NodeGuard: A Highly Efficient Two-Party Computation Framework for Training Large-Scale Gradient Boosting Decision Tree

Tianxiang Dai
Huawei European Research Center, Germany
Email: tianxiangdai@huawei.com

Yufan Jiang
Huawei European Research Center, Germany
Karlsruhe Institute of Technology, Germany
Email: yufan.jiang@{huawei.com, partner.kit.edu}

Yong Li ✉
Huawei European Research Center, Germany
Email: yong.li1@huawei.com

Fei Mei
Huawei European Research Center, Germany
Email: fei.mei@huawei.com

***Abstract*—The Gradient Boosting Decision Tree (GBDT) is a well-known machine learning algorithm, which achieves high performance and outstanding interpretability in real-world scenes such as fraud detection, online marketing and risk management. Meanwhile, two data owners can jointly train a GBDT model without disclosing their private dataset by executing secure Multi-Party Computation (MPC) protocols. In this work, we propose NodeGuard, a highly efficient two-party computation (2PC) framework for large-scale GBDT training and inference. NodeGuard guarantees that no sensitive intermediate results are leaked in the training and inference. The efficiency advantage of NodeGuard is achieved by applying a novel keyed bucket aggregation protocol, which optimizes the communication and computation complexity globally in the training. Additionally, we introduce a probabilistic approximate division protocol with an optimization for re-scaling, when the divisor is publicly known. Finally, we compare NodeGuard to state-of-the-art frameworks, and we show that NodeGuard is extremely efficient. It can improve the privacy-preserving GBDT training performance by a factor of 5.0 to 131 in LAN and 2.7 to 457 in WAN.**

***Index Terms*—MPC, Two-party computation, Gradient boosting decision tree, Secure bucket aggregation**

1. Introduction

As a well-known machine learning algorithm, Gradient Boosting Decision Tree [1] and its variant XGBoost [2] are widely used for regression and classification tasks. A lot of works have successfully integrated the GBDT algorithm to real-world scenarios such as fraud detection [3], recommendation system [4], online marketing [5], [6] and risk management [7]. To achieve a better performance, one can imagine that a model owner collaborates with some other data owners to train the model with an expanded dataset. Due to the user privacy policies and laws such as GDPR [8], it might be inappropriate for companies or organizations to exchange data in plaintext.

The federated learning (FL) paradigm was proposed by Google [9] where numerous data owners jointly train a machine learning model without directly revealing their private data. However, some works [10], [11], [12], [13] have already showed that the leakage of intermediate results in FL is problematic. This issue is addressed by secure multi-party computation (MPC) [14], [15], [16], which allows multiple parties to jointly compute a function and reveals only the pre-defined function output. Recent works such as [17], [18], [19], [20] use MPC protocols to construct a privacy-preserving GBDT framework. While some cryptographic operations are executed via plaintext in SecureBoost [17], both Pivot [18] and Squirrel [20] avoid executing critical plaintext computations by applying mixed cryptographic primitives. However, they all rely on a computationally expensive homomorphic encryption (HE) scheme in exchange for a communication efficiency. Meanwhile, [19] relies on a trusted third party in the preprocessing stage in order to generate correlated randomness for its online stage.

Our Contribution. In this work, we introduce NodeGuard, a two-party computation framework for privacy-preserving GBDT training on a distributed large-scale dataset. We summarize our contribution as follows:

- *A novel two-party bucket aggregation protocol.* The protocol does not rely on a computationally inefficient HE-based scheme, but achieves an even better communication efficiency. We then provide the ideal functionality $\mathcal{F}_{\mathsf{2PC}}^{\mathsf{KeyBuc}}$ implemented by our protocol under the universally composable framework [21].
- *A probabilistic approximate division protocol.* We extend the truncation protocol proposed in [22] to a probabilistic approximate division protocol and prove that it is also valid for a (public) negative divisor. We further improve the accuracy by applying a signed/unsigned conversion to avoid bad events mentioned in [23].
- *A comprehensive evaluation.* We compare NodeGuard with state-of-the-art secure GBDT frameworks and provide an in-depth evaluation in Section 7. We test the efficiency of each framework in both LAN and WAN network

settings, and NodeGuard stands out with an improvement by a factor of up to 131 in LAN and up to 457 in WAN.

2. Related Work

Federated learning. The concept of federated learning (FL) was proposed by Google in 2016 [9]. Depending on how the data is distributed, federated learning can be categorized as horizontal (HFL), vertical (VFL) and transfer learning (TFL) [24]. During a learning process [25], the intermediate results such as gradients are exchanged among parties instead of the private data holdings. Although the private input remains local, recent works [10], [11], [12], [13] showed that the leaked intermediate information can be used to infer the private sensitive data.

Privacy-Preserving GBDT. The first privacy-preserving tree was done by Lindell et al. [26] on a horizontally partitioned dataset. The following works such as [27], [28] continue this line of work. Recently, both Pivot [18] and Squirrel [20] are designed to train GBDT on a vertically partitioned dataset using mixed cryptographic primitives. Some other works [29], [30], [31] also consider to make use of a trusted execution environment (TEE) to accelerate the training process. As already mentioned by [17], [18], [19], [20] and some optimizations [32], [33], the secure bucket aggregation is the most costly computation during the privacy-preserving GBDT training process. While [26], [34] choose to use sharing-based protocols, Fang et al. [19] introduces a permutation-based approach, and other frameworks such as SecureBoost [17], Pivot [18] and Squirrel [20] implement the HE-based bucket aggregation protocol.

Differential Privacy. By using Differential Privacy (DP) [35], [36] to protect the privacy of users, a calibrated noise is added to the data so that any individual identity is indistinguishable from others. However, SecureBoost [17] has pointed out that applying DP can only protect private user data to a certain degree. And Fletcher et al. [37] further shows that the prediction accuracy will be significantly affected by using DP.

For more details regarding privacy-preserving tree-based model learning, we refer to a survey [38].

3. Preliminaries

Fixed-Point Computation. Suppose $\tilde{x} \in \mathbb{R}$, we let p denote the fraction precision and $\lfloor \cdot \rfloor$ denote round down of the fraction part. We define a fixed-point value x as a ℓ bit integer using two's complement, and let x be $\tilde{x}$'s fixed-point representation on $\mathbb{Z}_{2^\ell}$, where $x = \lfloor \tilde{x} \cdot 2^p \rfloor$. Then x consists of $\ell - p$ bits integer part and p bits fraction part.

