# SGBoost: An Efficient and Privacy-Preserving Vertical Federated Tree Boosting Framework

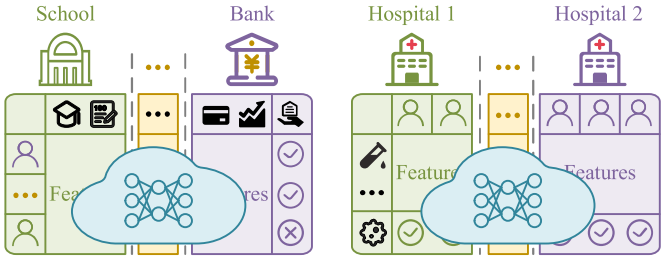
Jiaqi Zhao, Hui Zhu, *Senior Member, IEEE*, Wei Xu, Fengwei Wang, Rongxing Lu, *Fellow, IEEE*, and Hui Li, *Member, IEEE*

***Abstract*— Aiming at balancing data privacy and availability, Google introduces the concept of federated learning, which can construct global machine learning models over multiple participants while keeping their raw data localized. However, the exchanged parameters in traditional federated learning may still reveal the data information. Meanwhile, the training data are usually partitioned vertically in real-world scenes, which causes difficulties in model construction. To tackle these problems, in this paper, we propose an efficient and privacy-preserving vertical federated tree boosting framework, namely SGBoost, where multiple participants can collaboratively perform model training and query without staying online all the time. Specifically, we first design secure bucket sharing and best split finding algorithms, with which the global tree model can be constructed over vertically partitioned data; meanwhile, the privacy of training data can be well guaranteed. Then, we design an oblivious query algorithm to utilize the trained model without leaking any query data or results. Moreover, SGBoost does not require multi-round interactions between participants, significantly improving the system efficiency. Detailed security analysis shows that SGBoost can well guarantee the privacy of raw data, weights, buckets, and split information. Extensive experiments demonstrate that SGBoost can achieve high accuracy comparable to centralized training and efficient performance.**

***Index Terms*— Vertical federated learning, tree boosting, privacy-preserving, efficiency.**

## I. Introduction

DUE to the growing concerns about data privacy in modern society, coupled with the enacted serious privacy laws like GDPR [1], how to achieve both privacy preservation and data availability, has become an urgent problem. For example, GDPR strictly prohibits data sharing between different institutions, even different internal departments, which hinders the utilization of precious data. Subsequently, the introduction of federated learning (FL) [2] provides a solution to the above problem, as shown in Fig. 1, where multiple participants can collaboratively construct global machine learning models while keeping their data localized.




(a) Vertical Federated Learning (b) Horizontal Federated Learning

Fig. 1. The architectures of federated learning. (a) Vertical FL where participants have common data samples but different features. (b) Horizontal FL where participants have common features but different data samples.

Nevertheless, there are still many challenges in FL [3]. On the one hand, traditional FL cannot provide a formal guarantee of privacy. Although the raw data never leave the devices in FL, the exchanged parameters can also be used to infer data information [4], [5], [6], [7]. On the other hand, participants' training data are more likely to be vertically partitioned in real scenes, i.e., their data have a common sample space but a different feature space, which causes difficulties in decomposing the loss function at each participant [8], [9].

To tackle these challenges, plenty of secure vertical FL schemes have been proposed for the boosting tree model. Schemes [10] and [11] protect the gradients of boosting trees by homomorphic encryption [12], but the sensitive model weights and splits still will be leaked to participants. Meanwhile, since every tree boosting requires sending the ciphertext gradients to all participants, the system efficiency is less high. Based on differential privacy [13], [14], schemes [15] and [16] perturb the data buckets or model weights for privacy preservation, but the added noises will inevitably reduce the model accuracy. Scheme [17] strongly guarantees privacy by combining homomorphic encryption [12] and secure multi-party computation [18], but its overhead is unacceptable. Moreover, almost all schemes require participants to stay online all the time during model training, which is less realistic due to network and device constraints.

In this paper, we propose an efficient and privacy-preserving vertical federated framework for the boosting tree model, named SGBoost. In SGBoost, bucket sharing and best split finding algorithms are designed to achieve high-accuracy boosting tree model training, which is protected by secret

Manuscript received 9 June 2022; revised 17 November 2022; accepted 16 December 2022. Date of publication 28 December 2022; date of current version 6 January 2023. This work was supported in part by the National Natural Science Foundation of China under Grant U22B2030 and Grant 61972304 and in part by the Science Foundation of the Ministry of Education under Grant MCM20200101. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. George Theodorakopoulos. *(Corresponding author: Hui Zhu.)*

Jiaqi Zhao, Hui Zhu, Wei Xu, Fengwei Wang, and Hui Li are with the School of Cyber Engineering, Xidian University, Xi'an, Shaanxi 710126, China (e-mail: jq_zhao@stu.xidian.edu.cn; zhuhui@xidian.edu.cn; xuwei_1@stu.xidian.edu.cn; wangfengwei@xidian.edu.cn; lihui@mail.xidian.edu.cn).

Rongxing Lu is with the Faculty of Computer Science, University of New Brunswick, Fredericton, NB E3B 5A3, Canada (e-mail: rlu1@unb.ca).

Digital Object Identifier 10.1109/TIFS.2022.3232955

sharing [19] and functional encryption (FE) [20]. Subsequently, based on a novel symmetric homomorphic encryption (SHE) [21], we propose an oblivious query algorithm to provide secure model query services. Specifically, our main contributions are the following threefold.

- *SGBoost achieves high-accuracy tree boosting over vertically partitioned data.* Considering that the tree model training process is only related to the constant data orders, we first design a bucket sharing algorithm to share the data buckets of participants' partial features, through which participants do not need to stay online all the time. Then, based on the shared bucket information, the best split finding algorithm is proposed to perform high-accuracy global model training.
- *SGBoost is privacy-preserving in the training and query processes.* By carefully applying FE to the best split finding algorithm, SGBoost well guarantees the privacy of training data, bucket information, and split information. Moreover, we design an oblivious query algorithm to perform model queries over SHE ciphertexts, where the query data and results are also protected well.
- *SGBoost is efficient in both computational cost and communication overhead.* In SGBoost, there is no requirement for multi-round interactions between participants; meanwhile, most time-consuming calculations are securely outsourced to a high-performance cloud service provider, which improves the training efficiency significantly. We conduct experiments on several real-world datasets, and the experimental results show that SGBoost can achieve low computational cost and communication overhead for both model training and query.

The rest of this paper is organized as follows. In Section II, we define the system model and security requirements. In Section III, we outline some building blocks of SGBoost. Section IV gives a high-level description of our algorithm design. Section V introduces the model training and query processes of SGBoost detailedly, followed by the security analysis and performance evaluation in Section VI and Section VII. Finally, we review related works in Section VIII and draw a conclusion in Section IX.

## II. Models, Security Requirements, and Design Goals

In this section, we formally describe our system model and threat model, and then identify our design goals.

### A. System Model

In our system model, we mainly focus on i) constructing a global boosting tree model efficiently and securely over vertically partitioned data, and ii) using the trained model for providing secure query services, which mainly involves three types of entities (a cloud service provider CSP, multiple participants $\{P_1, P_2, \cdots, P_K\}$, and a query user U) and two processes (model training and model query), as shown in Fig. 2.

*1) Participants:* Participants $\{P_1, P_2, \cdots, P_K\}$ are different institutions (e.g., banks, securities companies, etc.) with vertically partitioned training data. Without loss of generality, we assume participant $P_K$ has all data labels. Since the label

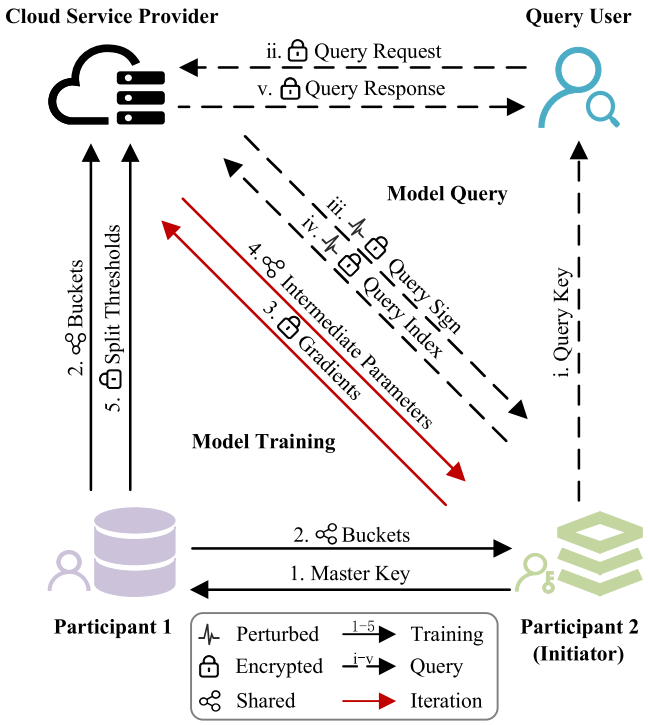



Fig. 2. System model under consideration (with two participants).

information is essential for boosting tree model training, $P_K$ can naturally dominate the FL process and obtain the global model, and other passive participants can get rewards from $P_K$ according to their contributions to the global model. It should be noted that the initiator or active participant is a generic setting adopted in most vertical FL schemes [11], [16], [22].

*2) Cloud Service Provider:* CSP is a cloud service provider with powerful computing and storage capabilities (e.g., Amazon, Microsoft, etc.), which is responsible for assisting in model training and providing model query services.

*3) Query User:* U is a query user who wants to securely obtain the query result on the trained model.

*Model training:* During model training, $P_K$ first generates and distributes master keys for passive participants. Then, passive participants encode and secretly share their data bucket information with CSP and $P_K$. After that, the model is constructed iteratively between CSP and $P_K$ based on the proposed secure best split finding algorithm. When the model converges, passive participants outsource the ciphertext split thresholds to CSP for providing query services.

*Model query:* During model query, U first obtains the query key from $P_K$ and sends the ciphertext query data to CSP. Then, with the assistance of $P_K$, CSP calculates the ciphertext query response based on the proposed oblivious query algorithm. Finally, the ciphertext query response is returned to U and decrypted.

### B. Threat Model and Security Requirements

In our threat model, CSP, U, and all participants are considered as honest-but-curious [23]. That is, they are obliged to faithfully execute the stipulated protocol process, but may

infer others' sensitive information as much as possible. For example, i) CSP or initiator $P_K$ may try to infer other participants' training data information from received shared buckets and parameters; ii) CSP or initiator $P_K$ may infer the query data or result from the query request. Moreover, participants are considered non-collusion with CSP in SGBoost to prevent leaking their own data information. Under these threat assumptions, SGBoost should satisfy the following security requirements.

*1) Privacy of Raw Data:* In FL, the raw training and query data directly involve user privacy, which should be strictly protected without a doubt.

*2) Privacy of Bucket Information:* On obtaining the bucket information of a certain participant, its data distribution will be leaked, which also causes a threat to user privacy. Therefore, participants' bucket information also should be protected.

*3) Privacy of Split Information:* As a part of the bucket information, the split information may also reveal the data distribution. Therefore, the split information also should be protected in SGBoost.

*4) Privacy of Weights:* As proved in [11], when executing model queries, passive participants can infer the first tree's leaf purity from model weights. Though less sensitive than the above items, it still has potential risks to user privacy and should be protected in SGBoost.

Moreover, there exist model stealing attacks [24] extracting the model parameters by querying the model multiple times, which can be defended by perturbing the query results or limiting the query times but will inevitably sacrifice the model availability [25]. In SGBoost, we aim to provide high-accuracy model training and query, so it is not our focus to design defensive strategies for model stealing attacks. Meanwhile, there are some active attacks (e.g., Sybil attack [26], data poisoning attack [27], [28], etc.) in vertical federated learning, which also are currently out of the scope of this paper and may be considered in future work.

### C. Design Goals

Under the above system and security models, aiming to achieve efficient and privecy-preserving vertical tree boosting, SGBoost should satisfy the following three objectives.

