FEDTREE: A FEDERATED LEARNING SYSTEM FOR TREES

Qinbin Li [1 2] **Zhaomin Wu** [1] **Yanzheng Cai** [3] **Yuxuan Han** [1] **Ching Man Yung** [1] **Tianyuan Fu** [1] **Bingsheng He** [1]

ABSTRACT

While the quality of machine learning services largely relies on the volume of training data, data regulations such as the General Data Protection Regulation (GDPR) impose stringent requirements on data transfer. Federated learning has emerged as a popular approach for enabling collaborative machine learning without sharing raw data. To facilitate the rapid development of federated learning, efficient and user-friendly federated learning systems are essential. Despite many existing federated learning systems designed for deep learning, tree-based federated learning systems have not been well exploited. This paper presents a tree-based federated learning system under a histogram-sharing scheme, named FedTree, that supports both horizontal and vertical federated training of GBDTs with configurable privacy protection techniques. Our extensive experiments show that FedTree achieves competitive accuracy to centralized training while incurring much less computational cost than the other generic federated learning systems.

1 INTRODUCTION

Federated Learning (FL) (Yang et al., 2019; Kairouz et al., 2019; Li et al., 2019) has been a very attractive research direction that enables collaborative machine learning among multiple parties without exchanging raw data. It has wide real-world applications including healthcare (Rieke et al., 2020; Pfitzner et al., 2021), finance (Long et al., 2020), and mobile services (Hard et al., 2018). There have been many FL systems (Liu et al., 2021; Beutel et al., 2020; He et al., 2020; Bonawitz et al., 2019) that play important roles in advancing the development and deployment of FL.

Nonetheless, tree-based models, featuring efficiency and explainability, have received less attention from the FL community despite their ubiquitous adoption in real applications. Most existing FL systems do not exploit trees in their design. Their systematical design is based on the FedAvg (McMahan et al., 2017) framework, which averages the local models to update the global model based on stochastic gradient descent (SGD). However, the model parameters of trees are non-differentiable and federated training of trees cannot be achieved by FedAvg.

Gradient Boosting Decision Trees (GBDT) is a popular tree model that has won many awards in machine learning and data mining competitions (Chen & Guestrin, 2016). Given the weakness of a single tree in fitting complicated data distribution, GBDT sequentially trains multiple trees to boost the model performance, where each tree is trained to fit the residual between the prediction values of previous trees and the label. GBDT has been successfully deployed in many applications (Richardson et al., 2007; Kim et al., 2009; Burges, 2010). Given the superiority of GBDT in efficiency and effectiveness compared to neural networks especially for tabular data, it is necessary to develop a specialized federated learning system for GBDT.

While existing GBDT systems such as XGBoost (Chen & Guestrin, 2016), LightGBM (Ke et al., 2017), and CatBoost (Dorogush et al., 2018) support the distributed training, they cannot be applied to the federated setting with privacy constraints. These systems assume that all data are in a host server that splits the data and distributes the jobs to each client machine in each round using tools such as Spark (Zaharia et al., 2010). This approach is not applicable in the federated setting, where the data are stored in clients and the server has no access to clients' data. Thus, we need to design new federated GBDT algorithms and systems.

A key question of such system design is *how to select the knowledge for exchanging in federated GBDT*. Model parameters and gradients are the key knowledge for sharing in FedAvg. However, as directly averaging the models is not possible in GBDT, we need to reconsider the format of knowledge for sharing. This knowledge should satisfy three requirements for practicability: 1) *Effectiveness*: The knowledge should contain the necessary information for accurate GBDT training and can be easily aggregated; 2) *Efficiency*: The size of knowledge should be small to reduce both communication and computation costs; 3) *Pri-*

[1]National University of Singapore, Singapore [2]UC Berkeley, USA [3]Tsinghua University, China. Correspondence to: Qinbin Li <liqinbin1998@gmail.com>.

Proceedings of the 6th MLSys Conference, Miami Beach, FL, USA, 2023.

vacy: The knowledge should be compatible with privacy techniques to provide rigorous privacy guarantees. By investigating the operations inside a tree node, we find that histogram (i.e., sums of segmented gradients) is a key statistic in GBDT training and satisfies the communication and privacy requirements.

We develop a unified histogram-sharing scheme for horizontal and vertical FL settings, in which parties compute local histograms and the server aggregates the histograms for subsequent updates. Based on the scheme, we design our aggregation and privacy operators to enable accurate and privacy-preserving knowledge sharing applicable to various federated learning scenarios. Utilizing these operators and the training scheme, we create horizontal and vertical federated GBDT systems that support different privacy levels. Additionally, we optimize communication and computation speeds by leveraging the node-independent properties of trees.

Through careful algorithmic design and engineering efforts, we implement the first comprehensive tree-specialized FL system FedTree that enables fast and accurate training of GBDT in various federated settings. FedTree supports standalone FL simulation on a single machine and distributed computing for real FL deployment on various scenarios with configurable privacy techniques. Our evaluation shows that FedTree can achieve almost the same accuracy as centralized learning while achieving up to 81x speedup compared with other generic FL systems.

Our work has the following main contributions: 1) FedTree is the first comprehensive federated learning system specialized for trees; 2) FedTree demonstrates a comprehensive histogram-sharing strategy, facilitating both horizontal and vertical FL; 3) By incorporating cryptographic and differential privacy methods, FedTree provides rigorous privacy guarantees; 4) FedTree incorporates effective techniques to expedite computation and communication by capitalizing on the inherent properties of trees within federated contexts.

2 Background

2.1 Gradient Boosting Decision Trees

The GBDT model has not only won many awards in machine learning and data mining competitions (Chen & Guestrin, 2016), but also has been widely used in real-world applications (Richardson et al., 2007; Kim et al., 2009). An example of GBDT with two trees is shown in Figure 1. In each tree, split nodes split the input into two directions and leaf nodes contain the prediction values. The final prediction value is the sum of prediction value of each tree.

The training of the GBDT model is in a sequential manner. In each iteration, a new tree is trained to fit the residual

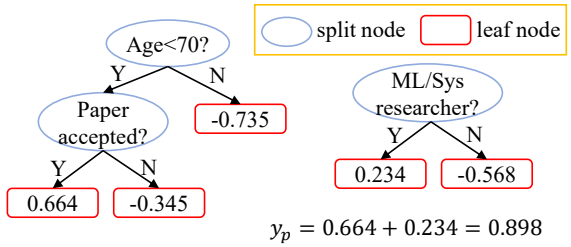

Figure 1. An example of GBDT predicting whether a person will attend MLSys conference.

between the prediction and the target. Formally, given a loss function l and a dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$, GBDT minimizes the following objective function

$$\mathcal{L} = \sum_i l(y_i, \hat{y}_i) + \sum_k \Omega(f_k), \tag{1}$$

where $\hat{y}_i$ is the prediction value, $\Omega(\cdot)$ is a regularization term and f_k denotes a decision tree. At the t-th iteration with second-order approximation (Chen & Guestrin, 2016), GBDT minimizes the following objective function

$$\begin{aligned}\tilde{\mathcal{L}}^{(t)} &= \sum_i l(y_i, \hat{y}_i^{t-1} + f_t(\mathbf{x}_i)) + \Omega(f_t) \\ &\approx \sum_i [l(y_i, \hat{y}_i^{t-1}) + g_i f_t(\mathbf{x}_i) + \frac{1}{2} h_i f_t^2(\mathbf{x}_i)] + \Omega(f_t)\end{aligned} \tag{2}$$

where $g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$ and $h_i = \partial^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$ are first and second order gradient statistics on the loss function.

