble of decision tables is securely built. The training time of each decision table is roughly the same when we set the same parameters for them. So the runtime must grow linearly with the number of decision tables, which is consistent with the results in Figs. 8(c) and 9(c). In summary, the above evaluation results demonstrate that Privet is scalable and capable of handling large-scale datasets with a large number of features and training samples.

VIII. Conclusion

In this article, we design, implement, and evaluate Privet, the first system framework enabling privacy-preserving VFL service for gradient boosted decision tables. Building on lightweight secret sharing techniques, Privet supports an arbitrary number of distributed participants to collaboratively train gradient boosted decision tables over vertically partitioned distributed datasets, offering strong protection for individual data as well as for intermediate outputs. Extensive experiments on several real-world datasets and synthetic datasets demonstrate that Privet achieves promising performance, with model utility comparable to the case of plaintext centralized learning. For future work, it would be an interesting direction to explore the possibility of leveraging GPUs to achieve further performance boost.

References

[1] K. Bonawitz et al., “Practical secure aggregation for privacy-preserving machine learning,” in *Proc. ACM Conf. Comput. Commun. Secur.*, 2017, pp. 1175–1191.

[2] C. Qiao, K. N. Brown, F. Zhang, and Z. Tian, “Federated adaptive asynchronous clustering algorithm for wireless mesh networks,” *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 3, pp. 2610–2627, Mar. 2023.

[3] P. Zhou, K. Wang, L. Guo, S. Gong, and B. Zheng, “A privacy-preserving distributed contextual federated online learning framework with Big Data support in social recommender systems,” *IEEE Trans. Knowl. Data Eng.*, vol. 33, no. 3, pp. 824–838, Mar. 2021.

[4] J. Zhao et al., “CORK: A privacy-preserving and lossless federated learning scheme for deep neural network,” *Inf. Sci.*, vol. 603, pp. 190–209, 2022.

[5] Q. Li, Z. Wen, and B. He, “Practical federated gradient boosting decision trees,” in *Proc. Conf. Assoc. Advance. Artif. Intell.*, 2020, pp. 4642–4649.

[6] S. Maddock, G. Cormode, T. Wang, C. Maple, and S. Jha, “Federated boosted decision trees with differential privacy,” in *Proc. ACM Conf. Comput. Commun. Secur.*, 2022, pp. 2249–2263.

[7] L. Xie, J. Liu, S. Lu, T.-H. Chang, and Q. Shi, “An efficient learning framework for federated XGBoost using secret sharing and distributed optimization,” *ACM Trans. Intell. Syst. Technol.*, vol. 13, no. 5, pp. 1–28, 2022.

[8] W. Fang et al., “Large-scale secure XGB for vertical federated learning,” in *Proc. ACM Int. Conf. Inf. Knowl. Manage.*, 2021, pp. 443–452.

[9] F. Fu et al., “VF^2 boost: Very fast vertical federated gradient boosting for cross-enterprise learning,” in *Proc. ACM Int. Conf. Manage. Data*, 2021, pp. 563–576.

[10] K. Cheng et al., “SecureBoost: A lossless federated learning framework,” *IEEE Intell. Syst.*, vol. 36, no. 6, pp. 87–98, Nov./Dec. 2021.

[11] Z. Tian, R. Zhang, X. Hou, J. Liu, and K. Ren, “FederBoost: Private federated learning for GBDT,” 2020, *arXiv: 2011.02796*.

[12] S. Tyree, K. Q. Weinberger, K. Agrawal, and J. Paykin, “Parallel boosted regression trees for web search ranking,” in *Proc. ACM Int. Conf. World Wide Web*, 2011, pp. 387–396.

[13] X. He et al., “Practical lessons from predicting clicks on ads at Facebook,” in *Proc. ACM Int. Workshop Data Mining Online Advertising*, 2014, pp. 1–9.

[14] N. Dhieb, H. Ghazzai, H. Besbes, and Y. Massoud, “Extreme gradient boosting machine learning algorithm for safe auto insurance operations,” in *Proc. IEEE Int. Conf. Veh. Electron. Saf.*, 2019, pp. 1–5.

[15] R. Kohavi and C. Li, “Oblivious decision trees, graphs, and top-down pruning,” in *Proc. Int. Joint Conf. Artif. Intell.*, 1995, pp. 1071–1079.

[16] Y. Lou and M. Obukhov, “BDT: Gradient boosted decision tables for high accuracy and scoring efficiency,” in *Proc. ACM Int. Conf. Knowl. Discov. Data Mining*, 2017, pp. 1893–1901.

[17] L. O. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin, “CatBoost: Unbiased boosting with categorical features,” in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2018, pp. 6639–6649.

[18] D. Dato et al., “Fast ranking with additive ensembles of oblivious and non-oblivious regression trees,” *ACM Trans. Inf. Syst.*, vol. 35, no. 2, pp. 15:1–15:31, 2016.