- *High accuracy.* In modern society, data have become a precious resource, which can greatly improve the accuracy of machine learning models and help people make more precise decisions. Therefore, SGBoost should make full use of the data worth of participants (even if dropping out midway) and construct the global model accuracy-losslessly.
- *Strong privacy.* Although no raw data flow out of the participants' local in FL, the exchanged parameters still leak the data information (containing values and distributions), which causes great challenges to user privacy. Therefore, SGBoost should guarantee the training privacy containing the raw training data, bucket information, and split information, and guarantee the query privacy containing the query data and results.
- *Low overhead.* In most instances, due to the introduction of some cryptographic operations like homomorphic encryption, the privacy-preserving machine learning schemes will add at least two orders of magnitude overhead than origin schemes. Therefore, the privacy-preserving algorithms should be carefully designed in SGBoost to achieve low computation and communication overhead.

## III. Preliminaries

This section briefly reviews one of the most widely used boosting tree models—XGBoost, functional encryption, and the SHE technique, which serve as the fundamental building blocks of SGBoost.

### A. XGBoost

As one of the most widely used boosting tree models in practice, XGBoost [29] supports parallel tree boosting, which can solve many machine learning tasks (such as classification and regression) in a fast and accurate way. Specifically, in XGBoost, the prediction output of a data sample $X$ is calculated with $R$ trees as $\hat{y} = \sum_{r=1}^{R} f_r(X)$, where $f_r$ denotes the mapping of the $r$th tree.

Given a dataset $\mathcal{D} = \{X_n, y_n\}_{n=1}^{N} \in \mathbb{R}^{N\times(D+1)}$ with $N$ samples and $D$ features, the model is trained by greedily adding a tree at the $r$th training round to minimize the loss

$$\sum_{n=1}^{N}\left(l(y_n, \hat{y}_n^{(r-1)}) + g_n f_r(X_n) + \frac{1}{2}h_n f_r^2(X_n)\right) + \Omega(f_r),$$

where $l$ is the loss function, $\hat{y}_n^{(r-1)}$ is the prediction output after the $(r-1)$th training round, and $g_n = \partial_{\hat{y}^{(r-1)}} l(y_n, \hat{y}^{(r-1)})$ and $h_n = \partial^2_{\hat{y}^{(r-1)}} l(y_n, \hat{y}^{(r-1)})$ are the first and second order gradients.

The $r$th tree is constructed by continuously splitting the node $I^{(r,t)}$ until it reaches the maximum tree depth $T$, i.e., $t \geq 2^T$. To be specific, each best split can maximize the gains among all data buckets, and each gain $\mathcal{G}$ can be calculated as

$$\frac{1}{2}\left(\frac{(\sum_{n\in I_L^{(r,t)}} g_n)^2}{\sum_{n\in I_L^{(r,t)}} h_n+\lambda} + \frac{(\sum_{n\in I_R^{(r,t)}} g_n)^2}{\sum_{n\in I_R^{(r,t)}} h_n+\lambda} - \frac{(\sum_{n\in I^{(r,t)}} g_n)^2}{\sum_{n\in I^{(r,t)}} h_n+\lambda}\right) - \gamma, \tag{1}$$

where $I_L^{(r,t)}$ and $I_R^{(r,t)}$ denote the left and right sample spaces splited by a certain bucket, and $\gamma$ and $\lambda$ are two regularization parameters.

After obtaining the best structure of the $r$th tree, the weights of leaf node $I^{(r,t)}(t = 2^T, \cdots, 2^{T+1}-1)$ can be calculated as

$$\omega_n^{(r)}(n \in I^{(r,t)}) = -\frac{\sum_{n\in I^{(r,t)}} g_n}{\sum_{n\in I^{(r,t)}} h_n + \lambda}.$$

It is noteworthy that there are two kinds of bucketing strategies in XGBoost [29]. The global strategy proposes all buckets at the beginning of model training and uses the same buckets for all levels' split finding, while the local strategy re-proposes buckets after each split. The global strategy can achieve a comparable model accuracy with the local one when more buckets are used. In SGBoost, we adopt the global bucketing strategy to design our bucket sharing algorithm, which can significantly reduce communication and computational overhead.

### B. Functional Encryption

As a generalization of public-key encryption, functional encryption [20], [30] can finely control the revealed information (i.e., calculated function) to a receiver, which can be written as FE = (Gen, Der, Enc, Dec). In the following, we briefly introduce a functional encryption construction [31] for calculating the inner-product $\vec{x} \cdot \vec{y}$, which is based on additive homomorphic public-key encryption HPKE = (Gen, Enc, Dec, C).

- $\text{FE}.\text{Gen}(\kappa, \zeta) \rightarrow (mpk, msk)$: Given the security parameter $\kappa$ and vector length $\zeta$, first generate $\zeta$ key pairs $(sk_1, pk_1), \cdots, (sk_\zeta, pk_\zeta)$ with $\text{HPKE}.\text{Gen}(\kappa)$. Then, output the master public key $mpk = (pk_1, \cdots, pk_\zeta)$ and master secret key $msk = (sk_1, \cdots, sk_\zeta)$.
- $\text{FE}.\text{Enc}(mpk, \vec{x}) \rightarrow [\![\vec{x}]\!]$: First, choose a shared random number $r$ from the randomness space of HPKE and calculate the commitment $[\![x_0]\!] = \text{HPKE}.\text{C}(r)$. Then, given each $x_i \in \vec{x} (i = 1, \cdots, \zeta)$, calculate $[\![x_i]\!] = \text{HPKE}.\text{Enc}(pk_i, x_i; r)$. Finally, output the ciphertext $[\![\vec{x}]\!] = ([\![x_0]\!], \cdots, [\![x_\zeta]\!])$.
- $\text{FE}.\text{Der}(msk, \vec{y}) \rightarrow sk_y$: Calculate the inner-product $\vec{y} \cdot msk$ as secret key $sk_y$.
- $\text{FE}.\text{Dec}([\![\vec{x}]\!], \vec{y}, sk_y) \rightarrow \vec{x} \cdot \vec{y}$: With $sk_y$, the inner-product of $\vec{x}$ and $\vec{y}$ can be decrypted by calculating $\text{HPKE}.\text{Dec}\left(sk_y, ([\![x_0]\!], \prod_{i=1}^{\zeta} [\![x_i]\!]^{y_i})\right)$.

### C. The SHE Technique

The symmetric homomorphic encryption (SHE) technique [21], [32] is one kind of somewhat homomorphic encryption [33], which allows ciphertext additions and a limited number of ciphertext multiplications. It satisfies indistinguishability under chosen-plaintext attack (IND-CPA) [34], [35]. The SHE technique contains three functions, namely, Gen, Enc, and Dec, which are described detailedly in the following.

- $\text{SHE}.\text{Gen}(k_0, k_1, k_2) \rightarrow pp, ssk$: Given the security parameters $(k_0, k_1, k_2)$ satisfying $k_1 \ll k_2 < k_0$, first select two large prime numbers $p, q$ with $|p| = |q| = k_0$ and calculate $\mathcal{N} = pq$. Then, select a $k_2$-bit random number $\mathcal{L}$. Finally, the symmetric key $ssk$ is $(p, q, \mathcal{L})$ and the public parameter $pp$ is $(k_0, k_1, k_2, \mathcal{N})$.
- $\text{SHE}.\text{Enc}(m) \rightarrow c$: Given a message $m \in (-2^{k_1}, 2^{k_1})$, the ciphertext $[m]$ is calculated with $ssk$ as

$$[m] = \text{Enc}(m) = (r\mathcal{L} + m)(1 + r'p) \bmod \mathcal{N},$$

where $r$ and $r'$ are $k_2$-bit and $k_0$-bit random numbers.

- $\text{SHE}.\text{Dec}([m]) \rightarrow m$: Given a ciphertext $[m]$, the corresponding plaintext $m$ can be retrieved with $ssk$ by computing

$$m' = ([m] \bmod p) \bmod \mathcal{L},$$

and

$$m = \text{Dec}([m]) = \begin{cases} m', (m' < \dfrac{\mathcal{L}}{2}) \\ m' - \mathcal{L}, (else) \end{cases}.$$

Specifically, given two ciphertexts $[m_1]$, $[m_2]$, and a plaintext $m_3$, the SHE technique has the following four homomorphic properties: $\text{SHE}.\text{Dec}([m_1] + [m_2]) = m_1 + m_2$, $\text{SHE}.\text{Dec}([m_1] + m_3) = m_1 + m_3$, $\text{SHE}.\text{Dec}([m_1] * [m_2]) = m_1 * m_2$, and $\text{SHE}.\text{Dec}([m_1] * m_3) = m_1 * m_3, (m_3 > 0)$.

Moreover, with different settings of security parameters, SHE can support different multiplication depths, and greater security parameters will bring more multiplication depth. Specifically, given parameters $k_0$ and $k_2$, the depth $\theta$ can be represented as $\theta = \lfloor \frac{k_0}{2k_2} - 1 \rfloor$ [34].

## IV. Technical Overview

In this section, we give a high-level description of our algorithm design. According to the introduction in Section III-A, the most core and frequent operation in boosting tree model training is to continuously find the best splits, i.e., calculate the gains and find the maximum one. From (1), we find that it needs two kinds of data information to calculate gains, where one is the gradients $g$ and $h$, and the other is the bucket information $I_L$ and $I_R$.

In SGBoost, the initiator $\text{P}_K$ has data labels and model weights, so it can calculate the gradients locally. Meanwhile, the data buckets are determined by other passive participants based on their data distribution. Since both the gradients and buckets contain sensitive data information, they cannot be directly shared to calculate gains.

- *Challenge 1. How to securely and efficiently share the necessary information for gain calculations?*

Different from other schemes in which $\text{P}_K$ shares changeful ciphertext gradients to every passive participant for each tree construction, we try to secretly share the constant bucket information only once. We have an observation that the bucket information is only related to data orders but not specific data values, so we can encode each bucket to one boolean vector. Moreover, since the global bucketing strategy is adopted in SGBoost, the buckets remain constant throughout the model training process.

Therefore, based on our bucket sharing algorithm, passive participants only need to share their boolean buckets once and do not need to stay online during model training, which improves the system efficiency significantly.

After bucket sharing, CSP and $\text{P}_K$ respectively receive a part of the buckets, and they need to collaboratively calculate gains and find the best splits.

- *Challenge 2. How to find the best splits in an efficient and privacy-preserving way?*

By observing (1), we find that the calculations for gains all involve summing gradients in different buckets, which is also equivalent to calculating the inner products between gradients and our proposed bucket vectors. So we utilize a simple inner-product FE to design our secure best split finding algorithm. Specifically, for each split finding, $\text{P}_K$ encrypts gradients with FE public key and sends them to CSP. Then, based on FE, CSP can calculate the inner products between gradients and its shared bucket vectors, and assist $\text{P}_K$ in calculating all gains.

Since most intensive calculations are securely outsourced to a high-performance cloud, the split findings can be efficiently executed. In terms of security, since the inner products are the summations of plenty of gradients, the privacy of gradients

can be well protected. Meanwhile, the split information is also protected by embedding to ciphertext gradients. Once obtaining the best splits, $\mathrm{P}_K$ can update the sample space by calculating ANDs between it and the best split vector, and then iteratively execute best split findings for constructing trees.

When the model training is completed, $\mathrm{P}_K$ obtains the best tree structure and weights, but the split thresholds are still mastered by passive participants, which are also sensitive and necessary for model querying.

• *Challenge 3. How to fully protect split thresholds, query data, and query results when executing model queries?*

For protecting the split thresholds, based on the SHE technique, passive participants first encrypt and outsource them to CSP. Similarly, $\mathrm{P}_K$ also outsources the ciphertext weights to CSP for providing query services. Hence the queries should be executed over these ciphertext model parameters while protecting both query data and results.

The core of boosting tree model query is to compare the query data and thresholds of every splits and to judge which leaf node the data belong to. In SGBoost, we design an oblivious query algorithm, which protects the query signs with a coin mechanism and calculates query results over SHE ciphertexts. First, CSP perturbs the ciphertext comparison results with blinding coins. Then, for facilitating ciphertext calculations, $\mathrm{P}_K$ decrypts and encodes the perturbed signs to the ciphertext index ($[0], [1]$) or ($[1], [0]$). Finally, CSP can recover the true indexes and obtain the ciphertext query results by calculating the multiplications of ciphertext weights and indexes, where a path with all indexes [1]s indicates the query data belong to this leaf node.