We use I to denote the instance ID set in the current node. Then, if the current node is a leaf node, to minimize Eq (2), the optimal leaf value is

$$V = -\frac{\sum_{i \in I} g_i}{\sum_{i \in I} h_i + \lambda} \tag{3}$$

If the current node is a split node, suppose the split value splits I into I_L and I_R. Then, the gain of the split value is defined by the loss reduction after split, which is

$$S = \frac{(\sum_{i \in I_L} g_i)^2}{\sum_{i \in I_L} h_i + \lambda} + \frac{(\sum_{i \in I_R} g_i)^2}{\sum_{i \in I_R} h_i + \lambda}. \tag{4}$$

Since it would be computationally expensive to traverse all the possible split values to find the one with the maximum gain, histogram-based training is usually adopted in practice. In histogram-based training, a small number of cut points as possible split candidates are proposed first and the gradient histogram can be computed based on the cut points. Then, the best split value is determined among the cut points using the gradient histogram. In such a way, the computation time can be significantly reduced.

2.2 Federated Learning

FedAvg (McMahan et al., 2017) has been a de facto approach for FL. There are four steps in each round of FedAvg. First, the server sends a global model to the parties. Second, the parties perform SGD to update their models locally. Third, the local models are sent to a central server. Last, the server averages the models to produce a global model for the training of the next round. Although FedAvg is popular and powerful, it cannot be used to train non-differentiable models such as GBDT. For decision trees, the model parameters including the split values and leaf values are determined by statistics and cannot be updated by gradient descent. Thus, it is necessary to design specialized algorithms and systems for decision trees in a federated setting.

Federated GBDT Systems There have been many FL systems nowadays. Among these systems, only a few systems support training trees in a federated setting here. FATE (Liu et al., 2021) is an FL platform that supports many kinds of machine learning models including GBDTs (Cheng et al., 2019). However, its federated GBDT algorithm does not follow the typical server-clients setting (e.g., requiring an additional arbiter). Moreover, as it is a complicated system, it is not easy to deploy and it does not exploit the unique attributes of trees to speed up the training. Secure XGBoost (Law et al., 2020) uses secure hardware enclaves such as Intel SGX to enable the collaborative training of XGBoost in a cloud. The requirement for a cloud with secure hardware enclaves limits the applications a lot. Moreover, as the training is conducted on encrypted data, Secure XGBoost is quite slow. FedLearner (fed) supports federated GBDT in the vertical FL setting. However, it does not provide any documentation about its algorithm.

Federated GBDT algorithms Besides the above systems, there are some studies that exploit federated GBDTs algorithms. Zhao et al. (2018) and Li et al. (2020a) propose to conduct horizontally federated GBDTs by transferring trees, i.e., each party trains some trees and transfers them to other parties for boosting. Such tree-level communication has a large accuracy gap between centralized GBDT since only local data is used in each tree. Fang et al. (2021) and Wu et al. (2020) propose to train vertical federated GBDT utilizing secure multi-party computation techniques for many operators, which is very computational-expensive. All the above studies do not consider the unified architecture to enable both horizontal and vertical FL and optimizations to improve efficiency from the systematic perspective.

2.3 Homomorphic Encryption

Homomorphic encryption (Acar et al., 2018) enables users to perform computations on the ciphertext without decrypting it. Specifically, additively Homomorphic Encryption (HE) supports the addition operations on the ciphertexts. A classic additively HE approach is Paillier cryptosystem (Paillier, 1999), which generates a public key k_{pub} for encryption and addition and private key k_{pri} for decryption. We use $Dec(\cdot)$ to denote the decryption function, $\|\cdot\| := Enc(\cdot)$ to denote the ciphertext, and $\oplus$ to denote the addition operation on the ciphertexts. Then, for any two number a and b, we have $Dec(\|a\| \oplus \|b\|) = a + b$.

2.4 Differential Privacy

Differential Privacy (Dwork et al., 2014) (DP) is a popular standard of privacy protection and has been widely used to protect the machine learning models (Abadi et al., 2016). It guarantees that the probability of producing a given output does not depend much on whether a particular data record is included in the input dataset or not. The definition of ε-Differential Privacy is below.

Definition 1. *(ε-Differential Privacy) Let $\mathcal{M}: \mathcal{D} \to \mathcal{R}$ be a randomized mechanism with domain $\mathcal{D}$ and range R. M satisfies ε-differential privacy if for any two adjacent inputs $d, d' \in \mathcal{D}$ and any subset of outputs $S \subseteq \mathcal{R}$ it holds that:*

$$\Pr[\mathcal{M}(d) \in S] \leq e^{\varepsilon} \Pr[\mathcal{M}(d') \in S]. \tag{5}$$

3 System Overview

As shown in Figure 2, FedTree has five major components to enable the easy usage and deployment of federated trees in real-world applications.

Interfaces FedTree supports two kinds of interfaces for ease of use: command-line interface (CLI) through a configuration file and Python interface aligned with scikit-learn (Pedregosa et al., 2011). Users only need to input the parameters (e.g., number of parties, federated setting) to define the training scenario. Then, FedTree can be launched with a one-line command.

Environment FedTree supports standalone simulation and distributed computing. Standalone simulation is mainly for research purposes, where it can simulate the federated setting in a single machine. We use CUDA (Sanders & Kandrot, 2010) to utilize GPUs for encryption acceleration. For real deployment on multiple machines, we adopt gRPC as the communication protocol.

Frameworks The core algorithms of FedTree are horizontal and vertical FL of GBDTs. In the horizontal FL setting, each party shares the same feature space but different sample spaces. In the vertical FL setting, each party shares the same sample space but different feature spaces. Besides horizontal and vertical FL, FedTree also integrates a simple FL framework that aggregates the local trees of different parties for ensemble.

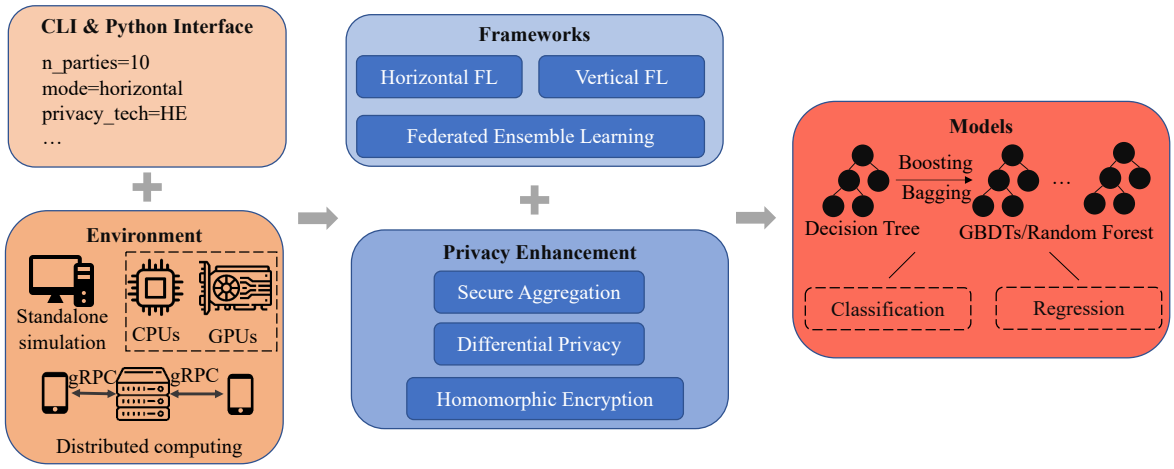

Figure 2. System components of FedTree.

Privacy Enhancement We provide HE and SA to protect the communicated messages without losing the model accuracy. DP is also implemented as an option to provide rigorous privacy guarantees.

Models The core supported model of FedTree is GBDT, which is based on the federated operators for training each tree node. FedTree also supports random forests as a plugin by applying instance bagging when training each tree. Multiple objective functions such as square loss, logistic loss, and softmax loss have been implemented to support both classification and regression tasks.

Next, we introduce the core design methodology and the major optimization techniques in FedTree.

