PIVODL
Privacy-Preserving Vertical Federated Learning Over Distributed Labels

Zhu, Hangyu; Wang, Rui; Jin, Yaochu; Liang, Kaitai

DOI
10.1109/TAI.2021.3139055

Publication date
2023

Document Version
Final published version

Published in
IEEE Transactions on Artificial Intelligence

PIVODL: Privacy-Preserving Vertical Federated Learning Over Distributed Labels

Hangyu Zhu, Rui Wang, Yaochu Jin, *Fellow, IEEE*, and Kaitai Liang, *Member, IEEE*

***Abstract*—Federated learning (FL) is an emerging privacy preserving machine learning protocol that allows multiple devices to collaboratively train a shared global model without revealing their private local data. Nonparametric models like gradient boosting decision trees (GBDTs) have been commonly used in FL for vertically partitioned data. However, all these studies assume that all the data labels are stored on only one client, which may be unrealistic for real-world applications. Therefore, in this article, we propose a secure vertical FL framework, named privacy-preserving vertical federated learning system over distributed labels (PIVODL), to train GBDTs with data labels distributed on multiple devices. Both homomorphic encryption and differential privacy are adopted to prevent label information from being leaked through transmitted gradients and leaf values. Our experimental results show that both information leakage and model performance degradation of the proposed PIVODL are negligible.**

***Impact Statement*—Federated learning is a distributed machine learning framework proposed for privacy preservation. Most federated learning algorithms work on horizontally partitioned data, with only a few exceptions considering vertically partitioned data that is widely seen in the real world. However, existing vertical federated learning makes an unrealistic assumption that data labels are distributed on only one device and no research has been reported so far that considers data labels distributed on multiple client devices. The PIVODL framework reported in this article allows us to build a secure vertical federated XGBoost system, in which the labels may distributed either on one device or on multiple devices, making it possible to apply federated learning to a wider range of real-world problems.**

***Index Terms*—Encryption, gradient boosting decision tree (GBDT), privacy preservation, vertical federated learning (VFL).**

I. Introduction

DATA privacy has become the main focus of attention in modern societies and the recently enacted General Data Protection Regulation prohibits users from wantonly sharing and exchanging their personal data. This may be a big barrier to model training, since standard centralized machine learning algorithms require to collect and store training data on one single cloud server. To tackle this issue, federated learning (FL) [1] is proposed to enable multiple edge devices to collaboratively train a shared global model while keeping all the users' data on local devices.

FL can be categorized into horizontal federated learning (HFL) and vertical federated learning (VFL) based on how data are partitioned [2]. HFL or instance-based FL represents the scenarios in which the users' training data share the same feature space but have different samples. A large amount of research work [3]–[7] are dedicated on enhancing the security level of HFL, since recent studies [8]–[11] have shown that the HFL protocol still suffer from potential risks of leaking local private data information. Secure multiparty computation [12], [13], homomorphic encryption (HE) [14], and differential privacy (DP) [15] are three most common privacy preserving mechanisms that are theoretically and empirically proved to be effective for HFL.

Compared with HFL, VFL is more likely to appear in the real-world applications. And the training data of participating clients in VFL have the same sample identity (ID) space but with different feature space. Privacy preservation is also a critical concern in VFL. Hardy *et al.* [16] introduce a secure ID alignment framework to protect the data ID information in vertical federated logistic regression. Meanwhile, Nock *et al.* [17] give a comprehensive discussion of the impact of ID entity resolution in VFL. In addition, Liu *et al.* [18] point out the privacy concern of the ID alignment in asymmetrical VFL. Yang *et al.* introduce a simplified two-party vertical FL framework [19] by removing the third party coordinator. Different from the aforementioned work for training *parametric* models in VFL, Cheng *et al.* [20] first propose a secure XGBoost [21] decision tree system named SecureBoost in a setting of vertically partitioned data with the help of HE. Based on this article, Wu *et al.* introduce a novel approach called Pivot [22] to ensure that the intermediate information is not disclosed during training. Tian *et al.* [23] design a Federboost scheme to train a GBDT over both HFL and VFL and vertical Federboost can satisfy the security requirements without any encryption operations.

However, the current privacy preserving VFL systems are built under the assumption that all the data labels are stored only on one *guest* or *active* party, which is not realistic in many real-world applications. For instance, in real-life medical

Manuscript received 2 August 2021; revised 16 October 2021 and 21 December 2021; accepted 25 December 2021. Date of publication 28 December 2021; date of current version 22 September 2023. This work was supported in part by an Alexander von Humboldt Professorship endowed by the Federal Ministry of Education and Research, Germany, and in part by a European Unions Horizon 2020 Research and Innovation Programme under Grant 952697 (ASSURED) and Grant 101021727 (IRIS). This paper was recommended for publication by Associate Editor C. L. Philip Chen upon evaluation of the reviewers' comments. *(Corresponding author: Yaochu Jin.)*

Hangyu Zhu is with the Department of Computer Science, University of Surrey, GU27XH Guildford, U.K. (e-mail: hangyu.zhu@surrey.ac.uk).

Rui Wang and Kaitai Liang are with the Department of Intelligent Systems, Delft University of Technology, 2628XE Delft, The Netherlands (e-mail: R.Wang-8@tudelft.nl; Kaitai.Liang@tudelft.nl).

Yaochu Jin is with Nature Inspired Computing and Engineering Group, Faculty of Technology, Bielefeld University, D-33619 Bielefeld, Germany, and also with the Department of Computer Science, University of Surrey, GU27XH Guildford, U.K. (e-mail: yaochu.jin@surrey.ac.uk).

Digital Object Identifier 10.1109/TAI.2021.3139055

systems, different hospitals in the same region may have the same group of patients but provide different modalities of disease tests. Therefore, it is very likely that each hospital just owns parts of patients' diagnosis results.

As a result, we consider a more realistic setting of training XGBoost decision tree models in VFL in which each participating client holds parts of data labels that cannot be shared and exchanged with others during the training process. Compared to the standard vertical boosting tree system, constructing a secure vertical federated XGBoost system over distributed labels has the following challenges.

1) There is no single guest client for gradient and Hessian computation.
2) If each client is responsible for both feature splits and impurity calculations, the raw gradients can still be tracked even if HE is adopted in the learning system.
3) Leaf weight values used for label predictions are required to be sent to multiple clients, which increases the potential risk of label leakage. Any client may track and guess the real labels from other clients through the received leaf weights.

To tackle the aforementioned challenges, we propose a novel privacy-preserving vertical federated learning system over distributed labels (PIVODL). PIVODL allows multiple clients to jointly construct a XGBoost tree model without disclosing feature or label information of the training data, given that the data labels are distributed across each of the participating clients in VFL. The contributions of this article can be summarized as follows.

1) We are the first to consider training XGBoost decision trees in VFL, in which the data labels are distributed over multiple data owners. The potential risk of privacy leakage under this condition is discussed in detail.
2) A novel secure node split protocol is proposed by setting *source* clients and *split* clients for node split. By combining the source clients with the split clients during tree node split in VFL, we can effectively defend differential attacks and prevent intermediate gradients and Hessian values from being leaked.
3) The calculated leaf node weights need to be sent to other clients for label prediction updates. Therefore, an extra partial DP scheme is introduced by adding Gaussian noise to the leaf weights before sending them to the source clients.
4) Empirical experiments are performed to demonstrate that the proposed PIVODL system can effectively protect users' data privacy with a negligible model performance degradation.

II. Preliminaries

A. Vertical Federated Learning (VFL)

Different from HFL [2], where each client owns all features of the training set, VFL [24] mainly focuses on the scenario where features are distributed across different clients. We denote $\mathcal{X}$ as the feature space, $\mathcal{Y}$ as the label space and $\mathcal{I}$ as the data IDs, the standard VFL can be defined as follows.

Algorithm 1: VFL for Logistic Regression.

```
1:  Training data $\mathcal{X} = \{\mathcal{X}^1, \mathcal{X}^2, \ldots \mathcal{X}^n\}$ on $n$ clients
2:  Initialize the local model $\theta_0^c$, $c \in \{1, \ldots n\}$
4:  for each communication round $t = 1, 2, \ldots, T$ do
5:    for batch data $\mathcal{X}_b^c \in (\mathcal{X}_1^c, \mathcal{X}_2^c, \ldots, \mathcal{X}_B^c)$ do
6:      for each Client $c = 1, 2, \ldots, n$ in parallel do
7:        Compute $z_b^c = \mathcal{X}_b^c \theta_t^c$ and send it to guest client
8:      end for
9:      Compute $\hat{y}_b = a(\sum_{c=1}^n z_b^c)$ and $L(y_b, \hat{y}_b)$ on guest
        client
10:     Compute each $\frac{\partial L}{\partial z_b}$ on the guest client and send it
        to the corresponding host client
11:     for each Client $c = 1, \ldots, n$ in parallel do
12:       $\theta_t^c \leftarrow \theta_t^c - \eta \frac{\partial L}{\partial z_b^c} \frac{\partial z_b^c}{\partial \theta_t^c}$
13:     end for
14:   end for
15: end for
```

Definition 1: VFL: Given a training set with m data points distributed across n clients, each client c has parts of data features $\mathcal{X}^c = \{X_1^c, \ldots, X_m^c\}$ and sample IDs $\mathcal{I}^c = \{I_1, \ldots, I_m\}$, where $c \in \{1, \ldots, n\}$. Only one client $l \in \{1, \ldots, n\}$ contains all the labels $\mathcal{Y} = \{y_1, \ldots, y_m\}$. For any two different clients c, c', they satisfy

$$\mathcal{X}^c \neq \mathcal{X}^{c'}, \mathcal{Y}^c = \mathcal{Y}^{c'}, \mathcal{I}^c = \mathcal{I}^{c'}, c \neq c'. \tag{1}$$

It indicates each connected client c in VFL shares the same data sample IDs $\mathcal{I}^c$ with the same corresponding labels $\mathcal{Y}^c$, but different clients may hold $\mathcal{X}^c$ sampled from different data feature space. Note that $\mathcal{Y}^c$ is an imaginary data label that is not held by client c, if $c \neq l$. And there are two kinds of clients in the standard VFL: one is the *guest* or *active* client, the other is the *host* or *passive* client.

Definition 2: Guest Client: It holds both data features $\mathcal{X}$ and labels $\mathcal{Y}$ and is responsible for calculating the loss function, gradients, Hessians, and leaf values for the corresponding data samples.

Definition 3: Host Client: It only holds data features $\mathcal{X}$ and is responsible for aggregating the encrypted gradients and Hessians in one bucket.