Since the signs of comparison results are perturbed, $\mathrm{P}_K$ cannot obtain the query results. Meanwhile, the values of comparison results are also perturbed, so the query data cannot be inferred by $\mathrm{P}_K$. Moreover, all calculations in CSP are based on SHE ciphertexts, so there is no sensitive information leaked to it. Therefore, in our oblivious query algorithm, both CSP and $\mathrm{P}_K$ cannot get any information about the query requests.

## V. Proposed Scheme

In this section, detailed descriptions of model training and query processes are first provided. Then, the correctness of the algorithms is proved to ensure the model accuracy. Moreover, the commonly used notations are listed in TABLE I.

TABLE I
NOTATIONS IN SGBOOST

| Notations | Definition |
|---|---|
| $K, N$ | The number of participants and data samples. |
| $T, R$ | Maximum tree depth and number. |
| $\epsilon, \eta$ | Precision parameter and learning rate. |
| $\gamma, \lambda$ | Regularization parameters. |
| $\kappa, k_0, k_1, k_2$ | Security parameters. |
| $ssk, ssk'$ | SHE symmetric keys. |
| $\sigma_k$ | SHE encryption parameter of $\mathrm{P}_k$. |
| $(mpk, msk)$ | FE master key pair. |
| $sk_s^{(k,d)}$ | FE secret key for $\langle B_s \rangle_2^{(k,d)}$. |
| $D_k, S_k$ | Feature and bucket numbers of $\mathrm{P}_k$. |
| $\tau_s^{(k,d)}$ | The $s$th bucketing threshold of $X_d^{(k)}$. |
| $\mathcal{M}^{(k,d)}, B_s^{(k,d)}, b_{s,n}^{(k,d)}$ | Boolean bucket matrix, vector, and value. |
| $G^{(r)}, H^{(r)}$ | Gradient vectors of the $r$th tree. |
| $\Omega^{(r)}$ | Weight vector of the $r$th tree. |
| $I^{(r,t)}, \Delta^{(r,t)}$ | The sample space and its vector representation. |
| $Q, \pi$ | Query data and query result. |
| $as^{(r,t)}, ai^{(r,t)}$ | Query sign and index of the $r$th tree's $t$th split. |
| $ps^{(r,t)}, pi^{(r,t)}$ | Perturbed query sign and index. |
| $\langle x \rangle_1, \langle x \rangle_2$ | Shared parts of $x$. |
| $[\![x]\!], [x]$ | FE and SHE ciphertexts of $x$. |

### A. Description of Model Training

The model training process in SGBoost mainly consists of four phases: 1) System initialization, 2) Data bucketing and sharing, 3) Iterative tree boosting, and 4) Training completed, as shown in Fig. 3.

*1) System Initialization:* In this phase, all participants initialize the system via parameter agreement and key generation.

• **Step 1: Parameter Agreement.**

In this step, all $K$ participants first agree to perform the stipulated protocol and are willing to collaboratively train a global boosting tree model over their local training data. Then, with an entity alignment protocol [36], participants delete the unrelated data samples and permute the common data samples accordingly, where the number of remaining data samples is written as $N$. After that, initiator $\mathrm{P}_K$ randomly initializes the model weight $\mathbf{\Omega}^{(0)} = (\omega_1^{(0)}, \cdots, \omega_N^{(0)})$. Finally, participants agree on the hyperparameters, containing FE security parameter $\kappa$, SHE security parameters $(k_0, k_1, k_2)$, loss function $l$, maximum tree depth $T$, maximum tree number $R$, regularization parameters $\lambda$ and $\gamma$, precision parameter $\epsilon$, and learning rate $\eta$.

• **Step 2: Key Generation.**

In this step, initiator $\mathrm{P}_K$ executes $\mathrm{FE}.\mathrm{Gen}(\kappa, N)$ to generate the master key pair $(mpk, msk)$, and executes $\mathrm{SHE}.\mathrm{Gen}(k_0, k_1, k_2)$ to generate SHE symmetric key $ssk$. Then, by encrypting value 0 twice with $ssk$, $\mathrm{P}_K$ calculates the SHE encryption parameter $\sigma_k$ for every passive participant $\mathrm{P}_k$, which is written as $\sigma_k = ([0]_0, [0]_1)$. Finally, $\mathrm{P}_K$ sends $msk$ and corresponding $\sigma_k$ to every passive participant $\mathrm{P}_k$ over secure channels.

*2) Data Bucketing and Sharing:* In this phase, passive participants perform bucketing for their local training data and generate the boolean bucketing matrices, which are secretly shared with $\mathrm{P}_K$ and CSP, respectively.

• **Step 1: Data Bucketing.**

After entity alignment, $\mathrm{P}_k$'s local training data $\mathcal{D}^{(k)} \in \mathbb{R}^{N \times D_k}$ can be represented as

$$\mathcal{D}^{(k)} = \{X_1^{(k)}, \cdots, X_{D_k}^{(k)}\},$$

where $D_k$ is the feature number of $\mathcal{D}^{(k)}$, $X_d^{(k)} = (x_{1,d}^{(k)}, \cdots, x_{N,d}^{(k)})$ is an $N$-dimension vector denoting the data values of the $d$th feature ($d = 1, \cdots, D_k$), and $D = \sum_{k=1}^{K} D_k$ is the total feature number. Moreover, initiator $\mathrm{P}_K$ has the label vector $Y = (y_1, \cdots, y_N)$. After that, the complete training

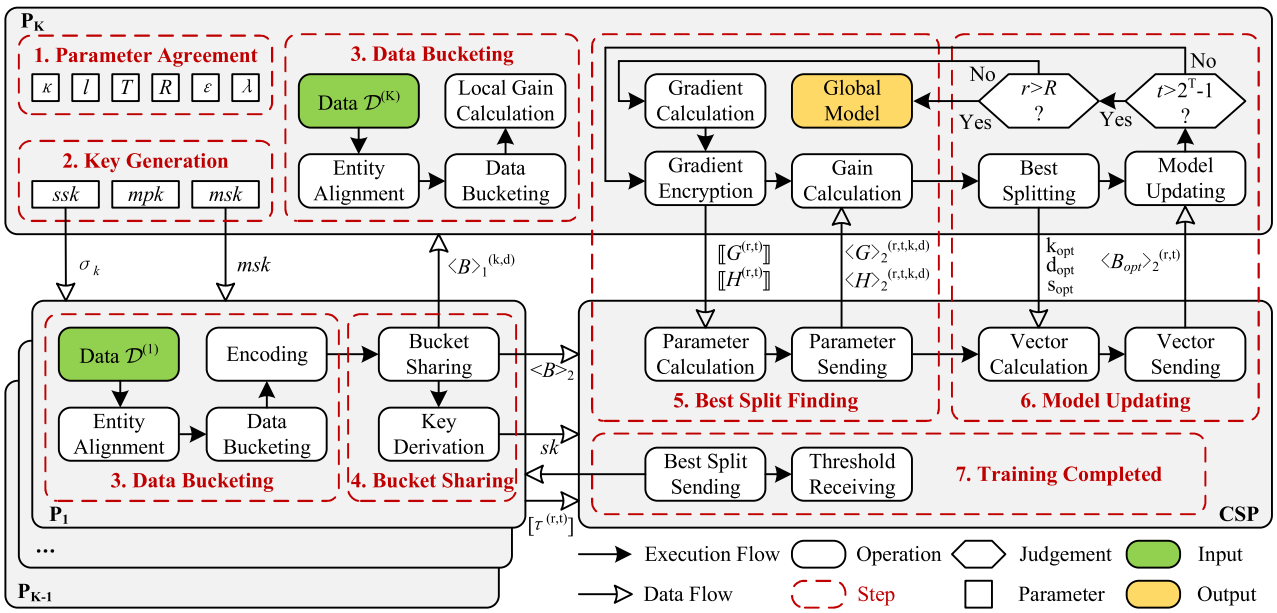



Fig. 3. The overview of model training in SGBoost.

data can be represented as $\mathcal{D} = \{\mathcal{D}^{(1)}, \cdots, \mathcal{D}^{(K)}, Y\}$, which are vertically partitioned over all participants.

In this step, every passive participant $\mathrm{P}_k$ first independently determines its bucket number $S_k$. Since the global bucketing strategy is adopted in SGBoost, the bucket number $S_k$ should be set appropriately more. Then, $\mathrm{P}_k$ can obtain the bucketing threshold vector $T^{(k,d)} = (\tau_1^{(k,d)}, \cdots, \tau_{S_k+1}^{(k,d)})$ of its $d$th feature (contains the maximum and minimum thresholds). After that, for the $d$th feature, $\mathrm{P}_k$ generates a boolean bucket matrix $\mathcal{M}^{(k,d)} \in \mathbb{B}^{S_k \times N}$ as

$$\mathcal{M}^{(k,d)} = \{B_1^{(k,d)}, \cdots, B_{S_k}^{(k,d)}\}^T.$$

Each bit $b_{s,n}^{(k,d)} \in B_s^{(k,d)}$ is calculated as

$$b_{s,n}^{(k,d)} = \begin{cases} 1, & (\tau_s^{(k,d)} < x_{n,d}^{(k)} \leq \tau_{s+1}^{(k,d)}) \\ 0, & (else) \end{cases},$$

where $d = 1, \cdots, D_k$, $s = 1, \cdots, S_k$, and $n = 1, \cdots, N$.

• **Step 2: Bucket Sharing.**

For each bucket matrix $\mathcal{M}^{(k,d)}$, passive participant $\mathrm{P}_k$ first generates a shared boolean matrix $\langle\mathcal{M}\rangle_1^{(k,d)} \in \mathbb{B}^{S_k \times N}$, where bit $\langle b_{s,n}\rangle_1^{(k,d)} \in \langle\mathcal{M}\rangle_1^{(k,d)}$ is set as

$$\langle b_{s,n}\rangle_1^{(k,d)} = \begin{cases} 0, & (b_{s,n}^{(k,d)} = 1) \\ 1, & (b_{s,n}^{(k,d)} = 0, p = \frac{1}{2}) \\ 0, & (b_{s,n}^{(k,d)} = 0, p = \frac{1}{2}) \end{cases}$$

Then, $\mathrm{P}_k$ calculates the shared matrix $\langle\mathcal{M}\rangle_2^{(k,d)}$ as

$$\langle\mathcal{M}\rangle_2^{(k,d)} = \langle\mathcal{M}\rangle_1^{(k,d)} \oplus (1 - \mathcal{M}^{(k,d)}),$$

where $\oplus$ denotes the component-wise XOR calculation.

After that, the bucketing matrix $\mathcal{M}^{(k,d)}$ is divided into $\langle\mathcal{M}\rangle_1^{(k,d)}$ and $\langle\mathcal{M}\rangle_2^{(k,d)}$, which are sent to $\mathrm{P}_K$ and CSP, respectively. Finally, $\mathrm{P}_k$ generates the secret key $sk_s^{(k,d)}$ for each shared row $\langle B_s\rangle_2^{(k,d)} \in \langle\mathcal{M}\rangle_2^{(k,d)} (s = 1, \cdots, S_k)$ through executing $\mathrm{FE}.\mathrm{Der}(msk, \langle B_s\rangle_2^{(k,d)})$, which is also sent to CSP.

*3) Iterative Tree Boosting:* In this phase, the global model is constructed by multi-round interactions between CSP and initiator $\mathrm{P}_K$ until the number of trees reaches $R$.

When training the $r$th tree, $\mathrm{P}_K$ initializes an $N$-dimension bit vector $\Delta^{(r,1)} = (\delta_1^{(r,1)}, \cdots, \delta_N^{(r,1)})$ to represent the sample space $I^{(r,1)}$, and $\delta_n^{(r,t)} = 1$ means that the $n$th data sample is used for the $t$th split finding of $r$th tree. Then, based on weights $(\Omega^{(0)}, \cdots, \Omega^{(r-1)})$ and data label $Y$, initiator $\mathrm{P}_K$ calculates the gradients $G^{(r)} = (g_1^{(r)}, \cdots, g_N^{(r)})$ and $H^{(r)} = (h_1^{(r)}, \cdots, h_N^{(r)})$. After that, $\mathrm{P}_K$ continuously splits the data samples until the depth of the $r$th tree reaches $T$, which can be further divided into two steps of best split finding and model updating.

• **Step 1: Best Split Finding.**

In this step, initiator $\mathrm{P}_K$ finds the $t$th $(t = 1, \cdots, 2^T - 1)$ best split as follows (Algorithm 1).