4 Federated GBDT Design

In this section, we introduce the design methodology of FedTree. We start by revisiting the building process in centralized GBDT training in Section 4.1, which motivates us to use histograms as the key knowledge for transferring. Next, we introduce the aggregation operators and privacy mechanisms in Section 4.2 and Section 4.3, respectively. Then, in Section 4.4, we introduce FedTree training frameworks in the horizontal and vertical FL settings with our designed architecture, operators, and privacy mechanisms.

4.1 Building Block - A Single Node

We start by investigating the process of building a single tree node in the centralized setting. Given the instance set in the current node, we first need to construct the histogram according to the cut points and the gradients. Given the cut points $\{c_i\}_{i=1}^{B}$ (B is the number of cut points) of feature a, supposed the left instance ID set split by cut point c_i is denoted as I_i (i.e., $I_i = \{k : \mathbf{x}_k^a \leq c_i\}$), the histogram is constructed as a vector of gradient pairs. Formally,

$$\mathbf{H} = \left[\left(\sum_{k \in I_i} g_k, \sum_{k \in I_i} h_k \right) \right]_{i=1}^{B}. \tag{6}$$

The sum of gradients of right-side instances can be easily computed by the sum of all gradients minus the histogram for the left one (i.e., $\mathbf{H}_B - \mathbf{H}_i$). After constructing the histogram, we can compute the gain of each cut point by Equation (4) and select the cut point with the maximum gain as the split value. If the gain is always smaller than zero or the tree reaches the pre-defined maximum number of depth, the current node is a leaf node and the leaf value is computed by Equation (3). Figure 3 summarizes the process of building a node. We observe that the histogram has the following nice properties: 1) The histogram is the only required statistic to update the node's value; 2) The histogram is usually small, which has a low dimension equal to the number of cut points; 3) The histogram naturally preserves the privacy of raw data in a level as it does not contain the raw feature values. Considering effectiveness, efficiency, and privacy, the histogram is a very appropriate choice for sharing in the federated setting. Thus, we develop our histogram-based FL architecture as below.

Histogram-based Learning Based on histogram sharing, we design our unified federated architecture as shown in Figure 4 and Algorithm 1. At the beginning of FL, the parties communicate with the server to get the initialized parameters (e.g., public keys for encryption) (line 1). In each round, communicating in a node-level, the parties compute the histograms locally and send the histograms to server with optional privacy mechanisms to protect the histograms (line 6). Then, the server aggregates the histograms and computes the node's parameter (line 8). Next, the parties receive the node from the server to further update the local

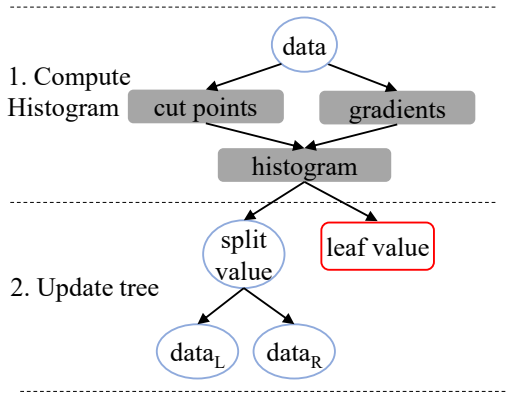

Figure 3. The process of updating a tree node.

statistics (line 10). If the node is a split node, the parties split the current instance set according to the split value. If the node is a leaf node, the parties update the prediction value. Such an architecture is applicable to various FL scenarios, including horizontal and vertical FL. Next, we introduce the aggregation operators and privacy mechanisms in detail.

Algorithm 1 The training procedure of FedTree

input Parties $P = \{P_1, ..., P_N\}$, server S, number of trees T, tree depth d

1: *Init(P, S)*
2: **for** $t = 1$ **to** T **do**
3: **for** $depth = 1$ **to** d **do**
4: **for** each node in current depth **do**
5: **for** $i = 1$ **to** N **do**
6: *PartyComputeHistogram(P_i)*
7: **end for**
8: $node \leftarrow$ *ServerAggregate(depth)*
9: **for** $i = 1$ **to** N **do**
10: *PartyUpdate(P_i, node)*
11: **end for**
12: **end for**
13: **end for**
14: **end for**

4.2 Histogram Aggregation

4.2.1 Histogram summation

The histogram is to compute the sum of gradients in buckets. If multiple parties have the feature values of the same feature (e.g., horizontal FL), we need to merge the local histograms by summation. Considering a single feature a, we use $\mathbf{C}^j = \{c_i^j\}_{i=1}^B$ to denote the cut points of party P_j, $I_i^j = \{k : \mathbf{x}_k^a \leq c_i^j\}$ to denote the left instance ID set split by cut point c_i^j in party P_j, and $\mathbf{H}^j$ to denote the local histogram of party P_j. If the cut points of each party are exactly the same (i.e., $\forall m, n \in [N], \mathbf{C}^m = \mathbf{C}^n$), we can easily compute the histogram for the global data with the same cut points by summing all the local histograms, i.e.,

$$\mathbf{H} = \left[\left(\sum_{j\in[N]} \sum_{k\in I_i^j} g_k, \sum_{j\in[N]} \sum_{k\in I_i^j} h_k \right) \right]_{i=1}^{B} = \sum_{j\in[N]} \mathbf{H}^j. \tag{7}$$

Inconsistent Cut Points In federated setting, the cut points locally proposed by each party may not be the same. One simple solution is to force all parties to have the same cut points with the help of the server. Specifically, the parties first propose the cut points to the server, which merges all the cut points and sends back all the cut points to the parties. The parties use the merged cut points to compute the histograms. This solution will introduce additional communication costs. Another solution is to directly merge the local histograms by approximate summation. Specifically, assume that the m-th cut point of the merged histogram is c_m, which is between two neighboring cut points c_i^j and c_{i+1}^j of party j. Then, assuming the feature values are uniformly distributed, the sum of gradients of instances between c_i^j and c_m can be approximated by the sum of gradients of instances between c_i^j and c_{i+1}^j scaling with the relative distance (i.e., $\frac{c_m - c_i^j}{c_{i+1}^j - c_i^j}$). We have

$$\mathbf{H}_m = \sum_{j\in[N]} \left[\mathbf{H}_i^j + (\mathbf{H}_{i+1}^j - \mathbf{H}_i^j) * \frac{c_m - c_i^j}{c_{i+1}^j - c_i^j} \right] \tag{8}$$

Equation (8) is also applicable when the cut point of the final point is same as a local cut point (i.e., $c_m = c_i^j$).

4.2.2 Histogram concatenation

The above summation only considers the histogram of a single feature. If the feature values of different features are distributed in multiple parties (e.g., vertical FL), we need to merge the histograms of different parties through concatenation. We use $\mathbf{H}^j$ to denote the histogram of party P_j, then the global histogram can be constructed by

$$\begin{aligned} \mathbf{H} &= \left[\left(\sum_{k\in I_1^j} g_k, \sum_{k\in I_1^j} h_k \right), ..., \left(\sum_{k\in I_B^j} g_k, \sum_{k\in I_B^j} h_k \right) \right]_{j=1}^{N} \\ &:= \bigcup_{j=1}^{N} \mathbf{H}^j \end{aligned} \tag{9}$$

Transferring Gradients with Homomorphic Encryption One possible issue is that some parties may not have the labels to compute the gradients in vertical FL. In such a setting, we require the host party (i.e., the party with labels) to