In general, the guest client and host client can be seen as parameter server and node worker defined in distributed data parallelism [25]–[27]. A typical vertical federated training paradigm for logistic regression [28] is shown in Algorithm 1. Since the loss function $L(y_b, \hat{y}_b)$ is derived on the guest client, the data labels $\hat{y}_b$ will not be revealed to other host clients.

B. VFL With XGBoost

In this article, we consider training XGBoost for classification and regression tasks. XGBoost [21] is a widely used boosting tree model in tabular data training because of its better interpretation, easier parameters tuning and faster training process compared with deep learning [29], [30]. Suppose that a training set with m data entries consisting of the feature space $\mathcal{X} =$

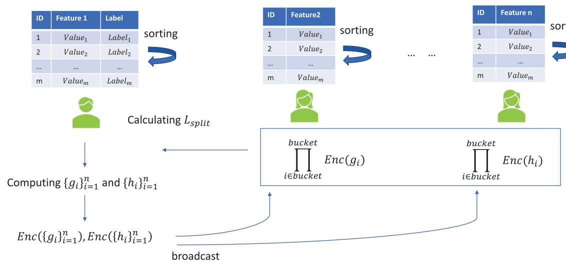

ID
Feature 1
Label
1
$Value_1$
$Label_1$
2
$Value_2$
$Label_2$
...
m
$Value_m$
$Label_m$
sorting
ID
Feature2
sorting
... ...
ID
Feature n
sorting
Calculating L_{split}
$\prod_{i\in bucket}^{bucket} Enc(g_i)$
$\prod_{i\in bucket}^{bucket} Enc(h_i)$
Computing $\{g_i\}_{i=1}^n$ and $\{h_i\}_{i=1}^n$
$Enc(\{g_i\}_{i=1}^n), Enc(\{h_i\}_{i=1}^n)$
broadcast

Fig. 1. Secureboost system.

$\{x_1, \ldots, x_m\}$ and label space $\mathcal{Y} = \{y_1, \ldots, y_m\}$. Gradients and Hessian can be calculated from (2) and (3) for each data entry, where $y_i^{(t-1)}$ denotes prediction of the previous booster tree for the ith data point

$$g_i = \frac{1}{1+e^{-y_i^{(t-1)}}} - y_i = \hat{y}_i - y_i \tag{2}$$

$$h_i = \frac{e^{-y_i^{(t-1)}}}{(1+e^{-y_i^{(t-1)}})^2}. \tag{3}$$

Before training starts, the threshold value for each feature split should be defined. For building trees, the XGBoost algorithm splits each node based on whether the current depth and the number of trees have reached the predefined maximum depth and maximum tree number. If neither of the aforementioned conditions is satisfied, a new split is selected from all possible splits based on the maximum impurity score L_{split} defined in (4), where λ and γ are regularization parameters

$$L_{split} = \frac{1}{2}\left[\frac{(\sum_{i\in I_L} g_i)^2}{\sum_{i\in I_L} h_i + \lambda} + \frac{(\sum_{i\in I_R} g_i)^2}{\sum_{i\in I_R} h_i + \lambda} - \frac{(\sum_{i\in I} g_i)^2}{\sum_{i\in I} h_i + \lambda}\right] - \gamma. \tag{4}$$

The current node is leaf if one of the aforementioned conditions is satisfied and the leaf value can be calculated according to

$$w = -\frac{\sum_{i\in I} g_i}{\sum_{i\in I} h_i + \lambda}. \tag{5}$$

Cheng *et al. [20] first propose a solution to privately train the XGBoost model in VFL. As a simple example shown in Fig. 1,* the guest client is responsible for calculating the corresponding gradients and Hessians for all data points and sends them to other host clients in ciphertexts under Paillier encryption. Each host client sorts features and makes splits according to the predefined feature thresholds (buckets). After that, each host client aggregates all the encrypted gradients and Hessians in one bucket and returns them back to the guest client for decryption and impurity calculations [see (4)]. The split corresponding to the maximum L_{split} is regarded as the best split for the current node. The aforementioned steps will be done recursively until the trees reach the maximum depth or the number of trees, or when L_{split} is smaller than the predefined boundary. Compared with [20], Tian *et al.* [23] propose a more efficient scheme in which the host clients sort their features first, and send the order of different features to the guest client with noise satisfying DP. Once the guest client receives all orders of the features, all the training steps can be processed on the guest client. During the training, the labels never leave from the guest client that only knows the order of different features. Therefore, the privacy of the training set is guaranteed.

C. Additively HE

Homomorphic encryption (HE) [31], [32] is widely used for secure outsourced computation. It enables multiple encrypted messages to compute directly in ciphertexts, whose results, when decrypted, are equal to the produced output of math operations on unencrypted messages.

Paillier [32] is a very popular additive homomorphic cryptosystem widely used in FL. And our proposed PIVODL applies Paillier to prevent label leakages during node splits. It works as follows.

1) *Key generation:* Randomly select two large prime numbers p and q s.t. $\gcd(pq, (p-1)(q-1)) = 1$. Let $n = pq$ and $\lambda = \text{lcm}(p-1, q-1)$. After that, randomly choose an integer $g \in \mathbb{Z}_{n^2}^*$ and compute $\mu = (L(g^\lambda \mod n^2))^{-1} \mod n$, where L is a function defined as $L(x) = \frac{x-1}{n}$. The public key pk and secret key sk are (n, g) and (λ, μ), respectively.
2) *Encryption:* To encrypt a message $m \in \mathbb{Z}_n^*$, choose a random number $r \in \mathbb{Z}_n^*$ as an ephemeral key, the ciphertext is calculated as $c = g^m \cdot r^n \mod n^2$.
3) *Decryption:* The plaintext message m can only be decrypted if the secret key (λ, μ) is available by computing $m = L(c^\lambda \mod n^2) \cdot \mu \mod n$.

The Paillier satisfies the additive homomorphic property: $\text{Enc}(m1) * \text{Enc}(m2) = \text{Enc}(m1 + m2) = g^{m_1} r_1^n \cdot g^{m_2} r_2^n \mod n^2$.

D. Differential Privacy (DP)

Differential privacy (DP) [33], [34] is a data privacy protection scheme. It can publish statistical information while keeping individual data private. If a substitution of an arbitrary single data entity does not cause statistically distinguishable changes, the algorithm used to run dataset satisfies DP. The definition of DP is presented as follows:

Definition 4: (ϵ-DP [33]). Given a real positive number ϵ and a randomized algorithm $\mathcal{A}$: $\mathcal{D}^n \to \mathcal{Y}$. Algorithm $\mathcal{A}$ provides ϵ-DP, if for all datasets D, $D' \in \mathcal{D}^n$ differs on only one entity, and all $\mathcal{S} \subseteq \mathcal{Y}$ satisfy

$$\Pr[\mathcal{A}(D) \in \mathcal{S}] \leq \exp(\epsilon) \cdot \Pr[\mathcal{A}(D') \in \mathcal{S}]. \tag{6}$$

To achieve ϵ-DP, some mechanisms are proposed to add designed noise to queries. In this article, we apply a relaxation of ϵ, δ-DP, called approximate DP [34]. The formal definition is as follows.

Definition 5: ((ϵ, δ)-DP). Given two real positive numbers (ϵ, δ) and a randomized algorithm $\mathcal{A}$: $\mathcal{D}^n \to \mathcal{Y}$. An algorithm

$\mathcal{A}$ provides (ϵ, δ)-DP if it satisfies

$$\Pr[\mathcal{A}(D) \in \mathcal{S}] \leq exp(\epsilon) \cdot \Pr[\mathcal{A}(D') \in \mathcal{S}] + \delta. \tag{7}$$

Gaussian mechanism [35]–[37] is usually applied in DP by adding Gaussian noise $\mathcal{N} \sim N(0, \Delta^2\sigma^2)$ to the output of the algorithm, where Δ is l_2 - norm sensitivity of D and $\sigma \geq \sqrt{2\ln(1.25/\delta)}$. According to [35], this noise can only achieve $(O(q\epsilon), q\delta)$-DP, where q is the sampling rate per lot. To make the noise small while satisfying DP, we follow the definition in [35] and set $\sigma \geq c\frac{q\sqrt{T\log(1/\delta)}}{\epsilon}$, c is a constant, and T refers to the number of steps, to achieve $(O(q\epsilon\sqrt{T}), \delta)$-DP.

III. Problem Formulation

A. System Model

All the participants use the private set intersection [38], [39] to align data IDs before the training starts. And the complete training set with m data entries consists of a feature space $\mathcal{X} = \{x_1, \ldots, x_m\}$, each containing d features, and a label space $\mathcal{Y} = \{y_1, \ldots, y_m\}$. Besides, n clients possessing at least one feature $\{X_j^c \mid j \in \{1, \ldots, d\}\}$ with all data points and part of labels $\{y_i^c \mid i \in \{1, \ldots, m\}\}$ choose to train a model collaboratively, where X_j^c and y_i^c represent the jth feature and the ith label owned by the cth client, respectively.

Since the assumption that all labels are held by only one guest client is not realistic, we consider a variant of VFL called VFL over distributed labels (VFL-DL), where labels are distributed on multiple clients. The formal definition of VFL-DL is as follows.

Definition 6: (VFL-DL). Given a training set with m data points, each participating client c consists of data features $\mathcal{X}^c = \{X_1^c, \ldots, X_m^c \mid c \in \{1, \ldots, n\}\}$ and parts of labels $\mathcal{Y}^c = \{y_i^c \mid i \in \{1, \ldots, m\}, c \in \{1, \ldots, n\}\}$. For any two clients $c, c' \in \{1, \ldots, n\}$

$$X^c \neq X^{c'}, y^c \neq y^{c'}, \mathcal{I}^c = \mathcal{I}^{c'} \forall c \neq c'. \tag{8}$$

The frequently used notations are summarized in Table I.

B. Threat Model

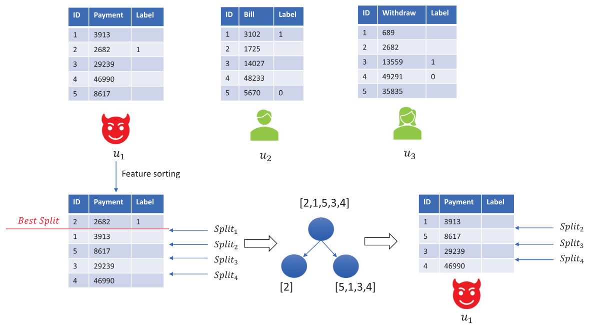
In this article, we mainly consider potential threats from participating clients. We assume that all participating clients during training are *honest-but-curious*, which means they strictly follow the designed algorithm but try to infer extra data information of other clients from the received information. The main goal of this work is to prevent the leakage of private data, and other attacks like data poisoning and backdoor attacks, which may deteriorate the model performance, are not considered here.