*a) Gradient Encryption:* $\mathrm{P}_K$ first calculates $ag^{(r,t)} = G^{(r)} \cdot \Delta^{(r,t)}$ and $ah^{(r,t)} = H^{(r)} \cdot \Delta^{(r,t)}$. Then, $\mathrm{P}_K$ encrypts gradients $G$ and $H$ through executing $\mathrm{FE}.\mathrm{Enc}(mpk, G^{(r)} \circ \Delta^{(r,t)})$ and $\mathrm{FE}.\mathrm{Enc}(mpk, H^{(r)} \circ \Delta^{(r,t)})$,[1] where $\circ$ denotes the component-wise multiplication. Finally, $[\![G^{(r,t)}]\!]$ and $[\![H^{(r,t)}]\!]$ are sent to CSP.

*b) Shared Parameter Calculation:* For the $d$th data feature of each participant $\mathrm{P}_k (k \neq K)$, $\mathrm{P}_K$ calculates its shared parameters $\langle G\rangle_1^{(r,t,k,d)}$ and $\langle H\rangle_1^{(r,t,k,d)}$, where $\langle g_s\rangle_1^{(r,t,k,d)} \in \langle G\rangle_1^{(r,t,k,d)}$ and $\langle h_s\rangle_1^{(r,t,k,d)} \in \langle H\rangle_1^{(r,t,k,d)}$ are

$$\begin{cases} \langle g_s\rangle_1^{(r,t,k,d)} = G \circ \Delta^{(r,t)} \cdot \langle B_s\rangle_1^{(k,d)} \\ \langle h_s\rangle_1^{(r,t,k,d)} = H \circ \Delta^{(r,t)} \cdot \langle B_s\rangle_1^{(k,d)} \end{cases}.$$

[1]Since the inner-product FE is based on the integer group, elements in $G^{(r,t)}$ and $H^{(r,t)}$ will be multiplied by $\epsilon$ and rounded to integers before encryption, and factor $\epsilon$ will be divided after decryption.

Similarly, CSP calculates the shared parameters $\langle g_s\rangle_2^{(r,t,k,d)} \in \langle G\rangle_2^{(r,t,k,d)}$ and $\langle h_s\rangle_2^{(r,t,k,d)} \in \langle H\rangle_2^{(r,t,k,d)} (s = 1, \cdots, S_k)$ as

$$\begin{cases} \langle g_s\rangle_2^{(r,t,k,d)} = \text{FE}.\text{Dec}(\llbracket G^{(r,t)}\rrbracket, \langle B_s\rangle_2^{(k,d)}, sk_s^{(k,d)}) \\ \langle h_s\rangle_2^{(r,t,k,d)} = \text{FE}.\text{Dec}(\llbracket H^{(r,t)}\rrbracket, \langle B_s\rangle_2^{(k,d)}, sk_s^{(k,d)}) \end{cases}, \quad (2)$$

where $k = 1, \cdots, K-1$, $d = 1, \cdots, D_k$, and $s = 1, \cdots, S_k$.

*c) Intermediate Parameter Calculation:* After receiving $\langle G\rangle_2^{(r,t,k,d)}$ and $\langle H\rangle_2^{(r,t,k,d)}$ from CSP, $\text{P}_K$ sets $lg_0^{(r,t,k,d)} = 0$ and $lh_0^{(r,t,k,d)} = 0$, and calculates intermediate parameters $lg_s^{(r,t,k,d)}$ and $lh_s^{(r,t,k,d)}$ of each $s$th bucket as

$$\begin{cases} lg_s^{(r,t,k,d)} = lg_{s-1}^{(r,t,k,d)} + ag^{(r,t)} - \langle g_s\rangle_1^{(r,t,k,d)} - \langle g_s\rangle_2^{(r,t,k,d)} \\ lh_s^{(r,t,k,d)} = lh_{s-1}^{(r,t,k,d)} + ah^{(r,t)} - \langle h_s\rangle_1^{(r,t,k,d)} - \langle h_s\rangle_2^{(r,t,k,d)} \end{cases}.$$

*d) Gain Calculation:* $\text{P}_K$ calculates every gain $\mathcal{G}_s^{(r,t,k,d)}$ as

$$\frac{1}{2}\left(\frac{(lg_s^{(r,t,k,d)})^2}{lh_s^{(r,t,k,d)}+\lambda} + \frac{(ag^{(r,t)} - lg_s^{(r,t,k,d)})^2}{ah^{(r,t)} - lh_s^{(r,t,k,d)}+\lambda} - \frac{(ag^{(r,t)})^2}{ah^{(r,t)}}\right) - \gamma, \quad (3)$$

and finds the maximum value of them (containing its local gains), which is the best gain written as $\mathcal{G}_{s_{opt}}^{(r,t,k_{opt},d_{opt})}$, where $k = 1, \cdots, K-1$, $d = 1, \cdots, D_k$, and $s = 1, \cdots, S_k$.

**Algorithm 1** Secure Best Split Finding

**Input**($\text{P}_K$): $\langle\mathcal{M}\rangle_1^{(k,d)}$, $G$, $H$, $\Delta^{(r,t)}$, $mpk$
**Input**(CSP): $\langle\mathcal{M}\rangle_2^{(k,d)}$, $sk_s^{(k,d)}$
**Output**(CSP): $\mathcal{G}_{s_{opt}}^{(r,t,k_{opt},d_{opt})}$

▷ *Gradient encryption* ◁
$\text{P}_K$ encrypts $G$ and $H$ with $mpk$.
$\text{P}_K$ sends $\llbracket G^{(r,t)}\rrbracket$ and $\llbracket H^{(r,t)}\rrbracket$ to CSP.
**for** $k = 1, \cdots, K-1, d = 1, \cdots, D_k$ **do**
  ▷ *Shared parameter calculation* ◁
  $\text{P}_K$ calculates $\langle G\rangle_1^{(r,t,k,d)}$, $\langle H\rangle_1^{(r,t,k,d)}$ with $\langle\mathcal{M}\rangle_1^{(k,d)}$.
  CSP calculates $\langle G\rangle_2^{(r,t,k,d)}$,$\langle H\rangle_2^{(r,t,k,d)}$ with $\langle\mathcal{M}\rangle_2^{(k,d)}$.
  CSP sends $\langle G\rangle_2^{(r,t,k,d)}$ and $\langle H\rangle_2^{(r,t,k,d)}$ to $\text{P}_K$.
  **for** $s = 1, \cdots, S_k$ **do**
    ▷ *Intermediate parameter calculation* ◁
    $\text{P}_K$ calculates $lg_s^{(r,t,k,d)}$ and $lh_s^{(r,t,k,d)}$.
    ▷ *Gain calculation* ◁
    $\text{P}_K$ calculates $\mathcal{G}_s^{(r,t,k,d)}$.
$\text{P}_K$ obtains the best gain $\mathcal{G}_{s_{opt}}^{(r,t,k_{opt},d_{opt})}$.

• **Step 2: Model Updating**

After obtaining the best gain, CSP calculates shared vector $\langle B_{opt}\rangle_2^{(r,t)} = \sum_{s=1}^{s_{opt}} \langle B_s\rangle_2^{(k_{opt},d_{opt})}$ and sends it to $\text{P}_K$.

After that, $\text{P}_K$ updates the bit vectors $\Delta^{(r,2t)}$ and $\Delta^{(r,2t+1)}$ as

$$\begin{cases} B_{opt}^{(r,t)} = \sum_{s=1}^{s_{opt}} (1 - \langle B_s\rangle_1^{(k_{opt},d_{opt})}) - \langle B_{opt}\rangle_2^{(r,t)} \\ \Delta^{(r,2t)} = \Delta^{(r,t)} \circ B_{opt}^{(r,t)} \\ \Delta^{(r,2t+1)} = \Delta^{(r,t)} \circ (1 - B_{opt}^{(r,t)}) \end{cases}.$$

Finally, if the tree depth reaches $T$, $\text{P}_K$ calculates the weight $\Omega^{(r)} = (\omega_1^{(0)}, \cdots, \omega_N^{(0)})$ as

$$\begin{cases} \omega_n^{(r)}(n \in I^{(r,2t)}) = -\dfrac{\Delta^{(r,2t)} \cdot G^{(r)}}{\Delta^{(r,2t)} \cdot H^{(r)} + \lambda} \\ \omega_n^{(r)}(n \in I^{(r,2t+1)}) = -\dfrac{\Delta^{(r,2t+1)} \cdot G^{(r)}}{\Delta^{(r,2t+1)} \cdot H^{(r)} + \lambda} \end{cases}.$$

Otherwise, $\text{P}_K$ launches the $2t$th and $(2t+1)$th split findings.

*4) Training Completed:* When the tree number reaches $R$, the iterative tree boosting terminates, and initiator $\text{P}_K$ obtains the best tree structure and weights. Since the thresholds of best splits are necessary for providing query services, which are not shared in the model training process, all participants outsource the ciphertext thresholds to CSP in this step.

Specifically, initiator $\text{P}_K$ sends every best split $(r, t, k_{opt}, d_{opt}, s_{opt})$ to corresponding passive participant $\text{P}_{k_{opt}}$.

Then, for each best split $(r, t, k_{opt}, d_{opt}, s_{opt})$, $\text{P}_{k_{opt}}$ encrypts the threshold $\tau_{s_{opt}}^{(k_{opt},d_{opt})}$ with its encryption parameter $\sigma_{k_{opt}}$ as

$$[\tau^{(r,t)}] = \tau_{s_{opt}}^{(k_{opt},d_{opt})} + (r_0 * [0]_0) + (r_1 * [0]_1), \quad (4)$$

where $r_0$ and $r_1$ are two random numbers. Similarly, initiator $\text{P}_K$ encryptes its best split threshold $\tau_{s_{opt}}^{(K,d_{opt})}$ with $ssk$.

Finally, all participants send the encrypted best split thresholds to CSP, and CSP obtains $[\tau^{(r,t)}]$s.

### *B. Description of Model Query*

In this section, we introduce the model query process of SGBoost, which contains three steps of query request, oblivious query, and query response.

After model training, CSP obtains all best splits, which can be re-ordered and represented as $(r, t, d_{opt})$s ($d_{opt} \in [1, D]$). Moreover, all ciphertext best split thresholds $[\tau^{(r,t)}]$s are outsourced to CSP for providing query service.

• **Step 1: Query Request**

In this step, for the registered query user U, initiator $\text{P}_K$ first generates the encryption parameter $\sigma' = ([0]_0, [0]_1, [-1])$ with $ssk$, which is sent to U and CSP. Then, $\text{P}_K$ generates a SHE symmetric key $ssk'$ as the query key, and sends it to U.

After receiving $\sigma'$, U encrypts its query data $Q = (q_1, \cdots, q_D)$ as in (4), and sends $[Q]$ to CSP.

• **Step 2: Oblivious Query**

In this step, the query is executed obliviously between CSP and $\text{P}_K$, and CSP returns the ciphertext query response to U (Algorithm 2). The detailed process is as follows.

*a) Query Sign Calculation:* On receiving $[Q]$, by comparing the query data value and best split threshold over ciphertexts, CSP obtains the query sign as $[as^{(r,t)}] = [q_{d_{opt}}] - [\tau^{(r,t)}] (r = 1, \cdots, R, t = 1, \cdots, 2^T - 1)$.

*b) Query Sign Perturbation:* For each query sign $[as^{(r,t)}]$, CSP first generates two signed random numbers $r_0^{(r,t)}$ and $r_1^{(r,t)}$ satisfying $|r_0^{(r,t)}| > |r_1^{(r,t)}|$. Then, CSP perturbs $[as^{(r,t)}]$ as

$$[ps^{(r,t)}] = \begin{cases} r_0^{(r,t)} * [as^{(r,t)}] + r_1^{(r,t)}, & (r_0^{(r,t)} > 0) \\ |r_0^{(r,t)}| * [-1] * [as^{(r,t)}] + r_1^{(r,t)}, & (r_0^{(r,t)} < 0), \end{cases} \quad (5)$$

The perturbed query signs $[ps^{(r,t)}]$s are sent to $\text{P}_K$.