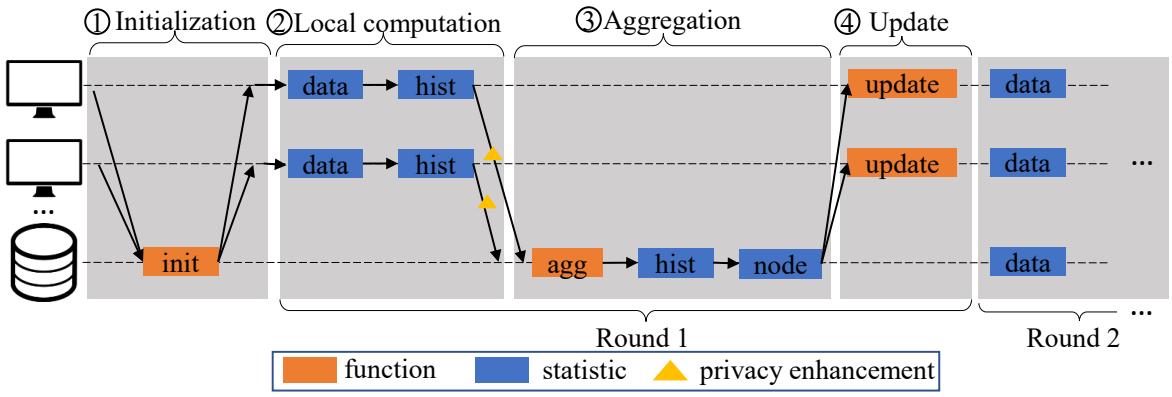

Figure 4. The architecture of FedTree with histogram-based learning scheme.

share the gradients to the parties without labels. Since gradients are computed based on the labels and raw gradients may leak the label information, we apply additive homomorphic encryption to protect the gradients. Specifically, the host party uses Paillier cryptosystem (Paillier, 1999) to generate public key k_{pub} and private key k_{pri}. Then, the host party encrypts the gradients using k_{pub} and shares the encrypted gradients and public keys to the parties without labels. The parties without labels can conduct addition operations on the encrypted gradients to compute the encrypted histogram.

4.3 Privacy Mechanisms

Threat Model We assume all parties and the server are honest-but-curious, i.e., they strictly follow the protocol but may infer sensitive information about others through the received information. Besides the model parameters and possibly the encrypted gradients, only the histograms are the shared information related to the local data. The gradient is computed by the distance between the prediction value and the label. Thus, sharing raw histograms may leak information about the labels of the raw data. Here we provide two mechanisms to protect the histograms.

Histogram Summation with Secure Aggregation Like FedAvg, we can also adopt secure aggregation (Bonawitz et al., 2016) to protect the histograms without accuracy loss. For every pair of parties (P_i, P_j), they use Diffie-Hellman (DH) key exchange (Diffie & Hellman, 2022) to share a common key k_{ij}, which is used as noise to encrypt the histogram. Specifically, for party i, it adds the noises $\sum_{j\in[N]} k_{ij} - \sum_{j\in[N]} k_{ji}$ to every element in $\mathbf{H}^i$. Then, the noises will cancel out with each other when summing the local histograms. We can still get the correct final histogram while the local histogram is well-protected.

Histogram Protection with Differential Privacy Although secure aggregation can protect the local histogram, the global histogram is still known by the server which also includes the label knowledge about the training data. Moreover, in the case that no histogram summation is applied (e.g., vertical FL), secure aggregation is not applicable and each local histogram is known by the server. Thus, we apply differential privacy to further protect the exchanged histogram. For each first-order gradient g, we clip them with a threshold R ($R > 0$). With a constant second-order gradient, the sensitivity of the histogram is $2R$. With Laplace mechanism (Dwork et al., 2014), when we add noises from the Laplace distribution with mean 0 and scale $\frac{2R}{\varepsilon}$ (denoted as $Lap(0, \frac{2R}{\varepsilon})$) to the histogram, the output histogram satisfies ε-Differential Privacy.

4.4 Federated Training Framework

We present the unified training procedure in Figure 4 and Algorithm 1. Next, we present the detailed operations for each step in the horizontal and vertical FL settings.

Horizontal FedTree In the horizontal FL setting, each party has the same feature space but different sample spaces (e.g., different hospitals have CT images of differential patients). The horizontal FedTree with optional secure aggregation and differential privacy is shown in Algorithm 2. First, in the initialization, each pair of parties generates a shared key using Diffie-Hellman key exchange (lines 1-4). Then, for the local histogram computation, we clip the gradients, compute the raw histogram, and add the noises from secure aggregation and differential privacy to the histogram (lines 5-11). During aggregation, the server computes the global histogram and updates the value of the current node (lines 12-19), which is sent to the parties to split the instance set or update the prediction value (lines 20-25).

Vertical FedTree In the vertical FL setting, each party has the same sample space but different feature spaces (e.g., a bank and an insurance company have the same users but

Algorithm 2 Horizontal FedTree

input Parties $P_1, ..., P_N$, server S, threshold R, number of trees T, tree depth d

1: ***Init***(P, S):
2: **for** every pair of $(P_i, P_j)(i \neq j)$ **do**
3: generate common key k_{ij} through DH key exchange
4: **end for**

5: ***PartyComputeHistogram***(P_i):
6: $\mathbf{G} \leftarrow UpdateGradients()$
7: $\mathbf{G} \leftarrow \mathbf{G}/max(1, \frac{|\mathbf{G}|}{R})$
8: $\mathbf{H}^i \leftarrow ComputeHistogram(\mathbf{G})$
9: $\mathbf{H}^i \leftarrow \mathbf{H}^i + \sum_j k_{ij} - \sum_j k_{ji}$
10: $\mathbf{H}^i \leftarrow \mathbf{H}^i + Lap(0, \frac{2R}{\varepsilon})$
11: Send $\mathbf{H}^i$ to the server

12: ***ServerAggregates***$(depth)$:
13: $\mathbf{H} \leftarrow \sum_i \mathbf{H}^i$
14: **if** $depth = d$ **then**
15: Compute leaf value by Equation (3)
16: **else**
17: Select the cut point with the maximum gain by Equation (4)
18: **end if**
19: Send the node information to the parties

20: ***PartyUpdate***$(P_i, node)$:
21: **if** $node$ is a split node **then**
22: Split the instance set to leaf and right
23: **else**
24: Update the prediction value for the instance set
25: **end if**

collect different features). Moreover, at least one party has the labels. For ease of presentation, we assume that party P_1 has the labels. Inspired by SecureBoost (Cheng et al., 2019), we adopt HE to protect the shared gradients to enable vertical FL. The algorithm is presented in Appendix A. At the beginning of training, the server generates the public and private keys for HE (lines 1-3). When computing the histogram (lines 4-20), the parties first update their gradients if they have the labels. For the parties that do not have the labels, the server sends the encrypted gradients to the parties. The histogram is computed based on the raw gradients or encrypted gradients. For the aggregation (lines 22-31), the server decrypts the encrypted histograms and concatenates all decrypted histograms to compute the split information or the leaf values, which are sent back to the parties. The parties use the received information to update the instance layout or the prediction value for the next round.

Privacy Analysis We analyze the privacy in horizontal and vertical FedTree as below.

Theorem 1. *The parties in Horizontal and Vertical FedTree satisfy the secure multi-party computation protocol, where all that the parties can learn is what they can learn from the final output (i.e., the final model) and their own input.*

Proof. During the whole training process, the parties only transfer the encrypted histograms or gradients to the server, and the server only transfers the model parameters to the parties. Thus, the parties can only know the final model. □

Theorem 2. *The server in Horizontal and Vertical FedTree only knows the aggregated histogram, which satisfies ε-DP.*

Proof. In both settings, the parties only transfer the histograms to the server. The local histogram satisfies ε-DP. In horizontal FedTree, since each party has disjoint data, due to the parallel composition property, the global histogram also satisfies ε-DP. In vertical FedTree, since each dimension has added the noises from the same distribution, the concatenated row also has injected noises from $Lap(0, \frac{2R}{\varepsilon})$. Thus, the global histogram also satisfies ε-DP. □

5 Optimization

FedTree designs the aggregation at a node-level to ensure the effectiveness of federation. However, the computation and communication overhead of federation is also at a node-level, which may be accumulated at a high frequency. Next, we introduce our optimization techniques to mitigate the computation and communication overhead.