[19] J. T. Hancock and T. M. Khoshgoftaar, “Catboost for Big Data: An interdisciplinary review,” *J. Big Data*, vol. 7, no. 1, 2020, Art. no. 94.

[20] I. Kuralenok, V. Ershov, and I. Labutin, “MonoForest framework for tree ensemble analysis,” in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2019, pp. 13777–13786.

[21] G. Capannini, C. Lucchese, F. M. Nardini, S. Orlando, R. Perego, and N. Tonellotto, “Quality versus efficiency in document scoring with learning-to-rank models,” *Inf. Process. Manag.*, vol. 52, no. 6, pp. 1161–1177, 2016.

[22] G. Ke et al., “LightGBM: A highly efficient gradient boosting decision tree,” in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 3149–3157.

[23] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*. Belmont, CA, USA: Wadsworth, 1984.

[24] J. R. Quinlan, *C4. 5: Programs for Machine Learning*. Amsterdam, The Netherlands: Elsevier, 2014.

[25] C. Leung, A. Law, and O. Sima, “Towards privacy-preserving collaborative gradient boosted decision trees,” UC Berkeley, Tech. Rep., 2019.

[26] F. McKeen et al., “Innovative instructions and software model for isolated execution,” in *Proc. Workshop Hardware Archit. Support Secur. Privacy*, 2013, pp. 1–1.

[27] P. Mohassel and Y. Zhang, “SecureML: A system for scalable privacy-preserving machine learning,” in *Proc. IEEE Symp. Secur. Privacy*, 2017, pp. 19–38.

[28] A. Gulin, I. Kuralenok, and D. Pavlov, “Winning the transfer learning track of Yahoo!’s learning to rank challenge with YetiRank,” *Yahoo! Learn. Rank Challenge*, ser. JMLR Proceedings, vol. 14, 2011, pp. 63–76.

[29] J. Dhar and A. K. Jodder, “An effective recommendation system to forecast the best educational program using machine learning classification algorithms,” *Ingénierie Des Systèmes D Inf.*, vol. 25, no. 5, pp. 559–568, 2020.

[30] D. C. Yadav and S. Pal, “An experimental study of diversity of diabetes disease features by bagging and boosting ensemble method with rule based machine learning classifier algorithms,” *SN Comput. Sci.*, vol. 2, no. 1, 2021, Art. no. 50.

[31] P.-C. Liao et al., “Integrating health data-driven machine learning algorithms to evaluate risk factors of early stage hypertension at different levels of HDL and LDL cholesterol,” *Diagnostics*, vol. 12, no. 8, 2022, Art. no. 1965.

[32] M. Abspoel, D. Escudero, and N. Volgushev, “Secure training of decision trees with continuous attributes,” *Proc. Priv. Enhancing Technol.*, vol. 2021, no. 1, pp. 167–187, 2021.

[33] K. Deforth, M. Desgroseilliers, N. Gama, M. Georgieva, D. Jetchev, and M. Vuille, “XORBoost: Tree boosting in the multiparty computation setting,” *Proc. Priv. Enhancing Technol.*, vol. 2022, no. 4, pp. 66–85, 2022.

[34] S. Adams et al., “Privacy-preserving training of tree ensembles over continuous data,” *Proc. Priv. Enhancing Technol.*, vol. 2022, no. 2, pp. 205–226, 2022.

[35] T. Chen and C. Guestrin, “XGBoost: A scalable tree boosting system,” in *Proc. ACM Int. Conf. Knowl. Discov. Data Mining*, 2016, pp. 66–85.

[36] J. Friedman, T. Hastie, and R. Tibshirani, “Additive logistic regression: A statistical view of boosting,” *Ann. Statist.*, vol. 28, no. 2, pp. 337–407, 2000.

[37] XGBoost Parameters, 2022. Accessed: Oct. 26, 2022. [Online]. Available: https://xgboost.readthedocs.io/en/stable/parameter.html

[38] Catboost Score Functions, 2023. Accessed: Oct. 26, 2022. [Online]. Available: https://catboost.ai/en/docs/concepts/algorithm-score-functions

[39] D. Beaver, "Efficient multiparty protocols using circuit randomization," in *Proc. Annu. Int. Cryptol. Conf.*, 1991, pp. 420–432.

[40] M. S. Riazi, C. Weinert, O. Tkachenko, E. M. Songhori, T. Schneider, and F. Koushanfar, "Chameleon: A hybrid secure computation framework for machine learning applications," in *Proc. ACM Asia Conf. Comput. Commun. Secur.*, 2018, pp. 707–721.

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

[42] "European Parliament and the Council: The General Data Protection Regulation (GDPR)," 2016. [Online]. Available: https://eur-lex.europa.eu/eli/reg/2016/679/oj

[43] J. Verbeke and R. Cools, "The Newton-Raphson method," *Int. J. Math. Educ. Sci. Technol.*, vol. 26, no. 2, pp. 177–193, 1995.