Secret Sharing. In 2PC case, a secret value $x \in \mathbb{Z}_{2^\ell}$ is *2-out-of-2* shared between parties, if $x = x_1 + x_2$ where $x_1, x_2 \in \mathbb{Z}_{2^\ell}$. We denote such a sharing scheme as $[\cdot]$. Addition and scalar-multiplication can be locally computed. Given a *Beaver's Triple* [16], [39], [40], multiplication of two shared values $[x]$ and $[y]$ requires parties to interact. We apply the Goldschmidt-Division algorithm [41] to compute the reciprocal $[1/y]$. The secure argmax computation is realized by executing multiple MPC comparison protocols [40] over the inputs using a binomial tree. The nonlinear sigmoid function is approximately computed by evaluating a piece-wise linear function [42] using above operations.

Threat Model. Our protocol is secure against semi-honest adversaries. We follow the universally composable framework (UC) described in [21], and we introduce our work based on the composition theorem provided in [21].

Review of Gradient Boosting Tree. Given a dataset $\mathbf{X} \in \mathbb{R}^{N \times F}$, where N denotes the sample amount and F denotes the feature amount, a GBDT model sequentially trains T decision trees. We assume that all trees are perfect binary trees. If D denotes the depth constraint, we have $n_{\mathsf{non}} = 2^D - 1$ non-leaf nodes and $n_{\mathsf{leaf}} = 2^D$ leaf nodes for each tree. By applying the approximate split finding algorithm [2], [43], we let $\mathcal{I}^{f,b}$ denote the indices of samples sorted into the b-th bucket of a certain feature f. Samples are sorted into B buckets based on the split points of a feature, which results overall $F \cdot (B-1)$ split candidates.

We use the node split metric of XGBoost as an example. [1] XGBoost uses first order and second order gradient statistics of the chosen loss function (L) to split a tree node:

$$g_i = \partial_{\hat{y_i}^{(t-1)}} L(y_i, \hat{y_i}^{(t-1)}), \quad h_i = \partial^2_{\hat{y_i}^{(t-1)}} L(y_i, \hat{y_i}^{(t-1)}) \tag{1}$$

where $\hat{y_i}^{(t-1)}$ represents the prediction result for a sample $\mathbf{x}_i$ at the previous tree building iteration, and y_i denotes the real label. $\hat{y_i}^0$ is set to $basescore$ at the beginning.

The gradient and hessian sums of one bucket are:

$$G^{f,b} = \sum_{i \in \mathcal{I}^{f,b}} g_i, \; H^{f,b} = \sum_{i \in \mathcal{I}^{f,b}} h_i. \tag{2}$$

Then the gradient sum of i-th candidate in a feature is defined as: $G_L = \sum_{b \leq i} G^{f,b}, \quad G_R = \sum_{i < b \leq B} G^{f,b}$. The same goes for the hessian sum.

Finally, the best split feature and the threshold value are determined by comparing and finding the maximum gain over all split candidates at each tree node:

$$Gain = \frac{1}{2}\left(\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda}\right) - \gamma, \tag{3}$$

where λ and γ are regularization parameters.

Once the best split information of a tree node is found, samples are allocated into the child nodes, which updates $\mathcal{I}^{f,b}$ at that child node. The above algorithm will be performed repeatedly at each child node until the depth D is reached. Then leaf weight is computed as:

$$\omega = -\frac{\sum_{i \in \mathcal{I}_{\mathsf{leaf}}} g_i}{\sum_{i \in \mathcal{I}_{\mathsf{leaf}}} h_i + \lambda}, \tag{4}$$

where $\mathcal{I}_{\mathsf{leaf}}$ denotes the indices of samples falling into one leaf node.

1. Our work is also applicable in other tree variants such as the classification and regression tree (CART) [44].

Protocol Π_{KeyBuc}

Private inputs: For each bucket b of each feature f, $\mathbf{P_h}$ holds $\mathcal{I}^{f,b}$, where $|\mathcal{I}^{f,b}| \leq N, 1 \leq b \leq B$ and $1 \leq f \leq F_h$. At a node j, parties hold $[\mathbf{g}_j]$, where $\mathbf{g}_j = (g_{j,1}, ..., g_{j,N})$.
Public inputs: Public parameters.
Outputs: For each bucket b of each feature f, parties output $[z_j^{f,b}]$, where $z_j^{f,b} = \sum_{i \in \mathcal{I}^{f,b}} g_{j,i}$.
Preprocessing:

- $\mathbf{P_h}$ sends $(\mathsf{BucSGen}, j, \mathbf{P_h}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{2PC}}^{\mathsf{Pre}}$, for each bucket b of each feature f:
  - If $j = 1$, receives $\mathbf{r}^{f,b}$, where $\mathbf{r}^{f,b} = (r_1^{f,b}, ..., r_N^{f,b})$.
  - Receives $[m_j^{f,b}]_h$, where $m_j^{f,b} = \mathbf{k}^{t,j} \cdot \mathbf{r}^{f,b}$.
- $\mathbf{P_{1-h}}$ sends $(\mathsf{BucSGen}, j, \mathbf{P_{1-h}}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{2PC}}^{\mathsf{Pre}}$:
  - Receives $\mathbf{k}_j$, where $\mathbf{k}_j = (k_{j,1}, ...k_{j,N})$.
  - For each bucket b of each feature f receives $[m_j^{f,b}]_{1-h}$.