5.1 Parallelism

Computing histograms is the most time-consuming part in the centralized GBDT training. Like centralized GBDT training (Chen & Guestrin, 2016; Ke et al., 2017; Wen et al., 2020), we apply node and feature level parallelism when computing the histograms as shown in Step 1 of Figure 5. Among different nodes in the same layer (i.e., depth), since they contain different instances, the computation of histograms between these nodes is independent and can be processed by different threads. Moreover, inside a node, the histograms of different features are also independent and can be computed in parallel by different threads. Ideally, the time complexity of computing histograms by using the node and feature level parallelism is $O(n_i \times B)$, while the time complexity without parallelism is $O(n_i \times B \times n_f \times 2^d)$, where n_i is the number of instances, n_f is the number of features, B is the maximum number of bins, and d is the current depth.

5.2 Layer-wise Batched Communition

Compared with neural networks with FedAvg that transfers the whole model each time, the communication in FedTree

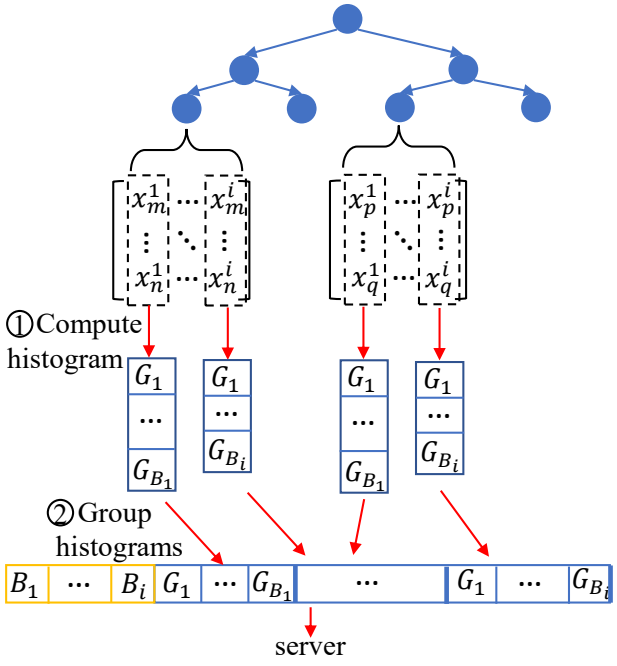

Figure 5. The computation and communication inside a party. Each red arrow represents a thread.

has two properties: 1) Frequent communication: Considering training 50 trees with a maximum depth of 6, there are about $50 \times (2^7 - 1) = 6350$ nodes in total, which introduces very high communication frequency. 2) Light communication: Histogram is usually much smaller than neural networks, which has a size of $O(B)$ and smaller than 10kB. Although the total communication size of FedTree is much smaller than FedAvg with moderate neural networks, the overhead for launching a gRPC communication is high.

To address the high overhead of launching the gPRC service, we propose layer-wise batching approach to reduce the communication times. As shown in Step 2 of Figure 5, we group the histograms into a sequence, and we add a sequence header to indicate the length of each independent histogram. While the additional communication cost of the header is negligible, the actual communication overhead can be significantly reduced.

5.3 CPU-GPU Codesign

Training of GBDT without encryption is efficient. When applying HE to encrypt the gradients for parties with missing labels, the HE operations will be computational-expensive. To alleviate the HE bottleneck, we utilize GPU to accelerate the HE computation. If a party needs to compute the encrypted gradients and has GPU, before the computation from gradients to histograms, it first transfers the raw gradients from CPU to GPU. Then, the computation for the local histogram on CPU and the computation for the encrypted gradients on GPU are conducted simultaneously. Due to the high number of available threads in GPU, the encryption operation can be accelerated significantly.

6 Evaluation

FedTree is implemented mainly in C++ and available on GitHub [1]. Next, we compare FedTree with other systems/approaches in aspects of model and system performance.

6.1 Experimental Setup

Datasets We conduct experiments on five public tabular datasets (breast, a9a, cod-rna, mnist, abalone) with classification and regression tasks downloaded from LIBSVM website[2]. The statistics of the datasets are shown in Appendix B of the supplementary material.

Baselines 1) FATE (Liu et al., 2021) v1.9.0: Besides FedTree, FATE is the only FL system that also supports both horizontal and vertical Federated GBDT to the best of our knowledge. 2) XGBoost (Chen & Guestrin, 2016): we train XGBoost in the centralized setting with all data. 3) SOLO: each party trains the trees locally without federation. 4) FEL: we adopt a tree-level aggregation (Zhao et al., 2018; Li et al., 2020a) to aggregate the trees locally trained by the parties. We do not compare with Secure-XGBoost (Law et al., 2020) as it requires secure hardware enclaves.

Settings We evaluate the performance of FedTree under both horizontal and vertical FL settings. By default, we do not apply DP to FedTree for a fair comparison. We set the number of trees to 50 and the maximum number of depth to 6. The learning rate is set to 0.1. The number of parties is set to 2 by default to simulate the cross-silo FL (FATE usually fails when the number of parties is larger than two in our experiments). We sample $q \sim Dir(\beta)$ and allocate a q_j proportion of the total data instances/features to P_j for horizontal/vertical FL, where β is equal to 0.5.

6.2 Standalone Simulation

For standalone simulation, we run experiments on a machine with four AMD EPYC 7543 32-Core Processors and four NVIDIA A100 GPUs to simulate the federated setting. By default, we only use CPUs in experiments (including FedTree) for fair comparison and we limit the number of cores for each experiment to 16.

[1] https://github.com/Xtra-Computing/FedTree

[2] https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/

Table 1. Accuracy comparison between different approaches. We report AUC for binary classification task (breast, a9a, and cod-rna), accuracy for multi-class classification task (mnist), and RMSE for regression task (abalone) on the test dataset. FATE cannot successfully finish training 50 trees for mnist dataset in horizontal FL setting.

Datasets	Centralized	Horizontal FL					Vertical FL				
	XGBoost	FedTree	FedTree+SA	FATE	SOLO	FEL	FedTree	FedTree+HE	FATE	SOLO	FEL
breast	1.0	1.0	1.0	0.995	0.999	1.0	1.0	1.0	1.0	0.944	0.988
a9a	0.902	0.902	0.902	0.902	0.883	0.896	0.902	0.902	0.902	0.608	0.573
cod-rna	0.993	0.993	0.993	0.992	0.968	0.977	0.993	0.993	0.993	0.667	0.754
mnist	0.983	0.983	0.983	✗	0.926	0.943	0.983	0.983	0.983	0.759	0.822
abalone	0.078	0.079	0.079	0.080	0.085	0.289	0.078	0.078	0.078	0.138	0.266

Table 2. The average training time (s) per tree of FedTree and FATE in the horizontal and vertical FL settings.

Datasets	Horizontal FL			Vertical FL		
	FedTree	FATE	Speedup	FedTree	FATE	Speedup
breast	0.117	1.97	**16.8x**	0.90	1.73	**1.9x**
a9a	0.289	18.41	**63.7x**	6.46	55.31	**8.6x**
cod-rna	0.388	25.84	**66.6x**	4.22	104.59	**24.8x**
mnist	8.102	529.49	**65.6x**	274.42	790.37	**2.9x**
abalone	0.075	1.55	**20.7x**	0.877	3.10	**3.5x**

Table 3. The average time (s) per tree. Here "FedTree" refers to vertical FedTree without parallelism and GPU acceleration. The speedup is computed by the improvement of applying both techniques.