[44] S. Tan, B. Knott, Y. Tian, and D. J. Wu, "CryptGPU: Fast privacy-preserving machine learning on the GPU," in *Proc. IEEE Symp. Secur. Privacy*, 2021, pp. 1021–1038.

[45] B. Knott, S. Venkataraman, A. Y. Hannun, S. Sengupta, M. Ibrahim, and L. van der Maaten, "CrypTen: Secure multi-party computation meets machine learning," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2021, pp. 4961–4973.

[46] M. Kim et al., "Secure logistic regression based on homomorphic encryption: Design and evaluation," *JMIR Med. Inform.*, vol. 6, no. 2, 2018, Art. no. e8805.

[47] P. Mohassel and P. Rindal, "ABY: A mixed protocol framework for machine learning," in *Proc. ACM Conf. Comput. Commun. Secur.*, 2018, pp. 35–52.

[48] X. Liu, Y. Zheng, X. Yuan, and X. Yi, "MediSC: Towards secure and lightweight deep learning as a medical diagnostic service," in *Proc. Eur. Symp. Res. Comput. Secur.*, 2021, pp. 519–541.

[49] D. Demmler, T. Schneider, and M. Zohner, "ABY - A framework for efficient mixed-protocol secure two-party computation," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2015, pp. 1–15.

[50] Y. Lindell, "How to simulate it - A tutorial on the simulation proof technique," in *Tutorials on the Foundations of Cryptography*. Berlin, Germany: Springer, 2017, pp. 277–346.

[51] R. Canetti, "Security and composition of multiparty cryptographic protocols," *J. Cryptol.*, vol. 13, no. 1, pp. 143–202, 2000.

[52] J. Katz and Y. Lindell, "Handling expected polynomial-time strategies in simulation-based security proofs," in *Proc. Theory Cryptography Conf.*, 2005, pp. 128–149.

[53] M. Curran, X. Liang, H. Gupta, O. Pandey, and S. R. Das, "Procsa: Protecting privacy in crowdsourced spectrum allocation," in *Proc. Eur. Symp. Res. Comput. Secur.*, 2019, pp. 556–576.

[54] X. Li et al., "OpBoost: A vertical federated tree boosting framework based on order-preserving desensitization," *Proc. VLDB Endow.*, vol. 16, no. 2, pp. 202–215, 2022.

[55] X. Jin, P.-Y. Chen, C.-Y. Hsu, C.-M. Yu, and T. Chen, "Cafe: Catastrophic data leakage in vertical federated learning," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2021, pp. 994–1006.

[56] R. Wang, O. Ersoy, H. Zhu, Y. Jin, and K. Liang, "FEVERLESS: Fast and secure vertical federated learning based on XGBoost for decentralized labels," *IEEE Trans. Big Data*, to be published, doi: 10.1109/TBDATA.2022.3227326.

Yifeng Zheng received the PhD degree in computer science from the City University of Hong Kong, Hong Kong, in 2019. He is an assistant professor with the School of Computer Science and Technology, Harbin Institute of Technology, Shenzhen, China. He worked as a postdoc with the Commonwealth Scientific and Industrial Research Organization (CSIRO), Australia, and City University of Hong Kong, respectively. His work has appeared in prestigious conferences (such as ESORICS, DSN, AsiaCCS, and PERCOM) and journals (such as *IEEE Transactions on Dependable and Secure Computing*, *IEEE Transactions on Information Forensics and Security*, *IEEE Transactions on Services Computing*, and *IEEE Transactions on Knowledge and Data Engineering*). His current research interests are focused on security and privacy related to cloud computing, IoT, machine learning, and multimedia.

Shuangqing Xu received the BE degree in computer science from Nanjing Tech University, Nanjing, China, in 2021. He is currently working toward the graduation degree with the Harbin Institute of Technology, Shenzhen, China. His current research interests include machine learning, data security, and federated learning.

Songlei Wang received the BE degree in Internet of Things from the China University of Petroleum (East China), Qingdao, China, in 2018 and the ME degree in computer technology from the Harbin Institute of Technology, Shenzhen, China, in 2021. He is currently working toward the PhD degree with the School of Computer Science and Technology, Harbin Institute of Technology, Shenzhen, China. His research interests include cloud computing security and secure machine learning.

Yansong Gao (Senior Member, IEEE) received the MSc degree from the University of Electronic Science and Technology of China in 2013 and the PhD degree from the University of Adelaide, Australia, in 2017. He is a tenured research scientist with Data61, CSIRO. Prior to that, he was an associate professor with the Nanjing University of Science and Technology. His current research interests are AI security and privacy, hardware security, and system security.

Zhongyun Hua (Senior Member, IEEE) received the BS degree from Chongqing University, Chongqing, China, in 2011, and the MS and PhD degrees in software engineering from the University of Macau, Macau, China, in 2013 and 2016, respectively. He is currently an associate professor with the School of Computer Science and Technology, Harbin Institute of Technology, Shenzhen, Shenzhen, China. His research interests include chaotic system and information security.