Protocol:

1. If $j = 1$, for each bucket b of each feature f:
   - $\mathbf{P_h}$ computes a secret vector $\mathbf{s}^{f,b} = (s_1^{f,b}, ..., s_N^{f,b})$, where $s_i^{f,b} = 1$ if $i \in \mathcal{I}^{f,b}$ and $s_i^{f,b} = 0$ otherwise.
   - $\mathbf{P_h}$ computes $\mathbf{v}^{f,b} = \mathbf{s}^{f,b} - \mathbf{r}^{f,b}$, then sends $\mathbf{v}^{f,b}$ to $\mathbf{P_{1-h}}$.
2. $\mathbf{P_{1-h}}$ computes $\mathbf{q}^{t,j} = [\mathbf{g}_j]_{1-h} - \mathbf{k}_j$, sends $\mathbf{q}_j$ to $\mathbf{P_h}$.
3. Parties locally output for each bucket b of each feature f:
   - $\mathbf{P_h}$ sets $[z_j^{f,b}]_h = \sum_{i \in \mathcal{I}^{f,b}} q_{j,i} + \sum_{i \in \mathcal{I}^{f,b}} [g_{j,i}]_h + [m_j^{f,b}]_h$.
   - $\mathbf{P_{1-h}}$ sets $[z_j^{f,b}]_{1-h} = \mathbf{k}_j \cdot \mathbf{v}^{f,b} + [m_j^{f,b}]_{1-h}$.

Figure 1. Two-party keyed bucket aggregation protocol

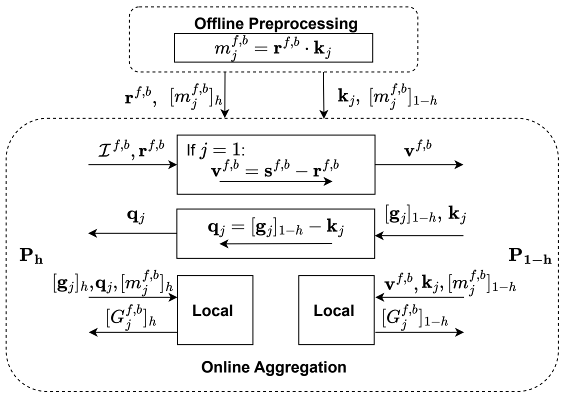
Private GBDT Training. In this paper, we focus on the **vertical federated learning** (VFL) setting, where $\mathbf{P_0}$ holds a dataset $\mathbf{X}_0 \in \mathbb{R}^{N \times F_0}$ and $\mathbf{P_1}$ holds the same samples with different feature space $\mathbf{X}_1 \in \mathbb{R}^{N \times F_1}$, such that $\mathbf{X} = \mathbf{X}_0 || \mathbf{X}_1$. Without loss of generality, we let $\mathbf{P_0}$ hold the whole label vector $\mathbf{Y} \in \mathbb{R}^N$. All resources (e.g. $\mathbf{X}$ and $\mathbf{Y}$) and sensitive intermediate values (e.g. bucket distribution $\mathcal{I}^{f,b}$, gradient, etc.) are not revealed. The node split information is only visible to party $\mathbf{P_h}$ ($\mathbf{h} \in \{0, 1\}$), who holds the best split candidate. The other party $\mathbf{P_{1-h}}$ knows nothing more than the bit $\mathbf{h}$. We achieve this using the same method as others [18], [19], [20], [26]: given $\mathbf{h}$, parties reveal the split feature identifier to $\mathbf{P_h}$. It is still an open question whether we could even hide the $\mathbf{h}$.

We use a shared indicator $[\mathbf{u}]$ to hide the node sample space after splitting, where $u_i = 1$ indicates that a sample $\mathbf{x}_i$ is allocated to left child node. In the private training process, we update the shared gradient at each child node by computing $[\mathbf{g}_{2j}] = [\mathbf{g}_j] \odot [\mathbf{u}]$ and $[\mathbf{g}_{2j+1}] = [\mathbf{g}_j] - [\mathbf{g}_{2j}]$ instead of "allocating" samples to the child node. Thus, parties do not have to update the local indices $\mathcal{I}^{f,b}$.

4. Keyed Bucket Aggregation

In this section, we formally describe the keyed bucket aggregation protocol Π_{KeyBuc} in Fig. 1 and demonstrate the protocol in Fig. 2.

At any node j, parties hold a freshly computed $[\mathbf{g}_j]$, where $\mathbf{g}_j = \{g_{j,1}, ..., g_{j,N}\}$. They are willing to compute the sum $G^{f,b}$ of each bucket b of each feature f. The feature



Offline Preprocessing
$m_j^{f,b} = \mathbf{r}^{f,b} \cdot \mathbf{k}_j$
$\mathbf{r}^{f,b}$, $[m_j^{f,b}]_h$
$\mathbf{k}_j$, $[m_j^{f,b}]_{1-h}$
$\mathcal{I}^{f,b}, \mathbf{r}^{f,b}$
If $j = 1$:
$\mathbf{v}^{f,b} = \mathbf{s}^{f,b} - \mathbf{r}^{f,b}$
$\mathbf{v}^{f,b}$
$\mathbf{q}_j$
$\mathbf{q}_j = [\mathbf{g}_j]_{1-h} - \mathbf{k}_j$
$[\mathbf{g}_j]_{1-h}$, $\mathbf{k}_j$
$\mathbf{P_h}$
$\mathbf{P_{1-h}}$
$[\mathbf{g}_j]_h, \mathbf{q}_j, [m_j^{f,b}]_h$
$[G_j^{f,b}]_h$
Local
Local
$\mathbf{v}^{f,b}, \mathbf{k}_j, [m_j^{f,b}]_{1-h}$
$[G_j^{f,b}]_{1-h}$
Online Aggregation

Figure 2. A demonstration of keyed bucket aggregation protocol Π_{KeyBuc}

owner (say $\mathbf{P_h} \in \{\mathbf{P_0}, \mathbf{P_1}\}$) who knows the distribution of this feature holds indices $\mathcal{I}^{f,b}$, where each $n \in \mathcal{I}^{f,b}$ indicates that a sample $\mathbf{x}_i$ belongs to the bucket b of feature f. Such $\mathcal{I}^{f,b}$ can also be presented as an indicator vector $\mathbf{s}^{f,b}$, where $\mathbf{s}^{f,b} = (s_1^{f,b}, ..., s_N^{f,b})$ and $s_i^{f,b} = 1$ if $i \in \mathcal{I}^{f,b}$ and $s_i^{f,b} = 0$ otherwise.