Datasets	FedTree	+ Para	+ Para&GPU	Speedup
breast	1.75	0.9	0.85	**2.1x**
a9a	17.8	6.46	5.55	**3.2x**
cod-rna	43.6	4.22	1.86	**23.4x**
mnist	567.7	274.42	257.13	**2.2x**
abalone	13.48	0.877	0.753	**17.9x**

6.2.1 *Effectiveness*

We compare the model performance of FedTree with other approaches as shown Table 1. We use "FedTree" to denote the horizontal or vertical FedTree without additional privacy mechanisms. From the table, we have the following observations: 1) The accuracy of FedTree is the same as centralized training with XGBoost and much better than SOLO and FEL in most cases, which demonstrates that our node-level histogram-sharing scheme is effective and even lossless. 2) The privacy techniques including SA and HE do not introduce accuracy loss while protecting the local histograms and gradients.

6.2.2 *Efficiency*

We compare the average training time per tree of FedTree and FATE as shown in Table 2. Note that here we apply SA to Horizontal FedTree and HE to vertical FedTree to ensure that FedTree and FATE achieve the same privacy level. Thanks to the specialized design for trees, FedTree is much faster than FATE and can achieve up to 66x speedup. Thus, FedTree is very efficient while maintaining the lossless model compared with centralized training.

6.2.3 *Ablation Study*

We study the effect of our parallelism design and CPU-GPU co-design by adding each component sequentially. The results are shown in Table 3. From the table, we can observe that both techniques can improve the training speed well. Our parallelism design can well exploit the multi-core CPUs to speed-up training and GPU can further accelerate the training by reducing the encryption time.

6.2.4 *Differential Privacy*

We apply DP to further protect the histograms during training. We set the clipping threshold R to one and the second-order gradient to constant one (Li et al., 2020b). The results are shown in Table 4. Given a moderate privacy budget (e.g., 5), horizontal and vertical FedTree can achieve a close model performance with the non-differentially private version. In the future, we plan to integrate more advanced DP mechanisms (e.g., Gaussian mechanism) and privacy loss calculators (e.g., Renyi DP (Mironov, 2017)) into FedTree.

6.2.5 *Heterogeneity*

In the default setting, we simulate quantity non-IID by partitioning different numbers of samples into different parties. In this section, we study how the heterogeneity among data distributions of different parties affects the performance of FedTree. Following the partitioning strategies in (Li et al., 2021), we simulate label and feature imbalance to assess the

Table 4. The model performance (AUC for a9a and RMSE for abalone) of FedTree with differential privacy under different privacy budgets on two datasets.

Datasets	ε	Horizontal FedTree	Vertical FedTree
a9a	no DP	0.902	0.902
	1	0.792	0.811
	2	0.875	0.861
	5	0.890	0.888
abalone	no DP	0.079	0.078
	1	0.126	0.142
	2	0.103	0.097
	5	0.082	0.081

Table 5. The model performance (AUC for a9a and RMSE for abalone) under different non-IID data settings.

	non-IID type	breast	a9a	cod-rna	mnist	abalone
Horizontal FedTree	label	1	0.902	0.989	0.981	0.079
	feature	1	0.902	0.989	0.981	0.079
Vertical FedTree	label	1	0.902	0.993	0.983	0.078
	feature	1	0.902	0.993	0.983	0.079
XGBoost	centralized	1	0.902	0.993	0.983	0.078

robustness of FedTree. For label imbalance, given N parties, we sample $p_k \sim Dir_2(0.5)$ and allocate a $p_{k,j}$ proportion of the instances of class k to party j, where $Dir(0.5)$ is the Dirichlet distribution with concentration parameter 0.5. For feature imbalance, we add noises $\hat{\mathbf{x}} \sim Gau(i/20)$ for Party P_i, where $Gau(i/20)$ is a Gaussian distribution with mean 0 and variance $i/20$. The results are shown in Table 5, which further verifies that our system can well handle the non-IID setting. In our solution, FedTree merges all the local cut points as the global one which captures the statistics of local data of all parties even in a non-IID data setting. Although non-IID data may affect the computation of the approximated accumulated gradients in horizontal FedTree as shown in Equation (8), the selection of best cut point in GBDTs is robust against small changes in the gradients. Thus, FedTree can still maintain high accuracy in the non-IID data setting.

6.3 Distributed Computing

For distributed computing, we run experiments on a cluster with eight machines, where each machine has 2 * Intel Xeon E5-2680 v4 CPUs with 1Gbps Ethernet. For distributed computing, the model performance of FedTree is same as the standalone simulation so we mainly report the system performance here.

Table 6. The average time (s) per tree in the distributed setting.

Datasets	Horizontal FL			Vertical FL		
	FedTree	FATE	Speedup	FedTree	FATE	Speedup
breast	0.22	10.30	**46.8x**	1.49	4.72	**3.2x**
a9a	0.66	23.51	**35.6x**	8.03	29.13	**3.6x**
cod-rna	0.34	27.43	**80.7x**	9.99	52.50	**5.3x**
mnist	25.9	838.04	**32.4x**	22.77	506.42	**22.3x**
abalone	0.29	6.39	**22.0x**	2.06	5.39	**2.6x**

Table 7. The total communication time (s) of FedTree with or without layer-wise batched communication (LBC) technique and the communication size (MB).

Datasets	Horizontal FedTree				Vertical FedTree			
	w/o LBC	w LBC	speedup	size	w/o LBC	w LBC	speedup	size
breast	8.82	4.64	**1.9x**	20.4	16.301	5.26	**3.1x**	23.5
a9a	11.72	4.04	**2.9x**	14.2	21.56	6.16	**3.5x**	27.0
cod-rna	12.42	5.4	**2.3x**	33.4	55.46	14.22	**3.9x**	48.5
mnist	30.16	8.15	**3.7x**	57.3	80.26	16.72	**4.8x**	59.4
abalone	13.156	5.06	**2.6x**	32.9	13.61	5.67	**2.4x**	43.3

6.3.1 *Efficiency*

Table 6 presents the average time per tree of FedTree and FATE. In the distributed setting, FedTree still significantly outperforms FATE given the same privacy level, especially in the horizontal FL setting. FedTree can achieve up to 80 times speedup compared with FATE, which shows that FedTree is good at real federated scenarios.

6.3.2 *Communication Cost*

We further investigate the communication costs of FedTree in the distributed setting. We do not compare with FATE for the communication costs since the output log of FATE does not explicitly show the communication size or time. In Table 7, we present the total communication time of FedTree running for 50 trees with or without our layer-wise batched communication (LBC) technique. The results show that our LBC technique can effectively reduce communication costs. The overhead of launching gRPC service is even higher than the overhead of transferring the messages since the communication size is small in FedTree.

6.3.3 *Scalability*

We vary the number of parties from 8 to 32 to study the scalability of FedTree, where the parties are evenly distributed in different machines. The time is reported in Table 8. We do not compare with FATE since we find that FATE cannot successfully run with a large number of parties. When the number of parties increases, the local data size per party is small and FedTree is faster since the bottleneck of training is the computation. Since FedTree adopts a centralized

Table 8. The average time (s) per tree of FedTree with different numbers of parties. We use ✗ to mark the settings when some parties do not have data due to partitioning.

datasets	Horizontal FedTree			Vertical FedTree		
	8	16	32	8	16	32
breast	0.15	0.12	0.12	0.75	✗	✗
a9a	0.31	0.22	0.19	4.75	3.26	2.88
cod-rna	0.19	0.14	0.12	6.52	✗	✗
mnist	13.7	10.3	7.76	12.9	9.98	8.57
abalone	0.16	0.15	0.11	1.66	✗	✗

communication architecture, the total communication cost only linearly increases with the number of parties. Note that the model performance of FedTree is unchanged when increasing the number of parties. In summary, FedTree can scale well with the number of parties.

7 DISCUSSIONS

In this section, we discuss the limitations and privacy guarantees of FedTree.