C. Privacy Concerns

Since the data labels are distributed across multiple clients in our framework, each client becomes "guest" client, which is responsible for both aggregating $\sum g_i$ and $\sum h_i$ and updating label predictions. This would further lead to potential node split and prediction update leakage during the execution of the vertical federated XGBoost protocol.

TABLE I
Notations Summary

Notation	Description
$\mathcal{X}$	feature space
X_j^c	the j-th data feature owned by the c-th client
x_i	i-th data point with d features
$\mathcal{Y}$	label space
y_i^c	the label of the i-th data point owned by the c-th client
$\mathcal{I}$	data index
g_i^c	the gradient of the i-th data point owned by the c-th client
$G_{j,v}^c$	the aggregated *left* gradient sum for feature j with threshold v owned by the c-th client
h_i^c	the Hessian of the i-th data point owned by the c-th client
$H_{j,v}^c$	the aggregated *left* Hessian sum for feature j with threshold v owned by the c-th client
m	number of data points
n	number of clients
d	number of features
b	number of buckets (feature thresholds)
ϵ, δ	parameters of differential privacy
Δ	sensitivity
L_{split}	similarity or impurity score
w	leaf value
pk	public key
sk	secret key



ID	Payment	Label
1	3913	
2	2682	1
3	29239	
4	46990	
5	8617	

ID	Bill	Label
1	3102	1
2	1725	
3	14027	
4	48233	
5	5670	0

ID	Withdraw	Label
1	689	
2	2682	
3	13559	1
4	49291	0
5	35835	

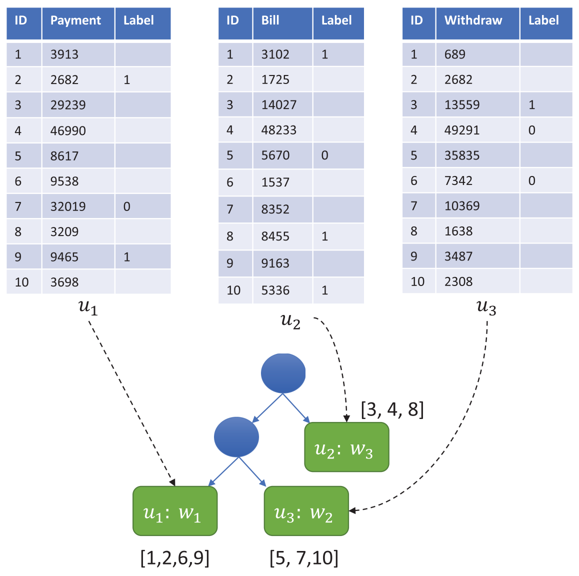
ID	Payment	Label
2	2682	1
1	3913	
5	8617	
3	29239	
4	46990	

ID	Payment	Label
1	3913	
5	8617	
3	29239	
4	46990	

Fig. 2. Simple example of split leakage.

1) Split Leakage: In order to find the best split with the largest L_{split} value for the current tree node, each client needs to ask other clients for the corresponding summation $G_{j,v}^c$ and $H_{j,v}^c$ of all possible splits. This may result in a risk of privacy leakage from two sources: one is the summation differences among all feature splits of the same node, and the other is the summation differences between parent node and child node. Moreover, the privacy issue still exists even when the summation of the gradients and Hessians are encrypted. This is because the aggregated summations should be decrypted for impurity calculation, and therefore, the raw gradients and Hessians can still be deduced via summation differences of multiple feature splits.

A simple example is shown in Fig. 2, where client u_1 wants to compute L_{split} values from Split_1 to Split_4 with the help of u_2 and u_3. The label information about u_2 or u_3 can be easily deduced by *differential attacks*, even if the message is encrypted. For instance, u_1 requires aggregated summation $G_{1,\text{split}_2} = g_2 + g_1$ and $G_{1,\text{split}_3} = g_2 + g_1 + g_5$ to calculate L_{split_2} and L_{split_3}, respectively. For privacy concerns, both g_1 and g_5 are encrypted before aggregation and u_1 only achieves aggregated summations and has no knowledge of these two gradient values. However,



ID	Payment	Label
1	3913	
2	2682	1
3	29239	
4	46990	
5	8617	
6	9538	
7	32019	0
8	3209	
9	9465	1
10	3698	

ID	Bill	Label
1	3102	1
2	1725	
3	14027	
4	48233	
5	5670	0
6	1537	
7	8352	
8	8455	1
9	9163	
10	5336	1

ID	Withdraw	Label
1	689	
2	2682	
3	13559	1
4	49291	0
5	35835	
6	7342	0
7	10369	
8	1638	
9	3487	
10	2308	

Fig. 3. Label prediction update in VFL with the true data labels distributed across multiple clients.

g_5 from u_2 can still be derived based on the *difference* (subtract here) between G_{1,split_3} and G_{1,split_2}. Similarly, g_1 can be easily obtained by $G_{1,\text{split}_2} - g_2$.

2) Prediction Update Leakage: Except for split leakages, private label information may also be disclosed in updating the predictions. As introduced in Section II-B, XGBoost is an ensemble boosting algorithm that requires to sequentially construct multiple decision trees for regression or classification tasks. And the label prediction $\widehat{y}$ for each data sample should be updated based on the leaf value of the current tree, whenever a new decision tree is being built. Unlike in the conventional VFL where the guest client performs all leaf weights calculations and label prediction updates; in PIVODL, labels are stored on multiple clients based on their split features of the parent nodes. As a result, it is very likely that some clients will receive predicted labels of other clients.

An example is shown in Fig. 3, where three leaf nodes are stored on three clients u_1, u_2, and u_3, respectively. It is clear to see that u_1 knows the leaf weight of data 1, 2, 6, and 9, thus, the label prediction of the current decision tree for data 1 on u_2 and data 6 on u_3 are leaked to u_1. There is no doubt that the predicted values will be closer to the true labels with a high probability over the training process. Therefore, a more accurate guess can be made for specific data samples along with the model convergence, if the clients can get some of their leaf weights.

IV. Proposed System

Our proposed PIVODL system is introduced in this section. PIVODL adopts a novel secure node split protocol to construct an ensemble of boosting trees without revealing intermediate gradients and Hessian information. And it also applies a partial DP strategy to protect the predicted labels with an acceptable drop on the model performance. Finally, attack inferences are also imposed to our system to verify that no client inside the system is able to receive or deduce label information from other clients.

A. Secure Node Split Protocol

As discussed in Section III, private label information may be disclosed during the calculation of L_{split} and the branch splits. This is because the client used for *node split* also *acquire* all the data ID information for all possible feature splits of the current tree node. Therefore, as long as the client knows the intermediate aggregation results of $G^c_{j,v}$ and $H^c_{j,v}$ from other clients, g^c_i and h^c_i can be easily deduced through the differences between two adjacent summation results, regardless of whether any cryptographic operations are used.

In order to fix the aforementioned issue, feature splits and summation aggregations during node split should be performed on two *separate* clients. One is called the *source client* and the other is called the *split client*. In the following, we define the roles of the source and split clients.

Definition 7: The source client is the split owner that contains all data IDs of the local features. However, it does not have any information of the corresponding summation values $G_{j,v}$ and $H_{j,v}$, and thus, is not able to calculate the impurity scores in (4) and leaf weights in (5).

Definition 8: The split client, on the other hand, is used to compute both the impurity scores in (4) and leaf weights in (5), however, it has no idea of their corresponding data IDs on other clients.

By making use of the source and split clients, the tree nodes can be split without leaking the data privacy in the following steps.

1) Each client locally generates its unique *Paillier* key pairs and exchanges the public key with each other. Meanwhile, the data IDs of the samples with labels I^c are also exchanged with each other for privacy concern, which will be discussed in the next section.
2) Each client c computes locally $G^c = \sum_i g^c_i$ and $H^c = \sum_i h^c_i$ for the current tree node according to the available data labels y^c_i. Two random clients are selected to be the *aggregation client* and *encryption client*, respectively. Each client uses the public key of the *encryption client* to encrypt G^c and H^c and sends the encrypted $\langle G^c\rangle$ and $\langle H^c\rangle$ to the *aggregation client* for summation: $\langle G\rangle = \prod_c\langle G^c\rangle = \langle\sum_c G^c\rangle$, $\langle H\rangle = \prod_c\langle H^c\rangle = \langle\sum_c H^c\rangle$. After that, the *aggregation client* sends $\langle G\rangle$ and $\langle H\rangle$ to the *encryption client* for decryption: $G = \text{Dec}\langle G\rangle$ and $H = \text{Dec}\langle H\rangle$. Finally, the *encryption client* can broadcast the decrypted G and H for calculating the information gain of the current node by $\frac{1}{2}\frac{G^2}{H+\lambda}$.
3) All client are regarded as the source clients and sort their data samples according to the split buckets (feature thresholds) of all the local features. In addition, each source client randomly selects one client from other clients to be its *unique* split client. And then, each source client c computes local left branch sum $G^c_{j,v} = \sum_{i\in\{i|x_{i,j}<v\}} g^c_i$ and $H^c_{j,v} = \sum_{i\in\{i|x_{i,j}<v\}} h^c_i$ based on its

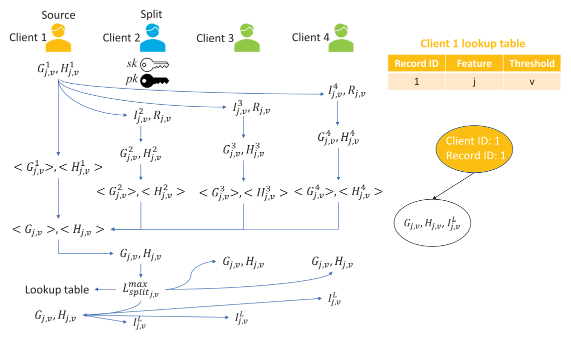


Fig. 4. Secure node split for the left branch, where we assume the feature j and threshold v will give the best split with the largest impurity score L_{split}.

owned labels $y^c_{i,j}$, and sends the *intersected* missing data IDs $I^{c'}_{j,v} = I^c_{j,v} \bigcap I^{c'}, c' \in [1, n], c' \neq c$ together with its split client ID and temporary record number $R_{j,v}$ to other clients.