Preprocessing Stage. For each bucket b of each feature f, we let $\mathbf{P_h}$ hold $\mathbf{r}^{f,b}$ to mask its secret $\mathbf{s}^{f,b}$, where $\mathbf{r}^{f,b} = (r_1^{f,b}, ..., r_N^{f,b})$. At any node j (including the initial node), we let $\mathbf{P_{1-h}}$ hold $\mathbf{k}_j$ to mask its secret share $[\mathbf{g}_j]_{1-h}$. We call $\mathbf{k}_j$ the key of the node j. Then for each bucket b of each feature f at that node, both parties receive $[m_j^{f,b}]$ where $m_j^{f,b} = \mathbf{k}_j \cdot \mathbf{r}^{f,b}$, which is used to unmask the final result. The functionality $\mathcal{F}_{\mathsf{2PC}}^{\mathsf{Pre}}$ in Appendix A can be implemented by a trusted party which hands over all outputs. It can also be realized by a two-party multiplication protocol.

Online Stage. At the initial node, $\mathbf{P_h}$ sends a masked vector $\mathbf{v}^{f,b}$ for each bucket b of each feature f to $\mathbf{P_{1-h}}$, where $\mathbf{v}^{f,b} = \mathbf{s}^{f,b} - \mathbf{r}^{f,b}$. Note that this message is sent only once in the entire GBDT training process regardless of the number of non-leaf nodes and the number of trees. Then at any node j (including the initial node), $\mathbf{P_{1-h}}$ masks its private input $[\mathbf{g}_j]_{1-h}$ with $\mathbf{k}_j$ by computing $\mathbf{q}_j = [\mathbf{g}_j]_{1-h} - \mathbf{k}_j$ and sends $\mathbf{q}_j$ to $\mathbf{P_h}$. Now, $\mathbf{P_h}$ is able to compute the sum of the corresponding $[g_{j,i}]_h$ and $q_{j,i}$, where $i \in \mathcal{I}^{f,b}$ for each bucket b of each feature f. Without knowing $\mathcal{I}^{f,b}$, $\mathbf{P_{1-h}}$ simply computes $\mathbf{k}_j \cdot \mathbf{v}^{f,b}$. Finally, both parties unmask their (local) shares by adding $[m_j^{f,b}]$.

4.1. Complexity Analysis

An abbreviated summary is placed in Table 1. We observe that Π_{KeyBuc} already provides a solution with minimum communication rounds compared to all other protocols. Although the communication rounds required in Π_{IndiBuc} (Indicator-based) is the same as in Π_{KeyBuc}, the communication overhead in Π_{KeyBuc} is strictly less than in Π_{IndiBuc}. Compared to Π_{PermBuc} (Permutation-based), a communication advantage of the proposed Π_{KeyBuc} holds, if $T\, n_{\mathsf{non}} > \frac{FB}{2(F-1)}$ and approximately $T\, n_{\mathsf{non}} > \frac{B}{2}$. Com-

TABLE 1. COMPARISON OF NodeGuard TO THE EXISTING SECURE TWO-PARTY BUCKET AGGREGATION PROTOCOLS REGARDING COMMUNICATION COST (BITS). IN [20], q IS SET TO 2^{109} FOR THE SECURITY GUARANTEE IN LWE. ℓ IS SET TO 64.

Framework	Protocol	Rounds	Overhead
Xie. et al. [34]	Indicator	$T\, n_{\mathsf{non}}$	$4TFBN\ell\, n_{\mathsf{non}}$
Fang et al. [19]	Permutation	$2T\, n_{\mathsf{non}}$	$2TFN\ell\, n_{\mathsf{non}}$
Squirrel [20]	LWE-HE	$2T\, n_{\mathsf{non}}$	$> 2TN\, n_{\mathsf{non}} \log_2 q$
NodeGuard	Sharing	$T\, n_{\mathsf{non}}$	$FBN\ell + 2TN\ell\, n_{\mathsf{non}}$

Ideal Functionality $\mathcal{F}_{\mathsf{2PC}}^{\mathsf{KeyBuc}}$

Private inputs: Let $\mathcal{P} = \{\mathbf{P_h}, \mathbf{P_{1-h}}\}$. For each bucket b of each feature f, $\mathbf{P_h}$ holds indices $\mathcal{I}^{f,b}$, where $1 \leq b \leq B$ and $1 \leq f \leq F_h$. Additionally, at a node j of a tree t, $\mathbf{P_h}$ holds $[\mathbf{g}_j]_h$ and $\mathbf{P_{1-h}}$ holds $[\mathbf{g}_j]_{1-h}$, where $\mathbf{g}_j = (g_{j,1}, ..., g_{j,N})$.
Public inputs: Public parameters.
Internal state: $\mathbf{ready} \in \{\mathsf{true}, \mathsf{false}\}$.
Outputs: For each bucket b of each feature f, $\mathbf{P_h}$ receives $[z_j^{f,b}]_h$ and $\mathbf{P_{1-h}}$ receives $[z_j^{f,b}]_{1-h}$, where and $z_j^{f,b} = \sum_{i \in \mathcal{I}^{f,b}} g_{j,i}$.
Initialization: Set $\mathbf{ready} = \mathsf{false}$.
Compute:

- Upon receiving $\mathcal{I}^{f,b}$ from $\mathbf{P_h}$ for each bucket b of each feature f, check whether **ready** is set to **false**:
  - If yes, record all $\mathcal{I}^{f,b}$, set $\mathbf{ready} = \mathsf{true}$.
  - Otherwise, send $(\mathsf{failed}, \mathbf{P_h}, \mathsf{sid})$ to $\mathbf{P_h}$.
- Upon receiving $(\mathsf{Comp}, [\mathbf{g}_j]_i, \mathbf{P_i}, \mathsf{sid})$ from each $\mathbf{P_i} \in \mathcal{P}$ at a node j, and $[z_j^{f,b}]_c$ for each bucket b of each feature f at that node j from $\mathcal{S}$ corrupting $\mathbf{P_c}$:
  1. Compute $\mathbf{g}_j = [\mathbf{g}_j]_h + [\mathbf{g}_j]_{1-h}$.
  2. For each bucket b of each feature f, compute $z_j^{f,b}$, where $z_j^{f,b} = \sum_{i \in \mathcal{I}^{f,b}} g_{j,i}$. Then compute $[z_j^{f,b}]_{1-c} = z_j^{f,b} - [z_j^{f,b}]_c$.
  3. For each bucket b of each feature f, send $([z_j^{f,b}]_i, \mathbf{P_i}, \mathsf{sid})$ to $\mathbf{P_i} \in \mathcal{P}$.