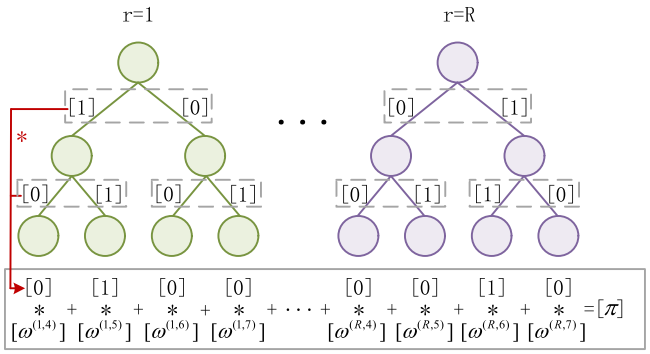



Fig. 4. An example of query response calculation ($T = 2$).

*c) Query Index Encoding:* $\mathrm{P}_K$ decrypts these perturbed query signs with $ssk$ and obtains $ps^{(r,t)}$s. $\mathrm{P}_K$ encodes every perturbed query index $[pi^{(r,t)}]$ as

$$[pi^{(r,t)}] = \begin{cases} ([0],[1]), & (ps^{(r,t)} \geq 0) \\ ([1],[0]), & (ps^{(r,t)} < 0), \end{cases} \tag{6}$$

where [0] and [1] are SHE ciphertexts encrypted with $ssk'$. Similarly, model weights are also encrypted with $ssk'$. $\mathrm{P}_K$ sends the perturbed query indexes and model weights to CSP.

*d) Query Index Recovering:* CSP recovers every perturbed query index $[pi^{(r,t)}]$ to its true index $[ai^{(r,t)}]$ according to

$$[ai^{(r,t)}] = \begin{cases} [pi^{(r,t)}], & (r_0^{(r,t)} > 0) \\ \mathrm{reverse}([pi^{(r,t)}]), & (r_0^{(r,t)} < 0), \end{cases} \tag{7}$$

where $\mathrm{reverse}([pi^{(r,t)}])$ denotes reversing the order of two ciphertexts in $[pi^{(r,t)}]$.

*e) Query Response Calculation:* As shown in Fig. 4, for each leaf node of the $r$th tree, CSP first calculates a ciphertext bit $[\rho^{(r,t)}]$ by multiplying its ciphertext indexes from root, where $t = 2^T, \cdots, 2^{T+1} - 1$. Then, CSP calculates the ciphertext response $[\pi]$ as

$$[\pi] = \sum_{r=1}^{R} \sum_{t=2^T}^{2^{T+1}-1} [\omega^{(r,t)}] * [\rho^{(r,t)}].$$

**• Step 3: Query Response**

CSP returns query response $[\pi]$ to user U. U decrypts $[\pi]$ with $ssk'$ and obtains its query result $\pi$.

*Remark* To prevent the query users from extracting model parameters from the query results, CSP could add a slight noise to the ciphertext query response $[\pi]$, or limit the query times of a single user.

### C. Correctness Analysis

In this section, the correctness of model training and query processes is proved, respectively.

*1) Correctness of Model Training:* To ensure the correctness of model training, we need to prove that the gains are calculated correctly during each split finding, and the model is updated correctly by the best splits.

*Theorem 1:* During the $t$th split finding of the $r$th tree, $\mathcal{G}_s^{(r,t,k,d)}$ is the correct gain of $d$th feature's $s$th bucket.

**Algorithm 2** Oblivious Query

**Input**(CSP): $[\Omega^{(r)}]s$, $[Q]$, $[\tau^{(r,t)}]s$, $(r, t, d_{opt})s$
**Input**($\mathrm{P}_K$): $ssk, ssk'$
**Output**(CSP): $[\pi]$

▷*Query sign calculation* ◁
CSP calculates query signs $[as^{(r,t)}]$s with $[\tau^{(r,t)}]s$ and $[Q]$.
▷*Query sign perturbation* ◁
CSP perturbs the query signs as $[ps^{(r,t)}]$s.
CSP sends $[ps^{(r,t)}]$s to $\mathrm{P}_K$.
▷*Query index encoding* ◁
$\mathrm{P}_K$ decrypts $[ps^{(r,t)}]$s.
$\mathrm{P}_K$ encodes the perturbed indexes $[pi^{(r,t)}]$s with $ssk'$.
$\mathrm{P}_K$ sends $[pi^{(r,t)}]$s and $[\Omega^{(r)}]$s to CSP.
▷*Query index recovering* ◁
CSP recovers the query indexes as $[ai^{(r,t)}]$s.
▷*Query response calculation* ◁
CSP calculates response $[\pi]$ with $[\Omega^{(r)}]$s and $[ai^{(r,t)}]$s.

*Proof:* First, the shared parameters $\langle g_s \rangle_1^{(r,t,k,d)}$ and $\langle g_s \rangle_2^{(r,t,k,d)}$ can be deduced as

$$\begin{cases} \langle g_s \rangle_1^{(r,t,k,d)} = \sum_{n=1}^{N} g_n * \delta_n^{(r,t)} * \langle b_{s,n} \rangle_1^{(k,d)} \\ \langle g_s \rangle_2^{(r,t,k,d)} = \sum_{n=1}^{N} g_n * \delta_n^{(r,t)} * \langle b_{s,n} \rangle_2^{(k,d)} \end{cases}.$$

Then, since the bit vector $\Delta^{(r,t)}$ denotes the sample space of the $t$th split finding, we can compute

$$\begin{aligned} & ag^{(r,t)} - \langle g_s \rangle_1^{(r,t,k,d)} - \langle g_s \rangle_2^{(r,t,k,d)} \\ &= \sum_{n=1}^{N} g_n * \delta_n^{(r,t)} * (1 - \langle b_{s,n} \rangle_1^{(k,d)} - \langle b_{s,n} \rangle_2^{(k,d)}) \\ &= \sum\nolimits_{n \in I^{(r,t)}} g_n * \delta_n^{(r,t)} * b_{s,n}{}^{(k,d)} \\ &= \sum\nolimits_{n \in I^{(r,t)}, \tau_s^{(k,d)} < x_{n,d}^{(k)} \leq \tau_{s+1}^{(k,d)}} g_n. \end{aligned}$$

After that, $lg_s^{(r,t,k,d)}$ can be deduced as

$$\begin{aligned} lg_s^{(r,t,k,d)} &= \sum_{i=1}^{s} \left( \sum\nolimits_{n \in I^{(r,t)}, \tau_i^{(k,d)} < x_{n,d}^{(k)} \leq \tau_{i+1}^{(k,d)}} g_n \right) \\ &= \sum\nolimits_{n \in I_L^{(r,t,k,d,s)}} g_n. \end{aligned}$$

Similarly, we can prove $lh_s^{(r,t,k,d)} = \sum_{n \in I_L^{(r,t,k,d,s)}} h_n$.

Therefore, it is obvious that (3) is equivalent to (1), i.e., gain $\mathcal{G}_s^{(r,t,k,d)}$ is correctly calculated. ■

*Theorem 2:* During the $t$th split finding of the $r$th tree, $\Delta^{(r,t)}$ is updated correctly by the best split $(r, t, k_{opt}, d_{opt}, s_{opt})$.

*Proof:* First, given $(k_{opt}, d_{opt}, s_{opt})$, $B_{opt}^{(r,t)}$ is deduced as

$$B_{opt}^{(r,t)} = \sum_{s=1}^{s_{opt}} (1 - \langle B_s \rangle_1^{(k_{opt},d_{opt})} - \langle B_s \rangle_2^{(k_{opt},d_{opt})})$$

$$= \sum_{s=1}^{s_{opt}} B_s^{(k_{opt},d_{opt})}.$$

Then, we can obtain

$$\delta_n^{(r,2t)} = \sum_{s=1}^{s_{opt}} \delta_n^{(r,t)} \cdot b_{s,n}^{(k_{opt},d_{opt})} = \begin{cases} 1, & (n \in I_L^{(r,t,k_{opt},d_{opt},s_{opt})}) \\ 0, & (else), \end{cases}$$

i.e., $\Delta^{(r,2t)}$ contains the data samples of the current best split's left node, which are used for the $2t$th split finding. Similarly, $\Delta^{(r,2t+1)}$ contains the right data samples using for the $(2t+1)$th split. Therefore, $\Delta^{(r,t)}$ is updated correctly by the best split. ∎

*2) Correctness of Model Query:* Here we prove that the query result is calculated correctly.

*Theorem 3:* $\pi$ is the correct query result of $Q$ querying the trained model.

*Proof:* According to (6) and (7), we can obtain

$$[ai^{(r,t)}] = \begin{cases} ([0],[1]), & (ps^{(r,t)} * r_0^{(r,t)} \geq 0) \\ ([1],[0]), & (ps^{(r,t)} * r_0^{(r,t)} < 0). \end{cases}$$

Since $|r_0^{(r,t)}| > |r_1^{(r,t)}|$, $r_1^{(r,t)}$ does not affect the sign of $r_0^{(r,t)} * as^{(r,t)}$, and $as^{(r,t)}$ has the same sign with $ps^{(r,t)} * r_0^{(r,t)}$. Then, from $as^{(r,t)} = q_{d_{opt}} - \tau^{(r,t)}$, we obtain

$$[ai^{(r,t)}] = \begin{cases} ([0],[1]), & (q_{d_{opt}} \geq \tau^{(r,t)}) \\ ([1],[0]), & (q_{d_{opt}} < \tau^{(r,t)}). \end{cases}$$

Obviously, only the indexes on the correct query path are all [1]s and the multiplication $\rho^{(r,t)}$ is [1]. Finally, the query response can be represented as

$$[\pi] = \sum_{r=1}^{R} \sum_{t=2^T}^{2^{T+1}-1} [\omega^{(r,t)}] * [\rho^{(r,t)}] = [\sum_{r=1}^{R} f_r(Q)],$$

which is the same with the prediction equation shown in Section III-A.

Therefore, $\pi$ is the correct query result of $Q$ querying the trained model. ∎

## VI. Security Analysis

In this section, under the threat model defined in Section II, we analyze the security in model training and query processes.

### A. Security of Model Training

The security in model training process involves the privacy of raw training data, bucket information, and split information.

*1) Raw Training Data:* First, in SGBoost, participants only use their data values to locally execute data bucketing, which do not involve privacy.

Then, during each best split finding, the gradients $G$ and $H$ (the superscripts are omitted for simplicity) are calculated by $\mathrm{P}_K$ based on its data labels, which are encrypted by FE and sent to CSP. As proved in [31], the functional encryption is selective security against chosen-plaintext attacks (s-IND-CPA). So the plaintext gradients cannot be obtained by CSP for inferring the data labels.

Meanwhile, $\langle G \rangle_2$ and $\langle H \rangle_2$ can be obtained by CSP, which are the inner products of the gradients and its shared bit vector $\langle B_s \rangle_2$. For ensuring the privacy of data labels, we only need to prove that CSP cannot obtain gradient vector $G$ based on the matrix $\langle \mathcal{M} \rangle_2$ and vector $\langle G \rangle_2$. The proof of $H$ is the same and is omitted.

*Theorem 4:* CSP cannot obtain $G$ based on $\langle G \rangle_2$ and $\langle \mathcal{M} \rangle_2$.

*Proof:* First, according to (2), $\langle G \rangle_2$ can be represented as

$$\langle G \rangle_2 = G \cdot [\langle \mathcal{M} \rangle_2]^T, \tag{8}$$

where $\langle \mathcal{M} \rangle_2$ is a $(S_k \times N)$-dimension random bit matrix. For obtaining $G$, CSP needs to solve the linear equation (8). According to the conditions for solving linear equations, when $N > S_k$, it has infinite solutions. In SGBoost, since the number of data samples is much larger than that of buckets, i.e., $N \gg S_k$, $G$ has infinite solutions. Therefore, CSP cannot obtain $G$. ∎

Moreover, CSP can obtain the weights $\omega$s but cannot obtain the splits $\Delta$s, so these weights cannot correspond to the data samples and do not have a threat to the data labels.

In summary, the raw training data (containing data values and labels) are protected well in SGBoost.

*2) Bucket Information:* In SGBoost, based on our bucket sharing algorithm, the bucket matrix $\mathcal{M}$ is divided into two matrices $\langle \mathcal{M} \rangle_1$ and $\langle \mathcal{M} \rangle_2$, which are sent to $\mathrm{P}_K$ and CSP. For ensuring the privacy of bucket information, we prove that $\mathrm{P}_K$ cannot infer which bucket a certain data sample $x_n$ exists in based on $\langle \mathcal{M} \rangle_2$, and the proof of $\langle \mathcal{M} \rangle_1$ is the same and is omitted.