4) Each client c calculates $G^c_{j,v} = \sum_{i \in \{i | i \in I^c_{j,v}\}} g^c_i$ and $H^c_{j,v} = \sum_{i \in \{i | i \in I^c_{j,v}\}} h^c_i$, and uses the public key of its corresponding split client to encrypt the summation values, after receiving intersected data IDs and corresponding split client ID from source clients. And then, each client returns the encrypted $\langle G^c_{j,v} \rangle$ and $\langle H^c_{j,v} \rangle$ back to the corresponding source client.
5) After the source client receives all $\langle G^c_{j,v} \rangle$ and $\langle H^c_{j,v} \rangle$ from other clients, it will aggregate (sum) all received encrypted summation with the local gradient and Hessian summation to get the final encrypted $\langle G_{j,v} \rangle = \prod_c \langle G^c_{j,v} \rangle = \langle \sum_c G^c_{j,v} \rangle$ and $\langle H_{j,v} \rangle = \prod_c \langle H^c_{j,v} \rangle = \langle \sum_c H^c_{j,v} \rangle$ for all bucket splits.
6) Each source client sends $\langle G_{j,v} \rangle$ and $\langle H_{j,v} \rangle$ to its corresponding split client for decryption and the split clients can compute the impurity scores by $L_{\text{split}_{j,v}} = \frac{1}{2}[\frac{G_{j,v^2}}{H_{j,v}+\lambda} + \frac{G - G_{j,v^2}}{H - H_{j,v}+\lambda} - \frac{G^2}{H+\lambda}] - \gamma$. All the selected split clients get the largest local impurity score at first, and then, broadcast these values for further comparison to achieve the largest global impurity score $L_{\text{split}^{\max}_{j,v}}$. The split client that owns $L_{\text{split}^{\max}_{j,v}}$ would send the temporary record number $R_{j,v}$ back to the source client. The source can get the best split feature j and threshold v according to received $R_{j,v}$ and privately build a lookup table [20] to record j and v with a unique *record ID*. And then, the tree node can be constructed by the source client ID and the record ID as shown in Fig. 4.
7) If the current tree node is not the leaf node, either the left or right child node will continue splitting. If the left node becomes the current node for splitting, the split client sets $G = G_{j,v}$ and $H = H_{j,v}$, and sends them to all other clients. Otherwise, the split client sets $G = G - G_{j,v}$ and $H = H - H_{j,v}$, and sends them to all other clients. In addition, the source client sends the left split node data IDs $I^L_{j,v}$ or the right split node data IDs $I^R_{j,v}$ to other clients.
Then, each client computes the current information gain $\frac{1}{2}\frac{G^2}{H+\lambda}$. Go to step (3) and repeat the process.

It is clear to see that the proposed secure node split protocol encrypts the intermediate summation values $G_{j,v}$ and $H_{j,v}$ for the privacy concern and is able to effectively defend differential attacks. The source client contains all the split node IDs and their corresponding split features and thresholds. However, as shown in Fig. 4, the source client 1 can only get the encrypted sum $\langle G_{j,v} \rangle$ and $\langle H_{j,v} \rangle$ for feature j and threshold v. Thus, differential attack does not work in this scenarios since only the split client 2 has the secret key to get the plaintext of G and H. On the other hand, the split client 2 only knows local intersected data IDs $I^2_{j,v}$ and the final summation values G and H, but it has no idea of the corresponding data IDs. In addition, the split client 2 only receives the temporary record number $R_{j,v}$ (this is just an arbitrary number defined by the source client, "temporary" means this number would be reset for the next tree node split) of the split and has no knowledge of the corresponding real split feature and threshold on the source client.

After all the split clients decrypt $G_{j,v}$ and $H_{j,v}$ for all available features j and thresholds v, they can calculate L_{split} according to (4). In order to reduce both potential privacy leakage and communication costs, each split client compares and gets the largest L_{split} value locally at first. The advantage of doing this is that each split client just needs to broadcast only one impurity score L_{split} for comparison. If the current tree node continues splitting to the left branch as shown in Fig. 4, the split client needs to transmit the decrypted $G_{j,v}$ and $H_{j,v}$ to all other clients as G and H of the next tree node. In addition, the source client needs to send the data IDs $I^L_{j,v}$ of the left split to other clients. Therefore, each client can deduce the gradient and Hessian values from other clients through the received $G_{j,v}$, $H_{j,v}$, and $I^L_{j,v}$ as discussed in Section III-C1.

In order to alleviate this privacy leakage issue, we adopt a simple data instance threshold strategy. For example, if the number of received intersected data IDs is less than the predefined instance threshold, this client will reject to compute the local $G^c_{j,v}$ and this split will be removed. It should be mentioned that continue splitting is robust to differential attacks since only one $G_{j,v}$ and $H_{j,v}$ are broadcast to each client. The overall secure node split protocol is also shown in Algorithm 2.

B. Construction of Private Tree Nodes

After the split client c' with the largest impurity score is determined, c' will send the temporary record number $R^{c,\text{best}}_{j,v}$ to the corresponding source client c. And then, the source client c can add the feature j and threshold v with a unique record ID into the local lookup table [20], as shown in Fig. 4. After that, c broadcasts the unique record ID to any other client that can annotate the split of current tree node with client c's ID and the record ID.

Meanwhile, the source client c can check and tell the split client c' whether the split child nodes are leaf or not based on the condition shown in lines 5 and 15 of Algorithm 3, respectively. If the child node is not leaf, the source client c broadcasts the split data IDs and the split client c' broadcasts the corresponding

Algorithm 2: Secure node split protocol.

```
1:  Input: $I$, data IDs of the current tree node
2:  Input: $G$, gradient sum of the current tree node
3:  Input: $H$, Hessian sum of the current tree node
4:  Input: $T$, instance threshold of the current tree node
6:  Split client sets $C' = \emptyset$
7:  for each source client $c = 1, 2, \ldots, n$ do
8:    Randomly select a split client $c'$, $c' \in [1, n], c' \neq c$
9:    $L^{c'}_{split} \leftarrow 0, C' \leftarrow C' \cup c'$
10: end for
12: for each source client $c = 1, 2, \ldots, n$ do
13:   $R^c_{j,v} \leftarrow 0$
14:   for each feature $j = 1, 2, \ldots, d^c$ do
15:     for each threshold $v = 1, 2, \ldots, b_j$ do
16:       $pk \leftarrow pk$ of client $c'$
17:       $G^c_{j,v} \leftarrow \sum_{i \in \{i | x_{i,j} < v\}} g^c_i$,
          $\langle G^c_{j,v} \rangle \leftarrow \text{Enc}_{pk}(G^c_{j,v})$
18:       $H^c_{j,v} \leftarrow \sum_{i \in \{i | x_{i,j} < v\}} h^c_i$,
          $\langle H^c_{j,v} \rangle \leftarrow \text{Enc}_{pk}(H^c_{j,v})$
19:       $R^c_{j,v} \leftarrow R^c_{j,v} + 1$
20:       for each client $c'' = 1, 2, \ldots, n, c'' \neq c$ do
21:         $I^{c''}_{j,v} \leftarrow I^c_{j,v} \bigcap I^{c''}$
22:         Send $I^{c''}_{j,v}$, $R^c_{j,v}$ to client $c''$:
23:         $pk \leftarrow pk$ of client $c'$
24:         if $|I^{c''}_{j,v}| \geq T$ then
25: $G^{c''}_{j,v} \leftarrow \sum_{i \in I^{c''}_{j,v}} g^{c''}_i$, $\langle G^{c''}_{j,v} \rangle \leftarrow \text{Enc}_{pk}(G^{c''}_{j,v})$
26: $H^{c''}_{j,v} \leftarrow \sum_{i \in I^{c''}_{j,v}} h^{c''}_i$, $\langle H^{c''}_{j,v} \rangle \leftarrow \text{Enc}_{pk}(H^{c''}_{j,v})$
27: Return $\langle G^{c''}_{j,v} \rangle$ and $\langle H^{c''}_{j,v} \rangle$ to source client $c$
28:         end if
29:       end for
30:       if Source client $c$ receives $n-1$ $\langle G^{c''}_{j,v} \rangle$ and
          $\langle H^{c''}_{j,v} \rangle$ then
31:         $\langle G_{j,v} \rangle = \prod_c \langle G^c_{j,v} \rangle = \langle \sum_c G^c_{j,v} \rangle$
32:         $\langle H_{j,v} \rangle = \prod_c \langle H^c_{j,v} \rangle = \langle \sum_c H^c_{j,v} \rangle$
33:         Send $\langle G_{j,v} \rangle$ and $\langle H_{j,v} \rangle$ to split client $c'$:
34:         $G_{j,v} \leftarrow \text{Dec}\langle G_{j,v} \rangle$
35:         $H_{j,v} \leftarrow \text{Dec}\langle H_{j,v} \rangle$
36:         $L_{\text{split}_{j,v}} = \frac{1}{2}[\frac{G_{j,v}{}^2}{H_{j,v}+\lambda} + \frac{G-G_{j,v}{}^2}{H-H_{j,v}+\lambda} - \frac{G^2}{H+\lambda}] - \gamma$
37:         if $L_{\text{split}_{j,v}} > L^{c'}_{\text{split}}$ then
38: $L^{c'}_{split} \leftarrow L_{split_{j,v}}$, $R^{c,best}_{j,v} \leftarrow R^c_{j,v}$
39:         end if
40:       end if
41:     end for
42:   end for
43: end for
45: Output: the split client $c'$ with the largest impurity
    score $L^{c'}_{split}$
```

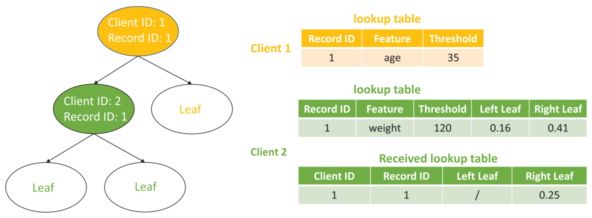
summation of gradients and Hessian for continuing splitting the child node. If the child node is determined to be a leaf node, the split client c' computes the local leaf weight and stores it in a *received* lookup table with source client c ID and record ID.