Figure 3. Two-party Functionality $\mathcal{F}_{\mathsf{2PC}}^{\mathsf{KeyBuc}}$

pared to Π_{HEBuc} (HE-based), we omit the communication $\mathcal{O}(2TFB\, n_{\mathsf{non}} \log q)$ of Π_{HEBuc} for simplicity. The condition becomes $T\, n_{\mathsf{non}} > \frac{FB\,\ell}{2(\log_2 q - \ell)}$. We end up with a conclusion that Π_{KeyBuc} achieves a better communication performance, when the convergence of a GBDT model requires numerous large-depth trees. Due to default training parameters suggested by [45], the above conditions are easily satisfied.

4.2. Theorem and Ideal Functionality

We provide the Theorem 1 and the ideal functionality $\mathcal{F}_{\mathsf{2PC}}^{\mathsf{KeyBuc}}$ in this section, and we place a proof sketch in Appendix B:

Theorem 1. Protocol Π_{KeyBuc} shown in Fig. 1 UC-realizes $\mathcal{F}_{\mathsf{2PC}}^{\mathsf{KeyBuc}}$ described in Fig. 3 in the $\mathcal{F}_{\mathsf{2PC}}^{\mathsf{Pre}}$-hybrid model, in the presence of a semi-honest adversary who can corrupt $\mathbf{P_i}$ where $\mathbf{P_i} \in \{\mathbf{P_h}, \mathbf{P_{1-h}}\}$, with static corruption.

5. Generalized Division by a Public Value

In this paper, we extend the truncation protocol in [22] by replacing the public divisor with y, where $y \in (0, 2^{\ell_x}] \cup [2^\ell - 2^{\ell_x}, 2^\ell)$ in the field $\mathbb{Z}_{2^\ell}$ (y is considered to be a power of two and positive in [22]). In short, we prove that for a large enough field, the reconstructed result after a local division, with high probability, is at most 1 off from the correct result x/y. Note that if a decimal number x is negative, we represent it in the field as $2^\ell - \lfloor |x| \rfloor$, and the division result x/y becomes $2^\ell - \lfloor |x|/y \rfloor$ (in case y is positive). We let $\mathsf{Rec}()$ denote the reconstruction function. The proof of Theorem 2 can be found in Appendix C.

Theorem 2. In field $\mathbb{Z}_{2^\ell}$, let $x \in [0, 2^{\ell_x}] \cup [2^\ell - 2^{\ell_x}, 2^\ell)$, where $l > \ell_x + 1$ and given shares $[x]_1$, $[x]_2$ of a shared $[x]$. Let $z \in \mathbb{Z}_{2^\ell}$, where $z = \lfloor x/y \rfloor$. If $y \in (0, 2^{\ell_x}]$, let $[z]_1 = [x]_1/y$ and $[z]_2 = 2^\ell - (2^\ell - [x]_2)/y$. If $y \in [2^\ell - 2^{\ell_x}, 2^\ell)$, let $[z]_1 = 2^\ell - [x]_1/(2^\ell - y)$ and $[z]_2 = (2^\ell - [x]_2)/(2^\ell - y)$. Then with probability $1 - 2^{\ell_x + 1 - \ell}$, $\mathsf{Rec}([z]_1, [z]_2) \in \{z - 1,\, z,\, z + 1\}$.

Optimization. We take the above computations as protocol $\Pi_{\mathsf{ProDivPub}}$, it can be used for re-scaling if the divisor is public (e.g. to avoid overflow in Equation 3). The accuracy of $\Pi_{\mathsf{ProDivPub}}$ relies on the fact that the most significant bit (MSB) of share $[x]_1$ is most likely not the same as $[x]_2$, if $|x| < 2^{\ell_x} \ll 2^\ell$. As an example of the failure, we suppose $x = 8$, $y = 2$, $[x]_1 = 4$ and $[x]_2 = 4$, where $\ell = 8$. By applying $\Pi_{\mathsf{ProDivPub}}$ (without shifting p decimal precision), $\mathbf{P_0}$ obtains $[z]_1 = [x]_1/y = 2$, and $\mathbf{P_1}$ obtains $[z]_2 = 2^\ell - ((2^\ell - [x]_2)/y) = 130$. Obviously, the reconstructed value $z = [z]_1 + [z]_2$ is far from the desired $x/y = 4$. To fix this issue, we extend $\Pi_{\mathsf{ProDivPub}}$ with the signed and unsigned integer conversion. Instead of directly dividing their local shares by the public value, both parties firstly convert their local shares from unsigned integers to signed integers, then do the same computation as above. Thus, $\mathbf{P_1}$'s signed local output becomes $[z]_2 = 0 - ((0 - 4)/2) = 2$, where $\mathbf{P_0}$ still has $[z]_1 = 4/2 = 2$. And once both parties convert their local shares back to the unsigned integer, this yields a correct reconstructed result as $[z]_1 + [z]_2 = 4$ (or approximately correct with an error magnitude of 1). Thus, the bad event happens only if the MSBs of shares $[x]_1$ and $[x]_2$ are **both** opposite to the MSB of the secret value x.

6. Optimizations on GPU

We implement NodeGuard by adding all GBDT-related functions to the codebase of Piranha [46]. Please refer to [46] for basic implementation details. In addition, we make many improvements to make it more efficient. We showcase the most significant ones in this section.

Carryout with Optimized Overhead. One of Piranha's main contribution is the iterator-based in-place carryout implementation, which cuts the peak memory load in half. We illustrate their procedure in the left part of Fig. 4. Notably, when computing carryout for a 32-bit value, they decompose it into 32 Bytes. Then perform the operation between even bytes and odd bytes in a tree order. The total communication volume is $16 + 8 + 4 + 2 + 1 = 31$ Bytes.