*Theorem 5:* $\mathrm{P}_K$ cannot obtain which bucket a certain data sample $x_n$ exists in based on $\langle \mathcal{M} \rangle_2$.

*Proof:* First, we formally the proof as

$$\Pr_{\mathrm{SGBoost}}[s \in \mathbb{Z}_S : b_{s,n} = 1] - \Pr_{\mathrm{Random}}[s \in \mathbb{Z}_S : b_{s,n} = 1] \leq \theta,$$

where $\theta$ is a small probability. Based on $\langle \mathcal{M} \rangle_1$ or $\langle \mathcal{M} \rangle_2$, we calculates the correct inference probability from SGBoost as

$$\Pr_{\mathrm{SGBoost}}[s \in \mathbb{Z}_S : b_{s,n} = 1] = \sum_{i=0}^{S-1} C_{S-1}^{i} \frac{1}{2^{S-1}} \frac{1}{S-i},$$

where $C$ denotes the combinatorial number. And the successful probability of random inference is $\Pr_{\mathrm{Random}}[s \in \mathbb{Z}_S : b_{s,n} = 1] = \frac{1}{S}$. Here assume the bucket number $S = 32$, then we can calculate $\Pr_{\mathrm{SGBoost}}[s \in \mathbb{Z}_S : b_{s,n} = 1] - \Pr_{\mathrm{Random}}[s \in \mathbb{Z}_S : b_{s,n} = 1] = 0.0312$, which is a small probability. Therefore, the inference based on $\langle \mathcal{M} \rangle_2$ is almost equivalent to random guessing, and $\mathrm{P}_K$ cannot obtain which bucket a certain data sample $x_n$ exists in. ∎

*3) Split Information:* Unlike [10] and [11], which directly share the plaintext split information, SGBoost embeds the split information to gradients and encrypts the gradients for every best split finding. According to the s-IND-CPA of functional encryption, CSP cannot obtain the true values of gradients, so it cannot infer the split information by judging whether a certain gradient is zero.

### B. Security of Model Query

The security in model query process involves the privacy of query data, query result, and weight information.

*1) Query Data:* During model query, users' query data are encrypted with $ssk$, which is only mastered by initiator $P_K$, and the query data are sent to CSP. According to the IND-CPA of SHE [34], which is equivalent to solving $(\mathcal{L}, p)$-based decision hard problem to distinguish SHE ciphertexts, the query data of users are protected well.

*2) Query Result:* For inferring the query results, CSP or $P_K$ needs to obtain the query signs and classify the query data into the corresponding nodes for every tree. So we only need to prove that these query signs cannot be obtained by CSP or $P_K$.

*Theorem 6:* CSP cannot obtain query signs $as^{(r,t)}$s.

*Proof:* First, the query signs $[as^{(r,t)}]$s are calculated over SHE cipertexts encrypted with $ssk$, which cannot be decrypted by CSP since only $P_K$ has $ssk$. Then, the indexes $pi^{(r,t)}$s also contain the query signs, e.g., $([0], [1])$ denotes the positive sign, but they are also encypted with SHE key $ssk'$ and cannot be decrypted by CSP. Therefore, according to the IND-CPA of SHE, CSP cannot obtain query signs $as^{(r,t)}$s. ■

*Theorem 7:* $P_K$ cannot obtain query signs $as^{(r,t)}$s.

*Proof:* During the model query, $P_K$ can decrypt the perturbed query signs and obtain $ps^{(r,t)}$s. However, according to (5), $as^{(r,t)}$s are protected by two random numbers $r_0^{(r,t)}$ and $r_1^{(r,t)}$ as $ps^{(r,t)} = r_0^{(r,t)} * as^{(r,t)} + r_1^{(r,t)}$, and $P_K$ cannot obtain no matter the true values or signs of $as^{(r,t)}$s. Therefore, the query signs cannot be obtained by $P_K$. ■

Therefore, due to without the query signs, CSP or $P_K$ cannot infer the query results of users.

*3) Weight Information:* Different from [10] and [11], which send the plaintext weights to passive participants, the weights in SGBoost are encrypted and then outsourced to CSP. Therefore, the query in SGBoost does not leak the weights and the leaf purity.

## VII. Performance Evaluation

In this section, we first introduce the experimental settings. Then, we evaluate and compare the model accuracy of SGBoost with centralized XGBoost [37]. After that, the running time and communication overhead of SGBoost are evaluated and compared with representative secure FL schemes SecureBoost [11] and PIVODL [16], which will be reviewed in Section VIII.

### A. Experimental Setting

In the experiments, we choose the XGBoost model to evaluate the performance of our SGBoost framework. The program of CSP is performed on a DELL Precision 7920 workstation with an Intel(R) Xeon(R) Gold 6226R CPU 2.90 GHz processor, 256 GB RAM, running Ubuntu 20.04, and participants and query users execute programs on Thinkpad P53 machines equipped with an Intel(R) Core(TM) i5-9400H 2.50GHz processor and 24 GB RAM, running Windows 10. These programs[2] are implemented with Java 16.0.1. The communication bandwidth between CSP/initiator and passive participants is set as 10 Mbps, that between CSP and initiator is set as 100 Mbps, and that between the query user and CSP is set as 10 Mbps.

[2]https://github.com/nds2022/SGBoost

TABLE II
Hyperparameters in Performance Evaluation

| Hyperparameter | Range | Default |
|---|---|---|
| $K$ | [2,4,6,8,10] | 4 |
| $T$ | [2,3,4,5,6] | 3 |
| $R$ | [2,3,4,5,6] | 5 |
| $\eta$ | 0.3 | 0.3 |
| $\lambda$ | 1 | 1 |
| $S_k$ | 32 | 32 |
| $\epsilon$ | 100 | 100 |
| $\kappa$ | 512 | 512 |
| $k_0$ $(ssk)$ | 1024 | 1024 |
| $k_0$ $(ssk')$ | [1024, 1408, 1792, 2176, 2560] | 1408 |

We evaluate the performance of SGBoost under different hyperparameter settings containing participant numbers, tree numbers, and tree depths. For simplicity, the bucket numbers of all participants are set to 32. During model training, we set the security parameter $\kappa = 512$, which is consistent with the comparison schemes. Since different SHE multiplication depths are needed under different tree depths for executing model queries, as explained in Section III-C, we set different security parameters of query key $ssk'$. The used hyperparameters of SGBoost are also listed in TABLE II.

Two real-world classification datasets are experimented in SGBoost, where 20% of data samples are randomly selected as the test data, and the data features are evenly divided into participants.

*1) Credit Card [38]:* This dataset records 24 credit scoring features of 30000 users, which aims to predict whether a user will make payment on time.

*2) Bank Marketing [39]:* This dataset records direct marketing campaigns (phone calls) of a Portuguese banking institution, which contains 45211 samples and 17 features. Its classification goal is to predict if the client will subscribe to a term deposit.

In the following performance evaluations, every experiment repeated five times independently, and only the average results are shown. The evaluation results of SecureBoost and PIVODL come from [11] and [16].

### B. Model Accuracy

In this subsection, we evaluate the model accuracy of SGBoost on the test datasets, which is measured by the probability of correct prediction, and make a comparison with the basic centralized XGBoost scheme.

From Fig. 5, first, we can find that the increase of model accuracy is not significant with the tree number or depth growth. Second, we find that the boosting tree model can converge with a small tree depth like 3 or 4. Third, on both two test datasets, we can see that our SGBoost has comparable accuracy with centralized XGBoost, and even can

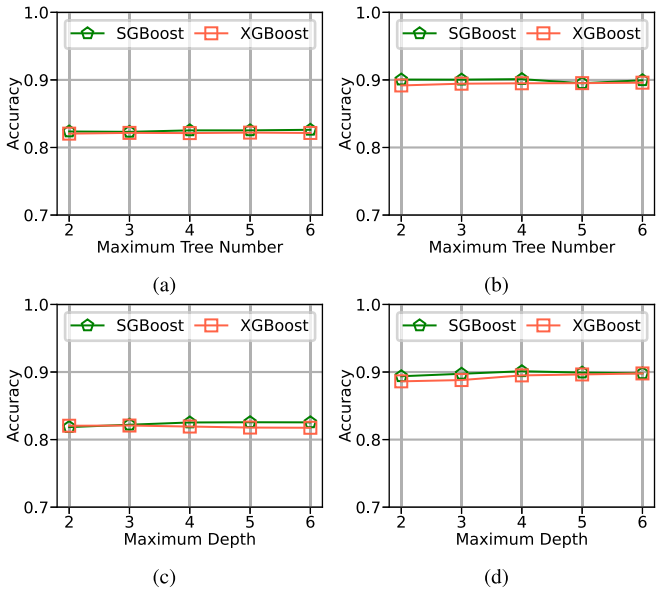



Fig. 5. Model accuracy comparison. (a) Varying different tree numbers over Credit Card dataset. (b) Varying different tree numbers over Bank Marketing dataset. (c) Varying different tree depths over Credit Card dataset. (d) Varying different tree depths over Bank Marketing dataset.

exceed the model accuracy of XGBoost under some settings. This phenomenon may be because the expand-then-round encryption method in SGBoost makes its arithmetic precision slightly lower than XGBoost, and the slightly lower arithmetic precision instead enhances the model generalization ability and reduces the model overfitting. Overall, the experimental results show that our SGBoost does not cause a loss of model accuracy and convergence speed.

### C. Running Time

In this subsection, we will show the total running time of SGBoost's training and query processes, and compare it with schemes PIVODL and SecureBoost. Moreover, we analyze the relationships between the running time and different participant numbers, tree depths, and tree numbers.

*1) Training Time With Different Participant Numbers:* From Fig. 6(a) and Fig. 6(b), we find that, under the same experimental settings, the total training time of SGBoost is significantly lower than that of PIVODL or SecureBoost, since SGBoost does not need frequent interactions between participants. Moreover, since the passive participants only need to securely share their boolean buckets, whose overhead is indeed low, the training time of SGBoost hardly changes with the participant number changes. Therefore, it demonstrates that SGBoost is more suitable for multi-participant training scenes.

*2) Training Time With Different Maximum Tree Numbers:* Fig. 6(c) and Fig. 6(d) show the linear increase relationship between the training time and tree number for all three schemes since every tree boosting costs the same time. Likewise, since a single tree boosting executes quickly, SGBoost costs much less time than PIVDOL or SecureBoost, and its running time increases slower with the tree number growing. Therefore, SGBoost can be used to train large-scale models.

*3) Training Time With Different Maximum Tree Depths:* Fig. 6(e) and Fig. 6(f) plot the training time with different tree

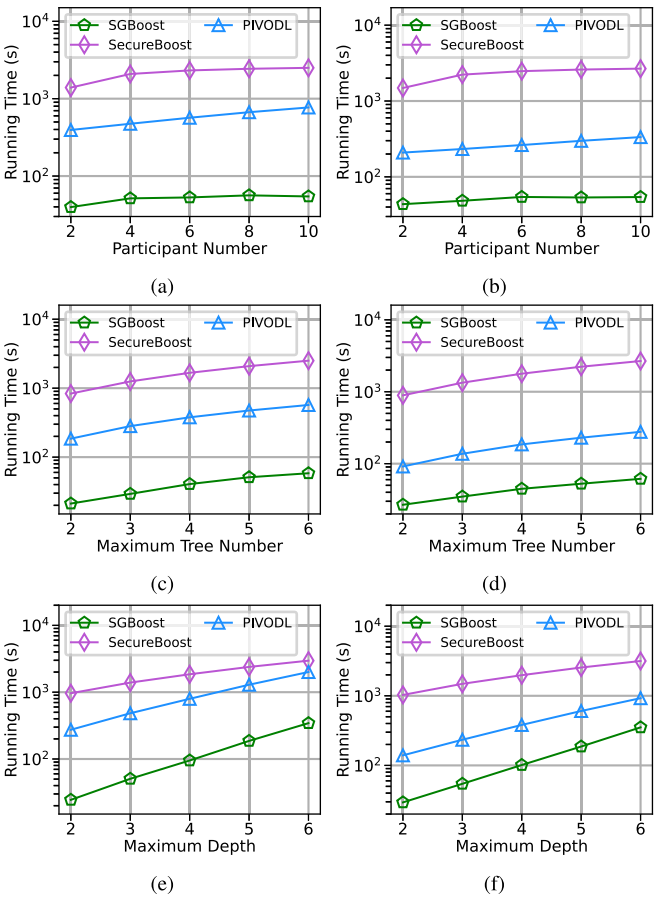



Fig. 6. Total training time comparison. (a) Varying different participant numbers over Credit Card dataset. (b) Varying different participant numbers over Bank Marketing dataset. (c) Varying different tree numbers over Credit Card dataset. (d) Varying different tree numbers over Bank Marketing dataset. (e) Varying different tree depths over Credit Card dataset. (f) Varying different tree depths over Bank Marketing dataset.

depths, and it increases significantly as the tree depth grows, which means that the tree depth is the main factor affecting the training time. Meanwhile, compared with the other two schemes, since SGBoost protects the split information, i.e., it will encrypt the gradients for every split, its training time grows faster when the tree goes deeper. Despite it, benefiting from the efficient algorithm design, SGBoost's training is also faster than PIVODL and SecureBoost, e.g., it only costs 50 seconds to train a 3-depth tree model on the Credit Card dataset, which prompts the privacy-preserving federated training more practical.