As a result, no client except the source client knows the feature j and threshold v of the tree node split, since the lookup table

Algorithm 3: Construction of private tree nodes, where c' is the split client with the largest impurity score, c is the source client of c', $R^{c,\text{best}}_{j,v}$ is the temporary record number of the best split on c, and T_{sample} is the minimum data samples for each decision node.

```
1:  $c'$ sends $R^{c,\text{best}}_{j,v}$ and split branch to $c$
2:  $c$ adds feature $j$ and threshold $v$ with a unique record
    ID into the local lookup table
3:  $c$ broadcasts the record ID and each client can annotate
    the split of current tree node with client $c$ ID and
    record ID
4:  if Split branch is left then
5:    if Reach the maximum depth or $|I_{j,v}| < T_{\text{sample}}$ then
6:      $c$ tells $c'$ left split node is the leaf node and stops
        splitting
7:      $w^L_{j,v} = -\frac{G_{j,v}}{H_{j,v}+\lambda}$
8:      $c'$ records left leaf value $w^L_{j,v}$ into received lookup
        table with record ID and client $c$ ID
9:    else
10:     Continue splitting left child node:
11:     $c$ broadcasts $I_{j,v}$ to all other clients
12:     $c'$ broadcasts $G_{j,v}$ and $H_{j,v}$ to all other clients
13:   end if
14: else if Split branch is right then
15:   if Reach the maximum depth or $|I - I_{j,v}| < T_{\text{sample}}$
      then
16:     $c$ tells $c'$ right split node is the leaf node and stops
        splitting
17:     $w^R_{j,v} = -\frac{G-G_{j,v}}{H-H_{j,v}+\lambda}$
18:     $c'$ records right leaf value $w^R_{j,v}$ into received
        lookup table with record ID and client $c$ ID
19:   else
20:     Continue splitting right child node:
21:     $c$ broadcasts $I - I_{j,v}$ to all other clients
22:     $c'$ broadcasts $G - G_{j,v}$ and $H - H_{j,v}$ to all other
        clients
23:   end if
24: end if
```

is stored locally without sharing it with any other client. And when it turns to new instance prediction, the source client needs to combine with the split client to get the leaf value. For instance, as shown in Fig. 5, client 1 is the source client of the root client and client 2 is its corresponding split client. If a new data sample with age 30 and weight 150 comes for prediction, it will be sent to client 1 for judgement. And then, client 1 uses its own lookup table to determine that this instance should go to the left branch (30 < 35). After that, these data are sent to client 2 to get its label prediction 0.41 (150 > 120, go to right leaf). Also considering potential label leakage during data prediction for ensemble decision trees, we adopt a secure aggregation scheme proposed in [13] for secure multiparty computation. Therefore, the final prediction for the data instance can be derived without revealing any leaf values.



lookup table (Client 1)

Record ID	Feature	Threshold
1	age	35

lookup table (Client 2)

Record ID	Feature	Threshold	Left Leaf	Right Leaf
1	weight	120	0.16	0.41

Received lookup table

Client ID	Record ID	Left Leaf	Right Leaf
1	1	/	0.25

Fig. 5. Each split node is recorded with the source client ID and a unique record ID. Both the lookup table and received lookup table are stored locally without sharing with any other client. The received lookup table needs to contain the source client ID and the corresponding record ID.

C. Partial DP

Each split client needs to send the leaf weights w to other clients for prediction updates before building the next boosting tree. There is no doubt that as the training proceeds, the *accumulated* label prediction $\widehat{y_i}$ will be closer to the true data label y_i, and since the source client knows all the split data IDs, as long as it receives the leaf weights, "parts of"[1] the true labels of the corresponding data samples may have the risk of being revealed.

In order to deal with this prediction update leakage issue, we propose a partial DP mechanism, where the leaf value is perturbed with Gaussian noise $\mathcal{N}$ before being sent to the source client. Since the calculation of the leaf value w is based on the summation of g and h (the number of g and h is not fixed, thus, its range is unpredictable), the sensitivity to noise cannot be achieved directly from the definition. In order to satisfy (ε, δ)-DP (see Section II-D), we use a clipping method here to let the sensitivity equals to the clipping boundary. As a result, our proposed algorithm naturally satisfies (ε, δ)-DP by clipping the leaf value before adding the Gaussian noise. The sensitivity Δ_w is calculated according to (9), where C refers to the predefined clip value and $\{G, H\}, \{G', H'\}$ are two sets, differing in one pair of g and h.

$$\begin{aligned}\Delta_w &= \max_{\{G,H\},\{G',H'\}} \|w^{\{G,H\}} - w^{\{G',H'\}}\| \\ &= 2\,C\ \forall \|w^{\{G,H\}}\|, \|w^{\{G',H'\}}\| \leq C.\end{aligned} \tag{9}$$

By *partial* DP, we mean that DP is applied on w sent to the source client only, since any other client just knows its intersected local data IDs. The advantage of using this strategy is that the correct distributed label predictions can be achieved to the greatest extent, while preventing the source client from guessing the true labels. The details of the partial DP mechanism are presented in Algorithm 4.

D. Security Analysis

Our proposed PIVODL framework avoids revealing both data features and labels on each participating client, and in this section, we will make a detailed security analysis of the leakage sources and protection strategies.

[1]Here, we call it "parts of," because XGboost is an ensemble tree algorithm. A client can get the real label predictions only if the leaf weights of all boosting trees of data samples are revealed to the client.

Algorithm 4 Partial DP. c' is the split client and c is the corresponding source client

```
1:  Input: $w_{j,v}$, computed leaf weight on split client $c'$
3:  for each $c'' = 1, 2, \ldots, n, c'' \neq c'$ do
4:    if $c'' = c$ then
5:      $\hat{w_{j,v}} \leftarrow w_{j,v}/\max(1, \frac{\|w_{j,v}\|}{C})$
6:      $\hat{w_{j,v}} \leftarrow \hat{w_{j,v}} + \mathcal{N} \sim N(0, 4\,C^2\sigma^2)$
7:      Send perturbed $\hat{w_{j,v}}$ to $c$
8:    else
9:      Send $w_{j,v}$ to $c''$
10:   end if
11: end for
```

During the training process, each client sorts the data instances based on the features and their corresponding bucket thresholds, which are stored privately on the local devices. In order to find the maximum impurity score L_{split}, the source clients require the corresponding g and h from other clients, since labels are distributed on multiple clients. The true data labels can be deduced from the computed gradients g.

Theorem 1: Given a gradient g, the true label of corresponding data point could be inferred.

Proof: According to (2), adversarial attackers are able to infer the true labels of the ith data sample by $y_i = \hat{y}_i - g_i$ if g_i is known. ■

To tackle this privacy issue, Paillier encryption is adopted in this work to encrypt all $\sum g$ and $\sum h$ for each bucket before sending them for summation aggregation. However, it is possible that the specific gradient information g_i can still be inferred by a differential attack, if the clients have data ID information of its adjacent bucket splits.

Theorem 2: There is still a high risk that the gradient g_i of the ith data sample may be deduced with a differential attack, even if the intermediate $\sum g$ and $\sum h$ are encrypted.

Proof: Assume L_n and $L_{n'}$ are two adjacent data sample sets for two possible splits that only differs in one data sample i (or a few data samples). Although the client only knows the aggregated summation $\sum_{i \in L_n} g_i$ and $\sum_{i \in L_{n'}} g_i$, g_i can be easily derived with a differential attack as $g_i = \sum_{i \in L_n} g_i - \sum_{i \in L_{n'}} g_i$. ■

Therefore, it is critical to ensure that the source client and split client perform the bucket split and aggregation separately. And then, the source client only knows the data IDs of each feature split but has no knowledge of their corresponding summation values, the split client holds the summation values but does not know their data IDs.

Before building the next boosting tree, the predictions of all data samples need to be updated based on the corresponding leaf weights. It will not be surprising that the predicted labels are the same as the true labels as the training proceeds. Therefore, attackers can guess the correct labels with a high confidence level through the received leaf weights.

Theorem 3: Given multiple boosting trees' leaf values, it is possible to deduce the true labels of the corresponding data points.

TABLE II
EXPERIMENTAL SETTINGS OF THE PIVODL SYSTEM

Hyperparameters	Range	Default
Number of clients	[2,4,6,8,10]	4
Maximum depth	[2,3,4,5,6]	3
Number of trees	[2,3,4,5,6]	5
Learning rate	/	0.3
Regularization	/	1
Number of buckets	/	32
Encrypt key size	/	512
ϵ	[2,4,6,8,10]	8
Sensitivity clip	/	2
Sample threshold	/	10

Proof: The label predictions $\hat{y}$ in a classification problem can be calculated with equation $\hat{y} = \sigma(\hat{y_0} + \eta \cdot W_1 + \cdots + \eta \cdot W_t)$, where η refers to the learning rate, W_t is the leaf weight of tth decision tree, and σ is the activation function. Therefore, once the source client knows parts of or even all the leaf values W, the true labels have a high risk of being inferred. ■

In order to address this issue, the split client applies the aforementioned partial DP mechanism before sending the leaf values to the source client. In this way, we can reduce the risk for the source client to correctly guess the true labels from other clients.

V. EXPERIMENTAL RESULTS

In this section, we first provide the experimental settings, and then, present the experimental results, followed by a discussion of the learning performance. After that, we evaluate the training time when implementing our privacy-preservation method. At the end, the communication cost and label prediction inference of PIVODL would be described.

A. Experimental Settings

Three common public dataset are used in the experimental studies, where the first two are classification tasks and the third is a regression task.

1) *Credit card* [40]: It is a credit scoring dataset that aims to predict if a person will make payment on time. It contains in total 30 000 data samples with 23 features.
2) *Bank marketing* [41]: The data are related to direct marketing campaigns of a Portuguese banking institution. The prediction goal is to evaluate whether the client will subscribe a term deposit. It consists of 45 211 instances and 17 features.
3) *Appliances energy prediction* [42]: It is a regression dataset of energy consumption in a low energy building, which has 19 735 data instances and 29 attributes.

We partition each dataset into training data and test data. The training and test dataset occupies 80% and 20% of the entire data samples, respectively. Other experimental settings like the number of clients and the number of the maximum depth of the boosting tree are presented in Table II. Besides, each experiment is repeated for five times independently and the results of the mean values together with their standard deviations are plotted in the corresponding figures.

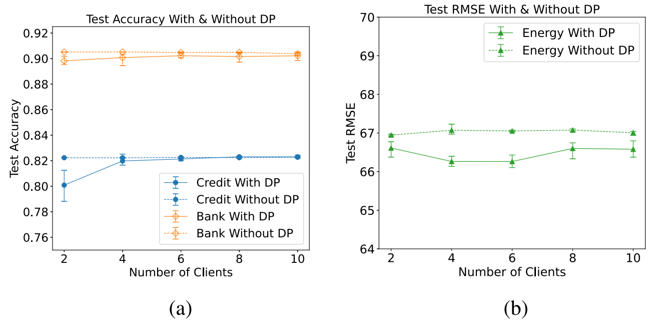


Fig. 6. Test accuracy and RMSE with and without applying DP over different numbers of participating clients, where (a) is the test accuracy on Credit card and Bank marketing datasets, and (b) is the test RMSE on the regression dataset. (a) Credit card and Bank marketing. (b) Appliances energy prediction.