To be more memory-efficient, we decompose the 32-bit value into 4 Bytes, reducing the memory consumption by

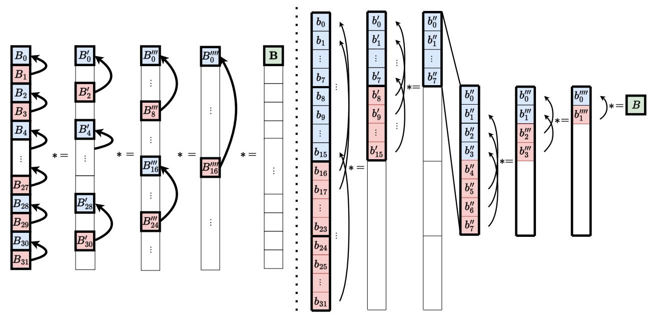


Figure 4. An illustration of Piranha's (left) and our (right) in-place carryout implementation for a 32-bit value. B denotes a Byte, b denotes a bit. A bold border points to one byte memory

c.a. 87.5%. Different from Piranha, we perform the operation between higher bytes and lower bytes. When we reach 1 Byte or 8 bits, we use bit operation within that byte. The process is shown in the right part of Fig. 4. The total communication volume of NodeGuard is $2+1+1+1+1 = 6$Bytes, which is only 20% of Piranha.

Reciprocal with Optimized Rounds. Reciprocal is an important operation in GBDT, which is required to compute gains (Equation 3) and leaf weights (Equation 4). Piranha uses the second order Taylor approximation to calculate reciprocal. The most expensive step in its implementation is to find the least power of 2 that is greater than the input, so as to determine the variable floating-point precision in the Taylor polynomial computation. Piranha implements this in a naïve approach, which loops through all the powers of 2 and compares them with the input one by one. Still take the 32-bit input as an example, assuming a comparison costs 5 rounds as mentioned in the above Section 6. The total communication rounds of this step are $32 \times 5 = 160$.

NodeGuard employs the Goldschmidt-Division algorithm [41] to approximate reciprocal, which leads to a higher precision with the same number of iterations. A similar preparation step to find the least power of 2 is also required here, in order to identify a good initial estimate. Differently, NodeGuard performs all the comparisons in a single round and appends an adjacent xor after that. Note that xor is a cheap local operation without any communication between parties. The total communication rounds are only 5, which is $\frac{1}{32}$ of Piranha. The only sacrifice here is the peak memory load, which is never an issue during our evaluations.

7. Evaluation

7.1. Evaluation Setup

Testbed Environment. We run all the experiments on a server with 2 CPUs, `Intel(R) Xeon(R) Platinum 8360Y CPU @ 2.40GHz`, and `16 × 128GB` of RAM. The server is equipped with 4 GPUs, `Nvidia RTX A5000` including `24GB` of dedicated VRAM. For network settings, we consider LAN with 1`Gbps` bandwidth + 0.2`ms` round-trip latency, and WAN with 100`Mbps` bandwidth + 40`ms` round-trip latency, both simulated with `tc` tool [2].

Baseline. To verify the accuracy, we compare NodeGuard with the plaintext XGBoost library. To verify the efficiency, we compare NodeGuard with following state-of-the-art works (Squirrel [20] has not published the source code yet):

- MP-XGB [34] [3], benchmarked natively on CPU.
- SecretFlow [19], [47] [4], benchmarked natively on GPU.
- Pivot [18] [5], benchmarked in the pre-built docker on CPU.

Fixed-Point Computation. We set the arithmetic sharing length $\ell = 64$ and the fraction precision $p = 20$.

7.2. Accuracy Comparison

To validate the accuracy of NodeGuard, we train the model with several public datasets and compare the training results to the plaintext training results.

Model Hyperparameter and Dataset. We use the same parameters for both plaintext and NodeGuard, such that the tree number $T = 20$, depth $D = 4$ and bucket size $B = 16$. The datasets Concrete Compressive Strength [48] and Energy [49] are selected for regression tasks, Breast Cancer [50] and Credit [51] are used for classification tasks. As for test set, we either use the original one if it is provided, or we split 20% of the dataset. All features are evenly allocated.

Accuracy. We run each experiment 5 times and record the average accuracy in Table 3. We further plot the decreased training loss when a GBDT model is trained with Energy or with Breast Cancer dataset. As shown in both Table 3 and Fig. 5, NodeGuard provides a very close training accuracy compared to the plaintext model.

7.3. Efficiency Comparison

We use random synthetic datasets to evaluate the efficiency, and we set sample size $N = 10,000$, depth $D = 4$, feature size $F = 10$ and bucket size $B = 8$ as default parameters.

2. https://man7.org/linux/man-pages/man8/tc.8.html
3. https://github.com/HikariX/MP-FedXGB at commit *46807ea*
4. https://github.com/secretflow/secretflow at commit *d7bb1d1*
5. https://hub.docker.com/repository/docker/lemonwyc/pivot at commit *942b66c*

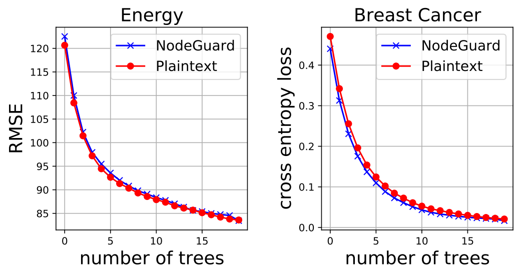


Figure 5. Training loss on Energy and Breast Cancer

TABLE 2. END-TO-END TRAINING TIME *(seconds per Tree)*

N	F	B	D	LAN				WAN			
				NodeGuard	SecretFlow	MP-XGB	Pivot	NodeGuard	SecretFlow	MP-XGB	Pivot
$10,000$	10	8	4	**2.67**	19.64	19.78	239	**35.53**	97.30	444.10	7,285
$50,000$	10	8	4	**4.58**	33.05	174.52	499	**44.61**	156.99	1,309.65	7,781
$10,000$	20	8	4	**3.84**	19.70	42.94	501	**39.25**	101.33	971.43	17,925
$10,000$	10	16	4	**3.92**	19.59	38.81	434	**39.65**	102.26	745.56	15,473
$10,000$	10	8	5	**5.60**	27.85	31.15	398	**45.55**	140.24	1,052.94	13,075

TABLE 3. ACCURACY COMPARISON FOR REGRESSION (REG.) AND CLASSIFICATION (CLS.) TASKS. WE USE CONCRETE(N=1030, F=8), ENERGY(N=19735, F=27), BREAST(N=683, F=30) AND CREDIT(N=30000, F=23) TO BENCHMARK THE ACCURACY.