7.1 Party Sampling and Dropping

We assume full-party participation in our algorithms. In particular situations, such as federated learning on edge devices, there may be a large number of parties, necessitating support for party sampling and the allowance for party dropping. For party sampling in Horizontal FedTree, we need to modify Line 9 of Algorithm 2 to only incorporate noises generated by each pair of selected clients. If party dropping occurs among the sampled parties in Horizontal FedTree, the injected noises cannot cancel out with each other when summing the local histograms, which results in an inaccurate global histogram. Then, we can simply discard the current round and not select the dropped parties in the next rounds. As for Vertical FedTree, Algorithm 3 remains functional in terms of party sampling and dropping as long as at least one party with labels participates in the round, allowing the server to obtain encrypted histograms. In the partial-party participation setting, since a subset of instances is used to update each node, the accuracy may be lower than in the full-party participation scenario. If full participation is not used at the beginning, the global cut points may not be optimal which affects the model performance.

7.2 Privacy Guarantees in FedTree

FedTree offers three levels of privacy protection by implementing various privacy preservation techniques: 1) L_0: exchanging raw local histograms between parties and the server; 2) L_1: sharing encrypted local histograms using cryptographic methods (secure aggregation in Hori-FedTree and homomorphic encryption in Verti-FedTree) among parties to maintain accuracy while adhering to secure multi-party computation protocols; 3) L_2: applying DP to local histograms before sharing to defend against inference attacks, thereby achieving a trade-off between privacy and accuracy by controlling the noise magnitude.

Regarding the L_0 configuration, like sharing raw models in FedAvg, sharing raw histograms in FedTree may expose information about training data. Histograms consist of cut points and accumulated gradients. As the purpose of cut points is to divide feature values into distinct bins evenly, they may disclose statistics concerning the distribution of these values. For example, given cut points $\langle 0.1, 0.2, 0.5 \rangle$, one could estimate that the number of instances with feature values in the range [0.1, 0.2] is approximately equal to those with feature values in the range [0.2, 0.5]. While the gradients are computed based on the current prediction values and the labels, it is challenging to infer the true labels from the accumulated gradients as the number of instances within a bin is unknown. In general, we suggest users use the L_0 configuration when the server and participants are trusted and the distribution of local data is not sensitive. The L_1 configuration serves as a suitable default for maintaining high model performance while safeguarding local data and its distribution when the server is trusted. L_2 is appropriate when the local data is highly sensitive and the server may conduct inference attacks based on the received messages.

8 CONCLUSION

In this paper, we present FedTree, the first comprehensive and specialized FL system for trees. Our experiments demonstrate the superiority of FedTree in aspects of model and system performance. We believe FedTree will advance tree-based FL research and deployment.

ACKNOWLEDGEMENTS

The authors thank Kevin Hsieh and MLSys reviewers for their constructive comments in improving the paper. This research is supported by the National Research Foundation Singapore and DSO National Laboratories under the AI Singapore Programme (AISG Award No: AISG2-RP-2020-018). Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not reflect the views of National Research Foundation, Singapore. Qinbin's work was mostly conducted during study at National University of Singapore.

REFERENCES

Fedlearner. https://github.com/bytedance/fedlearner. Accessed: 2022-10-25.

Abadi, M., Chu, A., Goodfellow, I., McMahan, H. B., Mironov, I., Talwar, K., and Zhang, L. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pp. 308–318, 2016.

Acar, A., Aksu, H., Uluagac, A. S., and Conti, M. A survey on homomorphic encryption schemes: Theory and implementation. *ACM Computing Surveys (Csur)*, 51(4):1–35, 2018.

Beutel, D. J., Topal, T., Mathur, A., Qiu, X., Parcollet, T., de Gusmão, P. P., and Lane, N. D. Flower: A friendly federated learning research framework. *arXiv preprint arXiv:2007.14390*, 2020.

Bonawitz, K., Ivanov, V., Kreuter, B., Marcedone, A., McMahan, H. B., Patel, S., Ramage, D., Segal, A., and Seth, K. Practical secure aggregation for federated learning on user-held data. *arXiv preprint arXiv:1611.04482*, 2016.

Bonawitz, K., Eichner, H., Grieskamp, W., Huba, D., Ingerman, A., Ivanov, V., Kiddon, C. M., Konečný, J., Mazzocchi, S., McMahan, B., Overveldt, T. V., Petrou, D., Ramage, D., and Roselander, J. Towards federated learning at scale: System design. In *SysML*, 2019. URL https://arxiv.org/abs/1902.01046.

Burges, C. J. From ranknet to lambdarank to lambdamart: An overview. *Learning*, 11(23-581):81, 2010.

Chen, T. and Guestrin, C. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pp. 785–794, 2016.

Cheng, K., Fan, T., Jin, Y., Liu, Y., Chen, T., Papadopoulos, D., and Yang, Q. Secureboost: A lossless federated learning framework. *arXiv preprint arXiv:1901.08755*, 2019.

Diffie, W. and Hellman, M. E. New directions in cryptography. In *Democratizing Cryptography: The Work of Whitfield Diffie and Martin Hellman*, pp. 365–390. 2022.

Dorogush, A. V., Ershov, V., and Gulin, A. Catboost: gradient boosting with categorical features support. *arXiv preprint arXiv:1810.11363*, 2018.

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

Fang, W., Zhao, D., Tan, J., Chen, C., Yu, C., Wang, L., Wang, L., Zhou, J., and Zhang, B. Large-scale secure xgb for vertical federated learning. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pp. 443–452, 2021.

Hard, A., Rao, K., Mathews, R., Ramaswamy, S., Beaufays, F., Augenstein, S., Eichner, H., Kiddon, C., and Ramage, D. Federated learning for mobile keyboard prediction. *arXiv preprint arXiv:1811.03604*, 2018.

He, C., Li, S., So, J., Zeng, X., Zhang, M., Wang, H., Wang, X., Vepakomma, P., Singh, A., Qiu, H., et al. Fedml: A research library and benchmark for federated machine learning. *arXiv preprint arXiv:2007.13518*, 2020.

Kairouz, P., McMahan, H. B., Avent, B., Bellet, A., Bennis, M., Bhagoji, A. N., Bonawitz, K., Charles, Z., Cormode, G., Cummings, R., et al. Advances and open problems in federated learning. *arXiv preprint arXiv:1912.04977*, 2019.

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., and Liu, T.-Y. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, 30:3146–3154, 2017.

Kim, S.-M., Pantel, P., Duan, L., and Gaffney, S. Improving web page classification by label-propagation over click graphs. In *Proceedings of the 18th ACM conference on Information and knowledge management*, pp. 1077–1086. ACM, 2009.

Law, A., Leung, C., Poddar, R., Popa, R. A., Shi, C., Sima, O., Yu, C., Zhang, X., and Zheng, W. Secure collaborative training and inference for xgboost. In *Proceedings of the 2020 workshop on privacy-preserving machine learning in practice*, pp. 21–26, 2020.

Li, Q., Wen, Z., and He, B. Practical federated gradient boosting decision trees. In *AAAI*, pp. 4642–4649, 2020a.

Li, Q., Wu, Z., Wen, Z., and He, B. Privacy-preserving gradient boosting decision trees. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 784–791, 2020b.

Li, Q., Diao, Y., Chen, Q., and He, B. Federated learning on non-iid data silos: An experimental study. *arXiv preprint arXiv:2102.02079*, 2021.

Li, T., Sahu, A. K., Talwalkar, A., and Smith, V. Federated learning: Challenges, methods, and future directions. *arXiv preprint arXiv:1908.07873*, 2019.

Liu, Y., Fan, T., Chen, T., Xu, Q., and Yang, Q. Fate: An industrial grade platform for collaborative learning with data protection. *J. Mach. Learn. Res.*, 22(226):1–6, 2021.

Long, G., Tan, Y., Jiang, J., and Zhang, C. Federated learning for open banking. In *Federated learning*, pp. 240–254. Springer, 2020.