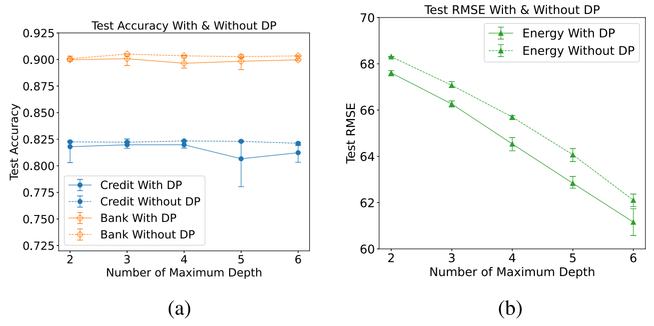


Fig. 7. Test accuracy and RMSE with and without applying DP over different maximum depths of a boosting tree, where (a) is the test accuracy on the Credit card and Bank marketing datasets, and (b) is the test RMSE on the regression dataset. (a) Credit card and Bank marketing. (b) Appliances energy prediction.

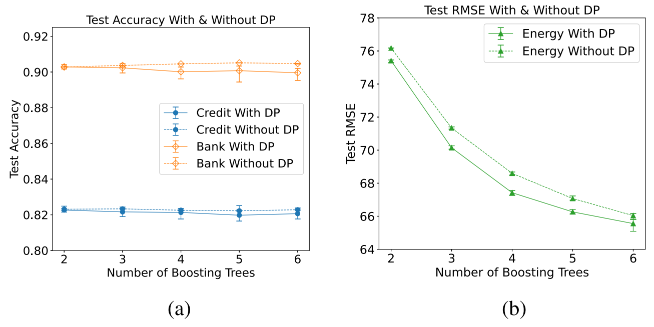
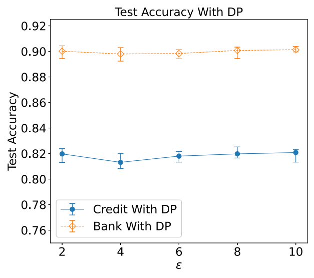


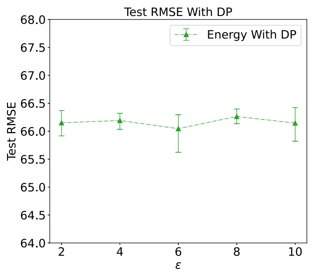
Fig. 8. Test accuracy and RMSE with and without applying DP over different numbers of boosting trees, where (a) is the test accuracy on the Credit card and Bank marketing datasets, and (b) is the test RMSE of the Appliances energy prediction dataset. (a) Credit card and Bank marketing. (b) Appliances energy prediction.

B. Sensitivity Analysis

Here, we empirically analyze the change of the learning performances as the number of participating clients, the maximum depth of the tree structure, the number of trees, and ϵ in the partial DP change. The results in terms of the mean, the best, and the worst performance over five independent runs are plotted in Figs. 6–9, respectively.

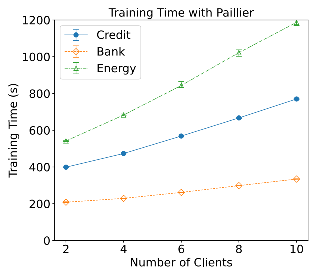


(a)

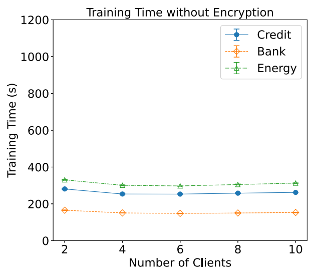


(b)

Fig. 9. Test performance over different ϵ values, where (a) is the test accuracy on the Credit card and Bank marketing datasets, and (b) is the test RMSE on Appliances energy prediction dataset. (a) Test accuracy with different ϵ. (b) Test RMSE with different ϵ.



(a)



(b)

Fig. 10. Total training time over different numbers of participating clients, where (a) is the training time with Paillier encryption and (b) is the training time without encryption. (a) Training time with Paillier encryption. (b) Training time without encryption.

From the results, we can see that varying the number of clients does not affect the training performance very much, especially in the experiments without applying the partial DP. As shown in Fig. 6, it is clear to see that the test performances on these three datasets are relatively insensitive to different numbers of clients. These results make sense since nonparametric models trained in the VFL setting have nearly the same performance as those trained in the standard centralized learning [43]. By contrast, however, the average test accuracy on the Credit dataset drops from 0.82 to 0.8 when the number of clients is reduced to 2. And it is surprising to see that the root-mean-squared error (RMSE) on the test data of the Energy dataset with DP is slightly higher than those without DP over different number of clients. The reason for this is that the model may have overfit the training data without applying DP. For instance, when the number of clients is 4 (the default value), the training RMSE with DP is about 70.9, which is higher than that without DP (66.6). By contrast, the test RMSE with DP is approximately 0.8% lower than that without DP. Overall, the performances with DP and without DP are almost the same and the test performance degradation resulting from the DP is negligible. Besides, the performance of the proposed PIVODL algorithm is rather stable in different runs.

Fig. 7 shows the performance when varying the number of maximum depth of the boosting tree. For classification tasks, the accuracy is about 82% and 90% for Credit card and Bank marketing datasets, respectively. And test accuracy of these two datasets remains almost constant as the number of maximum depth increases, except that the results on the Credit dataset with DP have slight fluctuations when the number of maximum depth is 5. Moreover, it is easy to find that the performances with DP and without DP are almost the same, which shows that the impact of the partial DP on the model performance is negligible. On the regression task, the average RMSE decreases as the maximum depth increases, since more complex tree structures may have better model performance. Similar to the previous cases, the test performance with DP is better than that without DP because of overfitting.

The performance change with the number of boosting trees is shown in Fig. 8. The results imply that only slight performance changes are observed when the number of boosting trees changes. On the Appliances energy prediction dataset, the RMSE slightly decreases with the increase in the number of boosting trees and the RMSE values with and without DP are nearly the same.

The test results on the three datasets over different ϵ values are shown in Fig. 9. And it is surprising to see that the test performances on all three datasets have no clear changes with the decrease of the ϵ value. This is because the partial DP is applied only on part of predicted labels in the source client, which makes it very unlikely to influence the node split at the next level of the node split. In addition, the split clients use denoised leaf values for prediction during the test, which further reduce the performance biases caused by the proposed partial DP method.

C. Evaluation on Training Time

In this section, we empirically analyze the training time affected by the number of participating clients, the number of maximum depth, and the number of boosting trees. All the experiments are run on Intel Core i7-8700 CPU and the Paillier encryption system is implemented with the package in [44]. Just like what we did in the previous set of experiments, the mean values together with the best and worst performances out of five independent runs are included in the results of the ablation studies.

Fig. 10 shows the training time when varying the number of clients with and without encryption. In general, it consumes the least training time on the Bank marketing dataset since it has the smallest amount of data. When no parameter encryption is adopted, the training time of the three datasets almost keeps constant over different numbers of clients, which is consistent with the results reported in [20] and [23]. The reason is that the data samples are partitioned across data features in VFL and setting different numbers of clients does not change the entire data entries. On the contrary, the training time with Paillier encryption increases linearly with the increase in the client numbers, since the amount of the encryption times is proportional to the number of clients in our proposed PIVODL algorithm.

The training times over different maximum depths of a boosting tree are shown in Fig. 11, and similarly, the cases with and

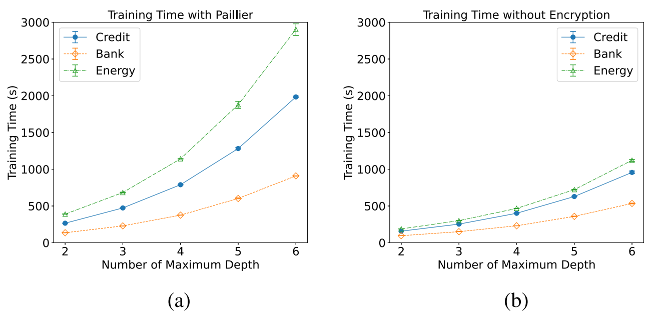


Fig. 11. Total training time over different maximum depths of a boosting tree, where (a) is the training time with Paillier encryption and (b) is the training time without encryption. (a) Training time with Paillier encryption. (b) Training time without encryption.

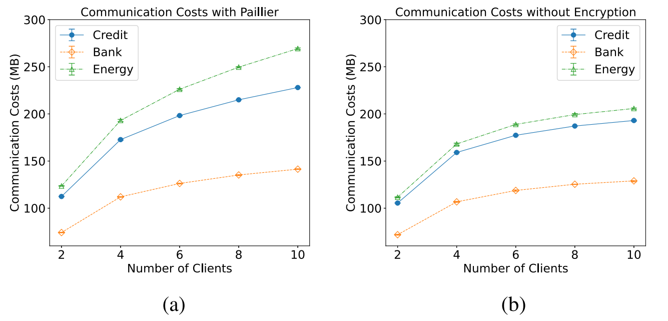


Fig. 13. Total communication costs over different number of clients (a) with Paillier encryption and (b) without encryption. (a) Communication costs with Paillier encryption. (b) Communication costs without encryption.

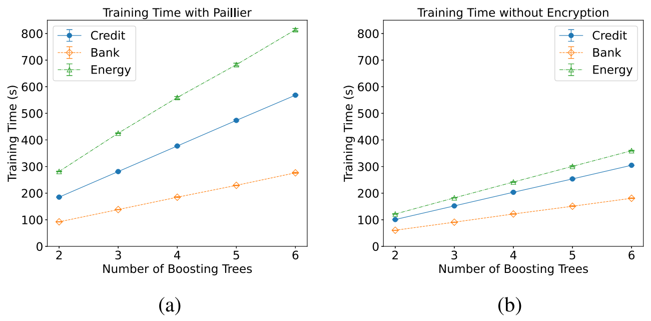


Fig. 12. Total training time over different number of boosting trees, where (a) is the training time with Paillier encryption and (b) is the training time without encryption. (a) Training time with Paillier encryption. (b) Training time without encryption.