Type	Dataset	Metric	Plain.	NodeGuard
Reg.	Concrete	RMSE	5.91	5.20
	Energy		85.46	85.14
Cls.	Breast	F1	0.97	0.98
	Credit		0.47	0.46

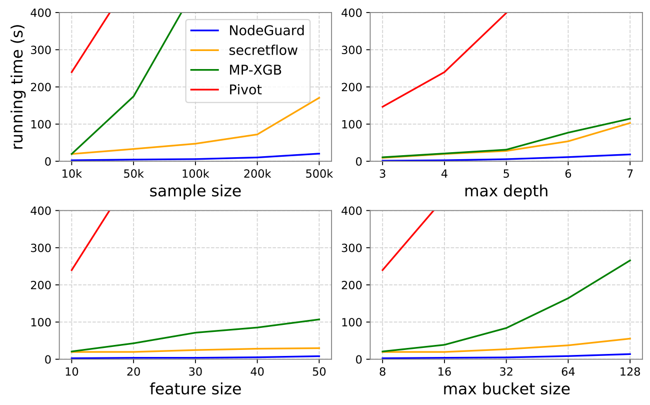


Figure 6. Benchmark of running time in each framework with increased training size in LAN

End-to-End Training Comparison. Given a set of training parameters, we measure the end-to-end training time of existing works in Table 2. Regardless of the training parameters, NodeGuard outperforms all other existing frameworks in both LAN and WAN settings. Such a performance gain is derived from the optimized communication cost and simpler local computation. Compared to the most competitive framework SecretFlow, NodeGuard is 5.0 $\times$ to 7.3 $\times$ faster in LAN, 2.7 $\times$ to 3.5 $\times$ faster in WAN. Against Pivot, NodeGuard is 71 $\times$ to 131 $\times$ faster in LAN, and 174 $\times$ to 457 $\times$ faster in WAN.

Impact of Training Size. We run experiments with different training parameters and report the results in Fig. 6. The plot of Pivot shows that its training time increases exponentially as the training size grows. When the GBDT model is trained with a larger sample size or a larger bucket size, MP-XGB shows a similar performance as Pivot. SecretFlow also tends to run with an exponentially increasing time, when the sample size and the max depth explodes. Apart from the dominance in all experiments, NodeGuard scales the best among all the projects, with a smooth linear growth.

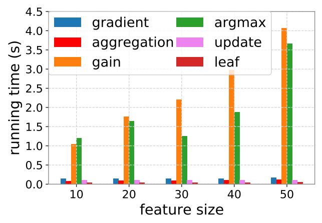


Figure 7. Micro-benchmark of computation modules on increased feature size in LAN

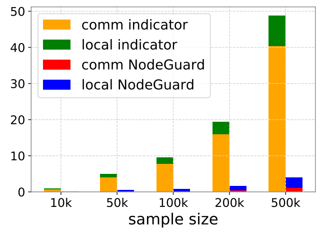


Figure 8. Running time of Π_{IndiBuc} and Π_{KeyBuc} on increased sample size in LAN

Module Scalability Comparison. We decompose the secure GBDT training process into several time-consuming modules, including bucket aggregation, gain, argmax, update sample space and leaf. We take the increased feature size as an example and show our experiment result in Fig. 7. As expected, the running time of Π_{KeyBuc} stays stable and shares only a small portion of the entire training time. The secure gain computation for each split candidate and the argmax computation now become the most costly part.

Impact of other Improvements. To better analyze the efficiency gain of Π_{KeyBuc}, we also implement another sharing-based protocol Π_{IndiBuc} in NodeGuard to eliminate the influence of other improvements we have made. As shown in Fig. 8, NodeGuard improves the communication time by 90% and reduces the local computation time by at least 50%, which matches the theoretical analysis in Table 1.

Inference Efficiency. In NodeGuard, inference is computed without disclosing any intermediate results. This is a user-friendly approach (e.g. in cloud service), if a GBDT model is trained jointly and the input sample must be kept secret. NodeGuard performs inference parallelly on all input shared samples. Given a shared GBDT model of $D = 4$ (16 leaves) per tree and a dataset of $(N = 500,000, F = 10)$, it takes $18.6s$ to handle all samples in LAN.

8. Conclusion

In this work, we have proposed NodeGuard, a highly efficient two-party computation framework for training large-scale gradient boosting decision tree. NodeGuard is secure and accurate for both training and inference without disclosing sensitive intermediate results. Benefiting from a novel secure bucket aggregation protocol and other components, NodeGuard outperforms the existing state-of-the-art secure GBDT frameworks in both LAN and WAN settings.

Appendix A. Preprocessing Ideal Functionality

See Fig. 9.

Appendix B. Proof Sketch

We construct an adversary $\mathcal{S}$ interacting with $\mathcal{F}_{\mathsf{2PC}}^{\mathsf{KeyBuc}}$ such that no environment machine $\mathcal{Z}$ can tell with non-negligible probability whether it is interacting with $\mathcal{A}$ and Π_{KeyBuc} in the real world or with $\mathcal{S}$ in the ideal process for $\mathcal{F}_{\mathsf{2PC}}^{\mathsf{KeyBuc}}$. For the following proofs, we always let parties complete the computation for all $1 \leq b \leq B$ and $1 \leq f \leq F_h$, and do not explicitly say that "for each b of each feature f".