McMahan, H. B., Moore, E., Ramage, D., Hampson, S., et al. Communication-efficient learning of deep networks from decentralized data. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, 2017.

Mironov, I. Rényi differential privacy. In *2017 IEEE 30th computer security foundations symposium (CSF)*, pp. 263–275. IEEE, 2017.

Paillier, P. Public-key cryptosystems based on composite degree residuosity classes. In *International conference on the theory and applications of cryptographic techniques*, pp. 223–238. Springer, 1999.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.

Pfitzner, B., Steckhan, N., and Arnrich, B. Federated learning in a medical context: a systematic literature review. *ACM Transactions on Internet Technology (TOIT)*, 21(2): 1–31, 2021.

Richardson, M., Dominowska, E., and Ragno, R. Predicting clicks: estimating the click-through rate for new ads. In *Proceedings of the 16th international conference on World Wide Web*, pp. 521–530. ACM, 2007.

Rieke, N., Hancox, J., Li, W., Milletari, F., Roth, H. R., Albarqouni, S., Bakas, S., Galtier, M. N., Landman, B. A., Maier-Hein, K., et al. The future of digital health with federated learning. *NPJ digital medicine*, 3(1):1–7, 2020.

Sanders, J. and Kandrot, E. *CUDA by example: an introduction to general-purpose GPU programming*. Addison-Wesley Professional, 2010.

Wen, Z., Liu, H., Shi, J., Li, Q., He, B., and Chen, J. ThunderGBM: Fast GBDTs and random forests on GPUs. *Journal of Machine Learning Research*, 21, 2020.

Wu, Y., Cai, S., Xiao, X., Chen, G., and Ooi, B. C. Privacy preserving vertical federated learning for tree-based models. *arXiv preprint arXiv:2008.06170*, 2020.

Yang, Q., Liu, Y., Chen, T., and Tong, Y. Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 10(2):1–19, 2019.

Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., and Stoica, I. Spark: Cluster computing with working sets. In *2nd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 10)*, 2010.

Zhao, L., Ni, L., Hu, S., Chen, Y., Zhou, P., Xiao, F., and Wu, L. Inprivate digging: Enabling tree-based distributed data mining with differential privacy. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, pp. 2087–2095. IEEE, 2018.

A VERTICAL FEDTREE

The vertical FedTree algorithm is presented at Algorithm 3.

Algorithm 3 Vertical FedTree

input Parties $P = \{P_1, ..., P_N\}$,, server S, threshold R, number of trees T, tree depth d

***Init**(P, S):*
S generates k_{pub}, k_{pri} with Paillier cryptosystem
S sends k_{pub} to P_1

***PartyComputeHistogram**(P_i)*
//conducted on party P_i:
if Party P_i has label **then**
  $\mathbf{G} \leftarrow UpdateGradients()$
  $\mathbf{G} \leftarrow \mathbf{G}/max(1, \frac{|\mathbf{G}|}{R})$
  $\mathbf{H}^i \leftarrow ComputeHistogram(\mathbf{G}, S)$
  $\mathbf{H}^i \leftarrow \mathbf{H}^i + Lap(0, \frac{2R}{\varepsilon})$
  Send $\mathbf{H}^i$ to the server
  if $i == 1$ **then**
    $||\mathbf{G}|| \leftarrow Enc(\mathbf{G}, k_{pub})$
    send $||\mathbf{G}||$ to the server
  end if
else
  Query $||\mathbf{G}||$ and k_{pub} from the server
  $\|\mathbf{H}^i\| \leftarrow ComputeHistogram(\|\mathbf{G}\|, S)$
  $\|\mathbf{H}^i\| \leftarrow \|\mathbf{H}^i\| + \|Lap(0, \frac{2R}{\varepsilon})\|$
  Send $\|\mathbf{H}^i\|$ to the server
end if

ServerAggregates(depth):
for each received encrypted $\|\mathbf{H}^i\|$ **do**
  $\mathbf{H}^i \leftarrow Decrypt(\|\mathbf{H}^i\|, k_{pri})$
end for
$\mathbf{H} \leftarrow \cup_{i=1}^{n} \mathbf{H}^i$
if $depth = d$ **then**
  Compute leaf value by Equation (3)
  Send the prediction value of the instances to parties
else
  Select the cut point with the maximum gain by Equation (4)
  Send the instance ID split information to parties
end if

***PartyUpdate**(P_i, $node$)*:
if $node$ is a split node **then**
  Split the instance set to leaf and right
else
  Update the prediction value for the instance set
end if

B STATISTICS OF DATASETES

The statistics of the datasets used in the experiments are shown in Table 9.

Table 9. Statistics of datasets in the experiments.

Datasets	#Instances	#Features	Task
breast	683	10	binary classification
a9a	32,561	123	
cod-rna	59,535	8	
mnist	60,000	780	multiclass classification
abalone	4,177	8	regression

C ARTIFACT APPENDIX

C.1 Abstract

In this section, we describe the instructions to use our system to run the experiments.

C.2 Artifact check-list (meta-information)

Obligatory. Use just a few informal keywords in all fields applicable to your artifacts and remove the rest. This information is needed to find appropriate reviewers and gradually unify artifact meta information in Digital Libraries.

- **Program: C++**
- **Compilation: CMake**
- **Data set: a9a**
- **Run-time environment: Ubuntu 22.04**
- **Hardware: a server with four AMD EPYC 7543 32-Core Processors and four NVIDIA A100 GPUs.**
- **How much time is needed to prepare workflow (approximately)?: Half an hour**
- **Publicly available?: Yes.**
- **Code licenses (if publicly available)?: Apache License 2.0**
- **Data licenses (if publicly available)?: Creative Commons Attribution 4.0 International (CC BY 4.0) license**

C.3 Description

C.3.1 How delivered

We provide the open-sourced code at https://github.com/Xtra-Computing/FedTree. Also, we provide the documentation at https://fedtree.readthedocs.io/.

C.3.2 Hardware dependencies

FedTree is able to run on any hardware. Nvidia GPUs that support CUDA are required if using GPU-accelerated encryption in FedTree.

C.3.3 Software dependencies

FedTree requires cmake[3] $\geq$ 3.15, GMP[4], and NTL[5]. gRPC[6] is required if using a distributed version of FedTree.

C.3.4 Data sets

We use public datasets in our experiments. All datasets can be downloaded from LIBSVM website https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/. We provide a9a dataset in our repository[7].

C.4 Installation

We provide detailed installation instructions in our documentation https://fedtree.readthedocs.io/en/latest/Installation.html.

C.5 Experiment workflow

After installing FedTree, users need to write a configuration file to specify the parameters to run FedTree. Below is an example to run horizontal FedTree on a9a dataset, which is also provided in our repository[8].

```
data=./dataset/adult/a9a_horizontal_p0,./
    ↪ dataset/adult/a9a_horizontal_p1
test_data=./dataset/adult/
    ↪ a9a_horizontal_test
n_parties=2
mode=horizontal
privacy_tech=sa
model_path=fedtree.model
learning_rate=0.1
max_depth=6
n_trees=50
partition=0
objective=binary:logistic
n_features=123
gamma=0.001
lambda=0.1
max_num_bin=64
min_child_weight=0
```

Then, you can simply run FedTree with the example configuration file as follows.

```
./build/bin/FedTree-train examples/adult
    ↪ /a9a_horizontal_simulation.conf
```

C.6 Evaluation and expected result

If running the provided script, you are expected to see 'AUC = 0.902442' in the last line of the output.

[3] https://cmake.org/
[4] https://gmplib.org/
[5] https://libntl.org/
[6] https://grpc.io/docs/languages/cpp/quickstart/
[7] https://github.com/Xtra-Computing/FedTree/tree/main/dataset/adult
[8] https://github.com/Xtra-Computing/FedTree/blob/main/examples/adult/a9a_horizontal_simulation.conf