*4) Query Time With Different Maximum Tree Numbers and Depths:* From Fig. 7, we can find that the query time increases slowly with the increase of the tree number, but increases rapidly with the tree depth growing. And the query time of two test datasets is almost the same under the same setting, which demonstrates that it has almost nothing to do with the datasets. Moreover, SGBoost only costs about 3 seconds to execute 1000 queries under the default setting, and it means that SGBoost has very low latency for query services.

### D. Communication Overhead

In this subsection, we plot and analyze the communication overhead with different participant numbers, tree depths, and

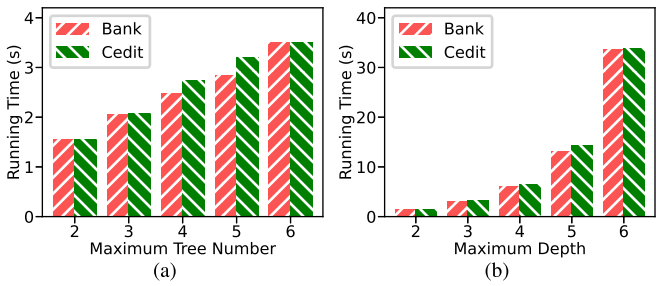



Fig. 7. Total query time per 1000 queries. (a) Varying tree numbers. (b) Varying tree depths.

tree numbers, and also make a comparison with the representative schemes PIVODL and SecureBoost.

*1) Training Communication Overhead With Different Participant Numbers:* From Fig. 8(a) and Fig. 8(b), we can find that the communication overhead of our SGBoost is almost unrelated to the participant number, since passive participants do not need to participate in the interactions after bucket sharing. Since the ciphertext gradients in PIVODL and SecureBoost are required to send to every participant, the communication overhead of PIVODL and SecureBoost increases linearly when the participant number grows. Moreover, the communication overhead of the Bank Marketing datasets is higher than that of Credit Card, which is because its data samples are more. Therefore, SGBoost is more practical and has low requirements for the network bandwidth when the participant number is large.

*2) Training Communication Overhead With Different Maximum Tree Numbers:* The communication overhead increases linearly when the tree number increases for all three schemes, as shown in Fig. 8(c) and Fig. 8(d). Moreover, SGBoost's communication overhead is lower than PIVODL on the Credit Card dataset but higher on the Bank Marketing dataset, which means SGBoost is more suitable for the dataset that has fewer samples and more features.

*3) Training Communication Overhead With Different Maximum Tree Depths:* Fig. 8(e) and Fig. 8(f) show that the communication overhead of SGBoost increases rapidly as the tree goes deeper, which is because the split information is protected in SGBoost and the ciphertext gradients are sent in every split finding. Fortunately, the XGBoost model does not need a lot of tree depth, and the depths 3 and 4 can achieve high accuracy for most datasets, which is shown in [29] and can also be seen in our accuracy evaluation. Moreover, in SGBoost, the vast majority of communication is between the initiator and CSP, whose communication bandwidth is much larger than that between participants.

*4) Query Communication Overhead With Different Maximum Tree Numbers and Depths:* Fig. 9 shows the communication overhead of queries increases more rapidly with tree depth growing than with tree number growing, and also shows that the communication overhead is almost not related to the datasets. Also, the communication overhead of the model query in SGBoost is quite low, and it only costs about 38 KB to achieve a model query with a default tree model.

In summary, SGBoost's training efficiency is higher than the existing schemes under most practical settings, and it is more efficient when the dataset has fewer samples and more features. Meanwhile, the efficiency of SGBoost is almost unrelated to the participant number and can be used for massive participant scenarios. Moreover, while satisfying the strong security, the overhead of the oblivious query is indeed low.

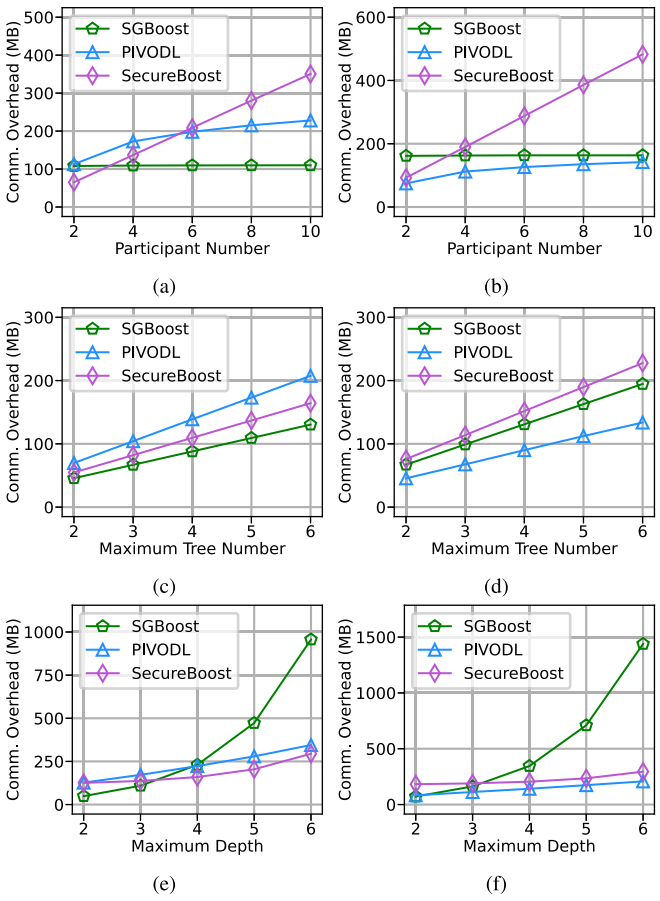



Fig. 8. Training communication overhead comparison. (a) Varying different participant numbers over Credit Card dataset. (b) Varying different participant numbers over Bank Marketing dataset. (c) Varying different tree numbers over Credit Card dataset. (d) Varying different tree numbers over Bank Marketing dataset. (e) Varying different tree depths over Credit Card dataset. (f) Varying different tree depths over Bank Marketing dataset.

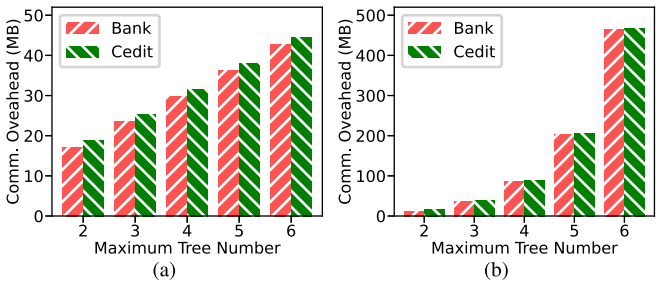



Fig. 9. Communication overhead per 1000 queries. (a) Varying tree numbers. (b) Varying tree depths.

## VIII. Related Work

According to data distribution, Yang et al. [8] classified FL into horizontal FL and vertical FL. In vertical FL, participants' data share a common sample space but a different feature

TABLE III
SECURITY AND FUNCTIONALITY COMPARISON OF VERTICAL FEDERATED TREE BOOSTING SCHEMES

| Schemes | Privacy Preservation | | | | Lossless | Online Unnecessary | Efficiency |
|---|---|---|---|---|---|---|---|
| | Raw data | Bucket | Split | Weight | | | |
| SecureBoost [11] | ✓ | ✓ | ✗ | ✗ | ✓ | ✗ | Less high |
| FederBoost [15] | ✓ | ✓ | ✗ | ✓ | ✗ | ✓ | High |
| PIVODL [16] | ✓ | ✓ | ✗ | ✓ | ✗ | ✗ | Less High |
| Pivot [17] | ✓ | ✓ | ✓ | ✓ | ✓ | ✗ | Low |
| SGBoost | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | High |

space, e.g., a bank and a securities company serving common users. In this section, we review some related works about vertical FL, which based on the targeted model type can be classified into schemes for linear models, non-linear models, and boosting tree models.

### A. Vertical FL Schemes for Linear Models

Since linear models are the simplest models in machine learning, the vertical linear FL schemes were first proposed.

By combining Yao's garbled circuits and tailored protocols, Gascon et al. [40] designed a hybrid secure inner-product computation protocol for building linear regression models with vertically partitioned data, which has comparable accuracy to other schemes without privacy constraints. Wang et al. [41] proposed a privacy-preserving vertical FL scheme for naive bayesian classification. Specifically, they designed a modified Paillier cryptosystem by splitting the secret key, which can securely aggregate the training data for model construction. Moreover, they applied the random masking technique to protect the query data.

### B. Vertical FL Schemes for Non-Linear Models

Besides the above linear models, vertical FL schemes for non-linear models like logistic regression also received the attention of researchers.

Hardy et al. [36] proposed a three-party federated logistic regression scheme over vertically partitioned data. In their scheme, the loss function is first approximated by Taylor series expansion for fitting homomorphic encryption. Then, two participants encrypt their partial gradients with additive homomorphic encryption, which are combined and decrypted by a third party for model updating. However, any participant can infer the partial gradients of another form the model updates. Zhao et al. [42] solved the above privacy issues by designing a data aggregation matrix construction algorithm, which can aggregate the vertically partitioned data for logistic regression model training, meanwhile, without multi-round interactions between participants and the third party.

### C. Vertical FL Schemes for Boosting Tree Models

Most vertical federated tree boosting schemes assume that participants are divided into an active one holding data labels and other passive ones.

Based on homomorphic encryption, Cheng et al. [11] proposed a vertical federated tree boosting framework named SecureBoost, where the ciphertext gradients are transferred between active and passive participants for performing split findings. However, the split information is leaked to passive participants in SecureBoost; moreover, the passive participants must stay connected with the active participant during training. Tian et al. [15] proposed an FL framework named FederBoost for gradient boosting decision tree model, which supports model training over both horizontally and vertically partitioned data. In vertical FederBoost, the bucket information of passive participants is perturbed by an exponential mechanism of differential privacy. However, the noises in buckets will reduce the accuracy of the global model. By combining the threshold partially homomorphic encryption and secure multi-party computation protocols, Wu et al. [17] designed a hybrid scheme named Pivot for vertical federated decision tree training. Pivot can strictly protect the data information, but the frequently used cryptographic operations cause unacceptable overhead.

Different from the above schemes, SGBoost achieves lossless and efficient federated tree boosting, while protecting the bucket and split information. Moreover, participants do not need to stay online in the model training process. Finally, we compare the security and functionality of SGBoost with several existing representative schemes in TABLE III.

## IX. CONCLUSION

In this paper, we have proposed SGBoost, an efficient and privacy-preserving vertical federated tree boosting framework, which can achieve the high-accuracy model training and query securely and efficiently. Security analysis demonstrates that SGBoost can protect the training and query data information against the inference of honest-but-curious CSP and participants. Moreover, experimental results on several real-world datasets show its high accuracy and efficient performance. To further improve the system efficiency, we consider establishing a trusted execution environment (TEE) [43] in the initiator to assist in model training, which might be explored in our future studies.

## REFERENCES

[1] European Parliament and Council of the European Union. (Apr. 2016). *General Data Protection Regulation*. [Online]. Available: https://eur-lex.europa.eu/eli/reg/2016/679/oj/

[2] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. Y. Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial Intelligence and Statistics* (Proceedings of Machine Learning Research), vol. 54. Fort Lauderdale, FL, USA: PMLR, Apr. 2017, pp. 1273–1282.