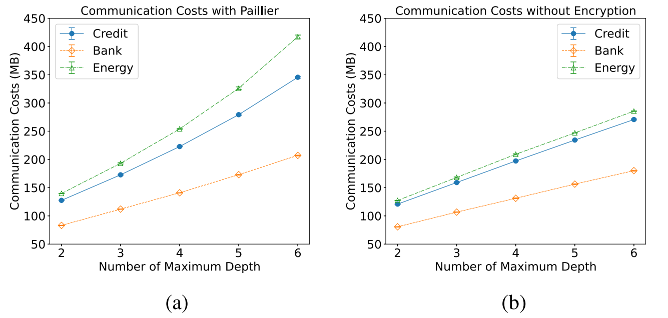


Fig. 14. Total communication costs over different maximum depths of a boosting tree, where (a) is the communication costs with Paillier encryption and (b) is the communication costs without encryption. (a) Communication costs with Paillier encryption. (b) Communication costs without encryption.

without Paillier encryption are presented for comparison. The training time grows exponentially with the increasing maximum depth, and this phenomenon becomes more obvious with Pailler encryption adopted. This makes sense since the computational complexity of one decision is $O(2^n)$, where n is the maximum depth of the tree. More specifically, the training time on the Appliance energy prediction dataset for six different depths is about 2 and 5 min without and with encryption, respectively.

Fig. 12 indicates the training time consumption over different numbers of boosting trees. The training time increases linearly on the three datasets with increase in the number of boosting trees. Similar to the previous results, the time consumption when using the Paillier encryption is much larger than that without encryption. For example, the runtime on the energy dataset for six boosting trees is approximately 800 s with encryption, while it takes only about 350 s without encryption.

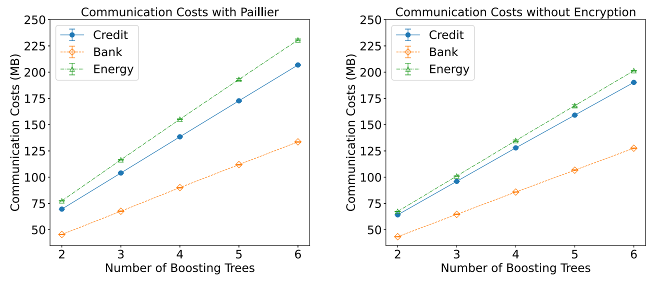
D. Evaluation on the Communication Cost

In this section, we empirically analyze the communication costs (in MB) affected by the number of participating clients, the number of the maximum depth and the number of boosting trees.

Communication costs are measured by varying the number of clients for both cases with and without using the Paillier encryption. As shown in Fig. 13, the total communication costs for the three datasets increase with the number of clients. More specifically, the communication costs go up dramatically from two clients to four clients, and then, increase relatively slightly from four clients to ten clients. Different from the results on the training time, the communication costs with Paillier encryption increase but not as much as the computation time compared with the cases without encryption. This can be attributed to the fact that it is unnecessary to transmit the ciphertexts of the gradients and Hessian values of each data sample for all possible node splits, and only the ciphertexts of the intermediate summation values are required for transmission (see line 27 in Algorithm 2).

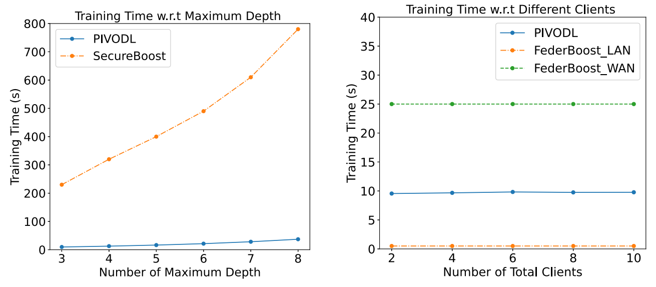
Fig. 14 shows the communication costs over different maximum depths with and without Paillier encryption. It is apparent to see that the communication costs are proportional to the maximum depth in both encryption and nonencryption scenarios. The communication costs are similar for the three datasets when the maximum depth is two, and they grow almost linearly with increase in the tree depth. The communication costs of the Appliances energy prediction dataset are 290 MB without encryption and 425 MB with encryption for a depth of six, which is the largest among all the three datasets.

The communication costs related to the number of boosting trees are shown in Fig. 15. Similar to the previous results, the communication costs increase linearly with the number of trees, since the model size of XGBoost is proportional to the number of trees. And it is clear to see that the impact of encryption



(a) (b)

Fig. 15. Total communication costs over different numbers of boosting trees, where (a) is the communication costs with Paillier encryption, and (b) is the communication costs without encryption.(a) Communication costs with Paillier encryption. (b) Communication costs without encryption.



(a) (b)

Fig. 16. Training time comparisons with SecureBoost and FederBoost, where (a) is the training time comparison with SecureBoost over different maximum tree depth, and (b) is the training time comparison with FederBoost over different number of clients. (a) Time comparison with Secure-Boost. (b) Time comparison with FederBoost.

TABLE III
GUESS ACCURACY ON DIFFERENT DATASETS

Dataset	Guess accuracy, no DP	Guess accuracy, $\epsilon = 2$	Guess accuracy, $\epsilon = 4$	Guess accuracy, $\epsilon = 6$	Guess accuracy, $\epsilon = 8$	Guess accuracy, $\epsilon = 10$
Credit card	68.75%	30.02%	23.02%	34.15%	27.78%	40.93%
Bank marketing	61.57%	39.57%	38.85%	25.85%	39.73%	14.45%

on communication costs is not as significant as that on the training time. The encryption brings a maximum of 25 MB extra communication costs for the model with six boosting trees on the Appliance energy prediction dataset.

E. *Inference of Predicted Labels*

In this section, we assume that each client is curious but honest and attempts to deduce the true label information of other clients through the available leaf weights. The attack strategy is, for instance, to ensemble all the leaf values and guess a label if one client knows several leaf values of one specific data sample. Since the received leaf values of each client are intrinsically determined by the local training data and the learning algorithm, we cannot manually determine the amount of leaf values the attacker uses to infer labels.

To characterize the protection impact of the partial DP, we define the following guess accuracy.

Definition 9 (Guess Accuracy): The accuracy of the attackers guessing the correct labels based on received leaf values.

Note that for a binary classification problem, the data labels are either 0 or 1 and the probability of a correct random guess is 50%. Therefore, if the guess accuracy is equal to or less than 50%, the label privacy is not revealed.

For the sake of brevity, we calculate the expected guess accuracy of all participating clients with and without partial DP using different ϵ values. As shown in Table III, the guess accuracy is more than 60% for both two datasets without using partial DP, which is more accurate than a random guess. After applying the proposed partial DP algorithm, however, the guess accuracies drop dramatically and are all less than 50%. Specifically on the Credit card dataset, the guess accuracy decreases to around 27% for $\epsilon = 10$ and 45% for $\epsilon = 4$. On the Bank marketing dataset, the guess accuracy drops to 14.45% when $\epsilon = 10$. Since the noise in the DP follows the Gaussian distribution, it is normal that a smaller ϵ values will give a higher guess accuracy. And some previous wrong guesses may be changed to correct guesses over different noise levels. Overall, as long as the guess accuracy is less than 50%, the label privacy is considered to be not revealed and the absolute accuracy does not really indicate the privacy preservation performance.

F. *Ablation Study*

In order to sensibly compare the proposed PIVODL algorithm with some related work described in the introduction section, some modifications are needed to adapt PIVODL to the conventional VFL. The "split" and "source" client scheme used for distributed labels is no longer necessary for the scenarios of centralized labels. And HE can also be removed without privacy leakage risk because all the impurity calculations and prediction updates can be performed on the guest client.

SecureBoost [20], FederBoost [23], and Pivot [22] are compared on the credit card dataset. The training time with respect to the different maximum depths of booster trees between PIVODL and SecureBoost is shown in Fig. 16(a). It is clear to find that PIVODL runs much faster than SecureBoost for all different maximum tree depths. The reason for this is that PIVODL for centralized labels does not require any encryption operations during training, because all the summation computations of gradients and Hessians are performed on the guest client only. By contrast, SecureBoost computes the summations in terms of ciphertexts for every node split on each host client, and consequently, the gradients must be encrypted to prevent the leakage of raw gradients.

The training time comparison related to different numbers of total clients between PIVODL and FederBoost is shown in Fig. 16(b). We can find that the training time of all three cases remain nearly the same over different numbers of clients. The time consumption of FederBoost trained by LAN and WAN networks are approximately 0.5 and 25 s, respectively. And the training time of PIVODL is around 9.6 s, which is smaller than the WAN network but larger than the LAN network.

TABLE IV
SYSTEM PERFORMANCE COMPARISON

Systems	Test performance		Speed	Distributed Labels
	Accuracy	AUC		
PIVODL	0.8223	0.7724	Fast	✓
SecureBoost	0.8180	0.7701	Slow	✗
FederBoost	/	0.7779	Very Fast	✗
Pivot	0.8251	/	Very Slow	✗

All results of model performance comparisons, except for the training time, are listed in Table IV. Both test accuracy and area under the curve (AUC) are provided. From these results, we can find that all these systems share a similar final test performance.

VI. CONCLUSION

In this article, we propose a secure learning system, called PIVODL, for privately training XGBoost decision tree models in a VFL environment with labels distributed on multiple clients. A secure node split protocol and a privacy-preserving tree training algorithm are proposed by requiring the source and split clients to separately split the data samples and calculate the corresponding impurity scores, effectively defending against differential attacks that may be encountered during the boosting tree node splits. In addition, a partial DP mechanism is adopted to deal with label privacy so that no client is able to guess the correct data labels through the revealed leaf weights.

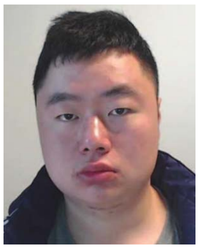
Our experimental results empirically indicate that the proposed PIVODL framework is able to securely construct ensemble trees with a negligible performance degradation. We show that the test performance of the resulting decision trees is relatively insensitive to different ϵ values of the introduced DP, since the proposed algorithm adds Gaussian noise only to the leaf weights that are sent to the source clients. Moreover, the partial DP can effectively prevent labels of the data from being revealed through the received leaf weights.

Although the proposed PIVODL system shows promising performance in VFL, improving training efficiency remains challenging. The encryption in PIVODL still takes a large proportion of both learning time and communication costs. Therefore, our future work will be dedicated to developing a light weighted and efficient secure XGBoost system for VFL with labels distributed on multiple devices.