[3] P. Kairouz et al., "Advances and open problems in federated learning," *Found. Trends Mach. Learn.*, vol. 14, nos. 1–2, pp. 1–210, Jun. 2021.
[4] L. Melis, C. Song, E. De Cristofaro, and V. Shmatikov, "Exploiting unintended feature leakage in collaborative learning," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2019, pp. 691–706.
[5] L. Zhu, Z. Liu, and S. Han, "Deep leakage from gradients," in *Proc. NeurIPS*, 2019, pp. 14747–14756.
[6] L. Lyu, H. Yu, and Q. Yang, "Threats to federated learning: A survey," 2020, *arXiv:2003.02133*.
[7] H. Yin, A. Mallya, A. Vahdat, J. M. Alvarez, J. Kautz, and P. Molchanov, "See through gradients: Image batch recovery via GradInversion," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2021, pp. 16337–16346.
[8] Q. Yang, Y. Liu, Y. Cheng, Y. Kang, T. Chen, and H. Yu, *Federated Learning* (Synthesis Lectures on Artificial Intelligence and Machine Learning). San Rafael, CA, USA: Morgan & Claypool, 2019.
[9] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *ACM Trans. Intell. Syst. Technol.*, vol. 10, no. 2, p. 12, 2019.
[10] Y. Liu, Y. Liu, Z. Liu, J. Zhang, C. Meng, and Y. Zheng, "Federated forest," 2019, *arXiv:1905.10053*.
[11] K. Cheng et al., "SecureBoost: A lossless federated learning framework," *IEEE Intell. Syst.*, vol. 36, no. 6, pp. 87–98, Dec. 2021.
[12] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Advances in Cryptology—EUROCRYPT* (Lecture Notes in Computer Science), vol. 1592. Prague, Czech Republic: Springer, 1999, pp. 223–238.
[13] C. Dwork, "Differential privacy: A survey of results," in *Theory and Applications of Models of Computation* (Lecture Notes in Computer Science), vol. 4978. Xi'an, China: Springer, 2008, pp. 1–19.
[14] K. Wei et al., "Federated learning with differential privacy: Algorithms and performance analysis," *IEEE Trans. Inf. Forensics Security*, vol. 15, pp. 3454–3469, 2020.
[15] Z. Tian, R. Zhang, X. Hou, J. Liu, and K. Ren, "FederBoost: Private federated learning for GBDT," 2020, *arXiv:2011.02796*.
[16] H. Zhu, R. Wang, Y. Jin, and K. Liang, "PIVODL: Privacy-preserving vertical federated learning over distributed labels," *IEEE Trans. Artif. Intell.*, early access, Dec. 28, 2021, doi: 10.1109/TAI.2021.3139055.
[17] Y. Wu, S. Cai, X. Xiao, G. Chen, and B. C. Ooi, "Privacy preserving vertical federated learning for tree-based models," *Proc. VLDB Endowment*, vol. 13, no. 12, pp. 2090–2103, Aug. 2020.
[18] I. Damgård, V. Pastro, N. P. Smart, and S. Zakarias, "Multiparty computation from somewhat homomorphic encryption," in *Advances in Cryptology* (Lecture Notes in Computer Science), vol. 7417. New York, NY, USA: Springer-Verlag, Aug. 2012, pp. 643–662.
[19] A. Beimel, "Secret-sharing schemes: A survey," in *IWCC Coding and Cryptology* (Lecture Notes in Computer Science), vol. 6639. Qingdao, China: Springer, 2011, pp. 11–46.
[20] D. Boneh, A. Sahai, and B. Waters, "Functional encryption: Definitions and challenges," in *Proc. TCC*, (Lecture Notes in Computer Science), vol. 6597. Cham, Switzerland: Springer, 2011, pp. 253–273.
[21] H. Mahdikhani, R. Lu, Y. Zheng, J. Shao, and A. A. Ghorbani, "Achieving $O(\log^3)n)$ communication-efficient privacy-preserving range query in fog-based IoT," *IEEE Internet Things J.*, vol. 7, no. 6, pp. 5220–5232, Jun. 2020.
[22] Q. Zhang, B. Gu, C. Deng, and H. Huang, "Secure bilevel asynchronous vertical federated learning with backward updating," in *Proc. AAAI Conf. Artif. Intell.* Palo Alto, CA, USA: AAAI Press, 2021, pp. 10896–10904.
[23] O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game, or a completeness theorem for protocols with honest majority," in *Providing Sound Foundations for Cryptography*. New York, NY, USA: ACM, 2019, pp. 307–328.
[24] S. Kariyappa, A. Prakash, and M. K. Qureshi, "MAZE: Data-free model stealing attack using zeroth-order gradient estimation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2021, pp. 13814–13823.
[25] M. Juuti, S. Szyller, S. Marchal, and N. Asokan, "PRADA: Protecting against DNN model stealing attacks," in *Proc. IEEE Eur. Symp. Secur. Privacy (EuroS P)*, Jun. 2019, pp. 512–527.
[26] C. Fung, C. J. M. Yoon, and I. Beschastnikh, "The limitations of federated learning in sybil settings," in *Proc. 23rd Int. Symp. Res. Attacks, Intrusions Defenses (RAID)*. Berkeley, CA, USA: USENIX Association, 2020, pp. 301–316.
[27] M. Fang, X. Cao, J. Jia, and N. Z. Gong, "Local model poisoning attacks to Byzantine-robust federated learning," in *Proc. USENIX Secur. Symp.* Berkeley, CA, USA: USENIX Association, 2020, pp. 1605–1622.
[28] X. Liu, H. Li, G. Xu, Z. Chen, X. Huang, and R. Lu, "Privacy-enhanced federated learning against poisoning adversaries," *IEEE Trans. Inf. Forensics Security*, vol. 16, pp. 4574–4588, 2021.
[29] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2016, pp. 785–794.
[30] A. B. Lewko, T. Okamoto, A. Sahai, K. Takashima, and B. Waters, "Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption," in *Advances in Cryptology—EUROCRYPT* (Lecture Notes in Computer Science), vol. 6110. Monaco: Springer, 2010, pp. 62–91.
[31] M. Abdalla, F. Bourse, A. D. Caro, and D. Pointcheval, "Simple functional encryption schemes for inner products," in *Public Key Cryptography* (Lecture Notes in Computer Science), vol. 9020. Gaithersburg, MD, USA: Springer, 2015, pp. 733–751.
[32] Y. Zheng, R. Lu, Y. Guan, S. Zhang, J. Shao, and H. Zhu, "Efficient and privacy-preserving similarity query with access control in eHealthcare," *IEEE Trans. Inf. Forensics Security*, vol. 17, pp. 880–893, 2022.
[33] A. Acar, H. Aksu, A. S. Uluagac, and M. Conti, "A survey on homomorphic encryption schemes: Theory and implementation," *ACM Comput. Surv.*, vol. 51, no. 4, p. 79, Jul. 2018.
[34] Y. Zheng, R. Lu, Y. Guan, J. Shao, and H. Zhu, "Efficient and privacy-preserving similarity range query over encrypted time series data," *IEEE Trans. Dependable Secure Comput.*, vol. 19, no. 4, pp. 2501–2516, Jul. 2022.
[35] Y. Guan, R. Lu, Y. Zheng, S. Zhang, J. Shao, and G. Wei, "Toward privacy-preserving cybertwin-based spatiotemporal keyword query for ITS in 6G era," *IEEE Internet Things J.*, vol. 8, no. 22, pp. 16243–16255, Nov. 2021.
[36] S. Hardy et al., "Private federated learning on vertically partitioned data via entity resolution and additively homomorphic encryption," 2017, *arXiv:1711.10677*.
[37] Distributed (Deep) Machine Learning Community. (Feb. 2014). *Scalable, Portable and Distributed Gradient Boosting (GBDT, GBRT or GBM) Library*. [Online]. Available: https://github.com/dmlc/xgboost
[38] 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," *Exp. Syst. Appl.*, vol. 36, pp. 2473–2480, Mar. 2009.
[39] S. Moro, P. Cortez, and P. Rita, "A data-driven approach to predict the success of bank telemarketing," *Decis. Support Syst.*, vol. 62, no. 1, pp. 22–31, Jun. 2014.
[40] A. Gascón et al., "Secure linear regression on vertically partitioned datasets," in *Proc. IACR Cryptol. ePrint Arch.*, 2016, p. 892.
[41] F. Wang, H. Zhu, R. Lu, Y. Zheng, and H. Li, "Achieve efficient and privacy-preserving disease risk assessment over multi-outsourced vertical datasets," *IEEE Trans. Dependable Secure Comput.*, vol. 19, no. 3, pp. 1492–1504, May 2022.
[42] J. Zhao et al., "ACCEL: An efficient and privacy-preserving federated logistic regression scheme over vertically partitioned data," *Sci. China Inf. Sci.*, vol. 65, no. 7, pp. 1–2, 2022.
[43] M. Sabt, M. Achemlal, and A. Bouabdallah, "Trusted execution environment: What it is, and what it is not," in *Proc. IEEE Trustcom/BigDataSE/ISPA*, Aug. 2015, pp. 57–64.

**Jiaqi Zhao** was born in China in 1997. He received the B.Eng. degree in information security from Xidian University, Xi'an, Shaanxi, China, in 2020, where he is currently pursuing the Ph.D. degree in cyberspace security.

His research has been concerned with privacy-preserving machine learning.

**Hui Zhu** (Senior Member, IEEE) received the B.Sc. degree from Xidian University, Xi'an, Shaanxi, China, in 2003, the M.Sc. degree from Wuhan University, Wuhan, Hubei, China, in 2005, and the Ph.D. degree from Xidian University in 2009.

He was a Research Fellow with the School of Electrical and Electronics Engineering, Nanyang Technological University, Singapore, in 2013. Since 2016, he has been a Professor with the School of Cyber Engineering, Xidian University. His current research interests include applied cryptography, data security, and privacy.

**Wei Xu** was born in China in 1999. He received the B.Eng. degree in information security from Xidian University, Xi'an, Shaanxi, China, in 2022, where he is currently pursuing the M.A. degree in cyberspace security.

His research has been concerned with privacy-preserving machine learning.

**Fengwei Wang** received the B.Sc. and Ph.D. degrees from Xidian University, China, in 2016 and 2021, respectively.

In 2019, he was a Visiting Scholar with the Faculty of Computer Science (FCS), University of New Brunswick (UNB), Canada. Since 2021, he has been a Lecturer with the School of Cyber Engineering, Xidian University. His research interests include applied cryptography, big data security, and privacy protection.

**Rongxing Lu** (Fellow, IEEE) received the Ph.D. degree from the Department of Electrical and Computer Engineering, University of Waterloo, Canada, in 2012. He is currently the Mastercard IoT Research Chair, a University Research Scholar, and an Associate Professor at the Faculty of Computer Science (FCS), University of New Brunswick (UNB), Canada. Before that, he worked as an Assistant Professor at the School of Electrical and Electronic Engineering, Nanyang Technological University (NTU), Singapore, from April 2013 to August 2016. He worked as a Post-Doctoral Fellow at the University of Waterloo from May 2012 to April 2013. His research interests include applied cryptography, privacy enhancing technologies, and the IoT-big data security and privacy. He has published extensively in his areas of expertise. He was awarded the most prestigious Governor General's Gold Medal, when he received the Ph.D. degree in 2012 and won the 8th IEEE Communications Society (ComSoc) Asia Pacific (AP) Outstanding Young Researcher Award in 2013. He was a recipient of nine best (student) paper awards from some reputable journals and conferences. He also serves as the Chair for IEEE Communications and Information Security Technical Committee (ComSoc CIS-TC) and the Founding Co-Chair for IEEE TEMS Blockchain and Distributed Ledgers Technologies Technical Committee (BDLT-TC). He is the Winner of 2016–2017 Excellence in Teaching Award, FCS, UNB.

**Hui Li** (Member, IEEE) received the B.Sc. degree from Fudan University in 1990 and the M.Sc. and Ph.D. degrees from Xidian University, China, in 1993 and 1998, respectively.

Since 2005, he has been a Professor with the School of Telecommunication Engineering, Xidian University. His research interests include cryptography, wireless network security, information theory, and network coding.

Dr. Li served as the TPC Co-Chair for ISPEC 2009 and IAS 2009, the General Co-Chair for E-Forensic 2010, ProvSec 2011, and ISC 2011, and the Honorary Chair for NSS 2014 and ASIACCS 2016.