REFERENCES

[1] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. Y. Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. Artif. Intell. Statist.*, 2017, pp. 1273–1282.
[2] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *ACM Trans. Intell. Syst. Technol.*, vol. 10, no. 2, pp. 1–19, Jan. 2019.
[3] L. T. Phong, Y. Aono, T. Hayashi, L. Wang, and S. Moriai, "Privacy-preserving deep learning via additively homomorphic encryption," *IEEE Trans. Inf. Forensics Secur.*, vol. 13, no. 5, pp. 1333–1345, May 2018.
[4] S. Truex *et al.*, "A hybrid approach to privacy-preserving federated learning," in *Proc. 12th ACM Workshop Artif. Intell. Secur.*, New York, NY, USA, 2019, pp. 1–11.
[5] R. Xu, N. Baracaldo, Y. Zhou, A. Anwar, and H. Ludwig, "Hybridalpha: An efficient approach for privacy-preserving federated learning," in *Proc. 12th ACM Workshop Artif. Intell. Secur.*, New York, NY, USA, 2019, pp. 13–23.
[6] C. Zhang, S. Li, J. Xia, W. Wang, F. Yan, and Y. Liu, "Batchcrypt: Efficient homomorphic encryption for cross-silo federated learning," in *Proc. USENIX Annu. Tech. Conf.*, Jul. 2020, pp. 493–506.
[7] H. Zhu, R. Wang, Y. Jin, K. Liang, and J. Ning, "Distributed additive encryption and quantization for privacy preserving federated deep learning," *Neurocomputing*, vol. 463, pp. 309–327, 2021.
[8] R. Shokri and V. Shmatikov, "Privacy-preserving deep learning," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur.*, New York, NY, USA, 2015, pp. 1310–1321.
[9] T. Orekondy, S. J. Oh, Y. Zhang, B. Schiele, and M. Fritz, "Gradient-leaks: Understanding and controlling deanonymization in federated learning," in *Proc. NeurIPS Workshop Federated Learn. Data Privacy Confidentiality*, 2019.
[10] J. Geiping, H. H. B. Dröge, and M. Moeller, "Inverting gradients—How easy is it to break privacy in federated learning?," in *Proc. Adv. Neural Inf. Process. Syst.*, 2020, vol. 33, pp. 16937–16947.
[11] H. Li and T. Han, "An end-to-end encrypted neural network for gradient updates transmission in federated learning," in *Proc. Data Compression Conf. (DCC)*, 2019, pp. 589–589, 2019, doi: 10.1109/DCC.2019.00101.
[12] O. Goldreich, "Secure multi-party computation," *Manuscript Preliminary Version*, vol. 78, 1998.
[13] K. Bonawitz *et al.*, "Practical secure aggregation for privacy-preserving machine learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, New York, NY, USA, 2017, pp. 1175–1191.
[14] C. Gentry *et al.*, *A Fully Homomorphic Encryption Scheme*, vol. 20. Stanford, CA, USA: Stanford Univ., 2009.
[15] C. Dwork, "Differential privacy: A survey of results," in *Theory and Applications of Models of Computation*, M. Agrawal, D. Du, Z. Duan, and A. Li, Eds. Berlin, Germany: Springer, 2008, pp. 1–19.
[16] S. Hardy *et al.*, "Private federated learning on vertically partitioned data via entity resolution and additively homomorphic encryption," 2017, *arXiv:1711.10677*.
[17] R. Nock *et al.*, "Entity resolution and federated learning get a federated resolution," 2018, *arXiv:1803.04035*.
[18] Y. Liu, X. Zhang, and L. Wang, "Asymmetrical vertical federated learning," 2020, *arXiv:2004.07427*.
[19] S. Yang, B. Ren, X. Zhou, and L. Liu, "Parallel distributed logistic regression for vertical federated learning without third-party coordinator," 2019, *arXiv:1911.09824*.
[20] K. Cheng *et al.*, "SecureBoost: A lossless federated learning framework," *IEEE Intell. Syst.*, vol. 36, no. 6, pp. 87–98, Nov./Dec. 2021.
[21] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, New York, NY, USA, 2016, pp. 785–794.
[22] 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, pp. 2090–2103, Jul. 2020.
[23] Z. Tian, R. Zhang, X. Hou, J. Liu, and K. Ren, "Federboost: Private federated learning for GBDT," 2020, *arXiv:2011.02796*.
[24] Y. Liu *et al.*, "A communication efficient vertical federated learning framework," 2019, *arXiv:1912.11187*.
[25] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: Distributed data-parallel programs from sequential building blocks," in *Proc. 2nd ACM SIGOPS/Eur. Conf. Comput. Syst.*, 2007, pp. 59–72.
[26] M. Isard and Y. Yu, "Distributed data-parallel computing using a high-level programming language," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2009, pp. 987–994.
[27] Y. Yu, M. Isard, D. Fetterly, M. Budiu, Ú. Erlingsson, P. K. Gunda, and J. Currey, "Dryadlinq: A system for general-purpose distributed data-parallel computing using a high-level language," in *Proc. 8th USENIX Conf. Oper. Syst. Des. Implementation*, 2008, vol. 8, pp. 1–14.
[28] D. W. Hosmer Jr., S. Lemeshow, and R. X. Sturdivant, *Applied Logistic Regression*, vol. 398. Hoboken, NJ, USA: Wiley, 2013.
[29] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep Learning*, vol. 1. Cambridge, MA, USA: MIT Press, 2016.
[30] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.
[31] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Trans. Inf. Theory*, vol. 31, no. 4, pp. 469–472, Jul. 1985.
[32] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Proc. Adv. Cryptol. EUROCRYPT*, (J. Stern, ed.), Berlin, Germany, 1999, pp. 223–238.

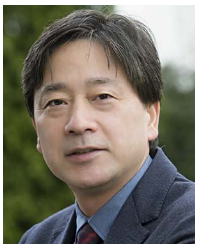
[33] C. Dwork, F. McSherry, K. Nissim, and A. Smith, “Calibrating noise to sensitivity in private data analysis,” in *Proc. Theory Cryptogr. Conf.*, 2006, pp. 265–284.
[34] C. Dwork, K. Kenthapadi, F. McSherry, I. Mironov, and M. Naor, “Our data, ourselves: Privacy via distributed noise generation,” in *Proc. Annu. Int. Conf. Theory Appl. Cryptogr. Techn.*, 2006, pp. 486–503.
[35] M. Abadi *et al.*, “Deep learning with differential privacy,” in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2016, pp. 308–318.
[36] K. Wei *et al.*, “Federated learning with differential privacy: Algorithms and performance analysis,” *IEEE Trans. Inf. Forensics Secur.*, vol. 15, pp. 3454–3469, Apr. 2020.
[37] R. C. Geyer, T. Klein, and M. Nabi, “Differentially private federated learning: A client level perspective,” 2017, *arXiv:1712.07557*.
[38] V. Kolesnikov, N. Matania, B. Pinkas, M. Rosulek, and N. Trieu, “Practical multi-party private set intersection from symmetric-key techniques,” in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2017, pp. 1257–1272.
[39] B. Pinkas, T. Schneider, and M. Zohner, “Faster private set intersection based on {OT} extension,” in *Proc. 23rd USENIX Secur. Symp.*, 2014, pp. 797–812.
[40] I.-C. Yeh and C.-H. Lien, “The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients,” *Expert Syst. Appl.*, vol. 36, no. 2, pp. 2473–2480, 2009.
[41] S. Moro, P. Cortez, and P. Rita, “A data-driven approach to predict the success of bank telemarketing,” *Decis. Support Syst.*, vol. 62, pp. 22–31, 2014.
[42] L. M. Candanedo, V. Feldheim, and D. Deramaix, “Data driven prediction models of energy use of appliances in a low-energy house,” *Energy Build.*, vol. 140, pp. 81–97, 2017.
[43] H. Zhu, J. Xu, S. Liu, and Y. Jin, “Federated learning on non-IID data: A survey,” *Neurocomputing*, vol. 465, pp. 371–390, 2021.
[44] Python Paillier Library, CSIRO Data61, 2013. [Online]. Available: https://github.com/data61/python-paillier



Hangyu Zhu received the B.Sc. degree in architecture electrical and intelligence from Yangzhou University, Yangzhou, China, in 2015, the M.Sc. degree in electrical and electronics from RMIT University, Melbourne, VIC, Australia, in 2017, and the Ph.D. degree in computer science from the University of Surrey, Guildford, U.K., in 2021.

His main research interests include federated learning, privacy-preserving machine learning, and evolutionary federated neural architecture search.

Rui Wang received the B.Sc. degree in applied physics from the Beijing University of Posts and Telecommunications, Beijing, China, in 2017, and the M.Sc. degree in cyber security from the University of Southampton, Southampton, U.K., in 2018. He is currently working toward the Ph.D. degree, focusing on privacy-preserving machine learning, with the Department of Intelligent Systems, Delft University of Technology, Delft, The Netherlands.



Yaochu Jin (Fellow, IEEE) received the B.Sc., M.Sc., and Ph.D. degrees, all in automatic control from Zhejiang University, Hangzhou, China, in 1988, 1990, and 1996, respectively, and the Dr.-Ing. degree in neuroinformatics from Ruhr University Bochum, Bochum, Germany, in 2001.

He is an Alexander von Humboldt Professor of artificial intelligence with the Faculty of Technology, Bielefeld University, Bielefeld, Germany. He is also a Distinguished Chair Professor of computational intelligence with the Department of Computer Science, University of Surrey, Guildford, U.K. He was a Finland Distinguished Professor in Finland, and Changjiang Distinguished Visiting Professor, in China. His research interests include evolutionary optimization, evolutionary and multi-objective machine learning, secure and privacy-preserving machine learning, and evolutionary developmental approaches to artificial intelligence.

Prof. Jin is currently the Editor-in-Chief for the IEEE TRANSACTIONS ON COGNITIVE AND DEVELOPMENTAL SYSTEMS and *Complex and Intelligent Systems*. He was an IEEE Distinguished Lecturer (during 2013, 2015, and 2017–2019) and was the Vice President for Technical Activities of the IEEE Computational Intelligence Society (during 2014–2015). He was the recipient of the 2015, 2017, and 2020 IEEE Computational Intelligence Magazine Outstanding Paper Award, and the 2018 and 2021 IEEE Transactions on Evolutionary Computation Outstanding Paper Award. He was named a Highly Cited Researcher by the Web of Science Group for 2019-2021. He is a Member of Academia Europaea.

Kaitai Liang (Member, IEEE) received the Ph.D. degree from the Department of Computer Science, City University of Hong Kong, Hong Kong.

He joined the Delft University of Technology, The Netherlands, in 2020. Before that he was an Assistant Professor of secure systems with the Department of Computer Science, University of Surrey, U.K. His research interests include applied cryptography and information security; in particular, data encryption, blockchain security, post-quantum crypto, privacy enhancing technology, and privacy-preserving machine learning.