Suppose $\mathbf{P_h}$ is corrupted, we denote the input of $\mathbf{P_h}$ as $\mathcal{I}^{f,b}$ and $[\mathbf{g}_j]_h$ at a node j. Firstly, $\mathcal{S}$ plays the role of $\mathcal{F}_{\mathsf{2PC}}^{\mathsf{Pre}}$, samples $\mathbf{r}^{f,b} = (r_1^{f,b}, ..., r_N^{f,b})$ where $r^{f,b} \xleftarrow{\$} \mathbb{Z}_{2^\ell}$, and $[m_j^{f,b}]_h$ where $[m_j^{f,b}]_h \xleftarrow{\$} \mathbb{Z}_{2^\ell}$. $\mathcal{S}$ sends all $\mathbf{r}^{f,b}$ and $[m_j^{f,b}]_h$ to $\mathbf{P_h}$. At the initial node, $\mathcal{S}$ plays the role of an honest $\mathbf{P_{1-h}}$, receives $\mathbf{v}^{f,b}$ from $\mathcal{A}$. Then $\mathcal{S}$ sends all $\mathcal{I}^{f,b}$ to $\mathcal{F}_{\mathsf{2PC}}^{\mathsf{KeyBuc}}$, which sets the internal state $\mathbf{ready}$ to be true. At any node j, $\mathcal{S}$ samples $\mathbf{q}_j = (q_{j,1}, ..., q_{j,N})$ where $q_{j,i} \xleftarrow{\$} \mathbf{Z}_{2^\ell}$, sends $\mathbf{q}_j$ to $\mathcal{A}$. $\mathcal{S}$ computes $[z_j^{f,b}]_h$ as $\mathcal{A}$ will do during the execution of Π_{KeyBuc}, sends $\mathcal{A}$'s input $[\mathbf{g}_j]_h$ and $\mathcal{A}$'s output $[z_j^{f,b}]_h$ to $\mathcal{F}_{\mathsf{2PC}}^{\mathsf{KeyBuc}}$. Note that $\mathcal{A}$'s input $[\mathbf{g}_j]_h$ is received by $\mathcal{S}$, when $\mathbf{P_h}$ switches the role with $\mathbf{P_{1-h}}$. In the real protocol execution, $\mathbf{q}_j$ is computed by $\mathbf{P_{1-h}}$ as $\mathbf{q}_j = [\mathbf{g}_j]_{1-h} - \mathbf{k}_j$, where in the ideal execution, $\mathbf{q}_j$ is chosen by $\mathcal{S}$ randomly. We notice that since each $\mathbf{k}_j$ is distributed uniformly at random to the environment machine $\mathcal{Z}$ in the real protocol execution, the computed $\mathbf{q}_j$ is distributed uniformly at random as well. Thus, the message sent from $\mathcal{S}$ in the ideal execution is indistinguishable from the one computed by $\mathbf{P_{1-h}}$ in the real protocol execution.

Suppose $\mathbf{P_{1-h}}$ is corrupted, we denote the input of $\mathbf{P_{1-h}}$ as $[\mathbf{g}_j]_{1-h}$ at a node j. Firstly, $\mathcal{S}$ plays the role of $\mathcal{F}_{\mathsf{2PC}}^{\mathsf{Pre}}$, samples $\mathbf{k}_j$, where $\mathbf{k}_j = (k_{j,1}, ...k_{j,N})$ and $k_{j,i} \xleftarrow{\$} \mathbb{Z}_{2^\ell}$, and $[m_j^{f,b}]_{1-h}$ where $[m_j^{f,b}]_{1-h} \xleftarrow{\$} \mathbb{Z}_{2^\ell}$. $\mathcal{S}$ sends both $\mathbf{k}_j$ and $[m_j^{f,b}]_{1-h}$ to $\mathbf{P_{1-h}}$. At the initial node, $\mathcal{S}$ plays the role of an honest $\mathbf{P_h}$, samples $\mathbf{v}^{f,b}$ where $\mathbf{v}^{f,b} = (v_1^{f,b} ..., v_N^{f,b})$ and $v_i^{f,b} \xleftarrow{\$} \mathbb{Z}_{2^\ell}$, sends all $\mathbf{v}^{f,b}$ to $\mathcal{A}$. Then at any node j, $\mathcal{S}$ receives $\mathbf{q}_j$ from $\mathcal{A}$, computes $[z_j^{f,b}]_{1-h}$ just as $\mathcal{A}$ will do. $\mathcal{S}$ sends $\mathcal{A}$'s input $[\mathbf{g}_j]_{1-h}$ and $\mathcal{A}$'s output $[z_j^{f,b}]_{1-h}$ to $\mathcal{F}_{\mathsf{2PC}}^{\mathsf{KeyBuc}}$. In the real protocol execution, $\mathbf{v}^{f,b}$ is computed by $\mathbf{P_h}$ as $\mathbf{v}^{f,b} = \mathbf{s}^{f,b} - \mathbf{r}^{f,b}$, where in the ideal execution, $\mathbf{v}^{f,b}$ is chosen by $\mathcal{S}$ randomly. Since $\mathbf{r}^{f,b}$ is distributed uniformly at random to the environment machine $\mathcal{Z}$, the computed $\mathbf{v}^{f,b}$ is distributed uniformly at random as well. Thus, the message sent from $\mathcal{S}$ in the ideal execution is indistinguishable from the one computed by $\mathbf{P_h}$ in the real protocol execution.

$\mathcal{F}_{\mathsf{2PC}}^{\mathsf{Pre}}$

Private inputs: None.
Public inputs: Public parameters.
Internal State: $\mathbf{ready} \in \{\mathsf{true}, \mathsf{false}\}$.
Initialization: Set $\mathbf{ready} = \mathsf{false}$.
Compute:
Let $\mathcal{P} = \{\mathbf{P_h}, \mathbf{P_{1-h}}\}$. Upon receiving $(\mathsf{BucSGen}, j, \mathbf{P_i}, \mathsf{sid})$ from $\mathbf{P_i} \in \mathcal{P}$:

- If $j = 1$: For each bucket b of each feature f, sample $\mathbf{r}^{f,b}$, where $r_i^{f,b} \xleftarrow{\$} \mathbb{Z}_{2^\ell}$. Record and send all $\mathbf{r}^{f,b}$ to $\mathbf{P_h}$ and set $\mathbf{ready} = \mathsf{true}$.
- If $\mathbf{ready} = \mathsf{true}$:
  - Sample $\mathbf{k}_j$, where $\mathbf{k}_j = (k_{j,1}, ..., k_{j,N})$ and $k_{j,i} \xleftarrow{\$} \mathbb{Z}_{2^\ell}$. Send $\mathbf{k}_j$ to $\mathbf{P_{1-h}}$.
  - For each bucket b of each feature f: Compute $m_j^{f,b} = \mathbf{k}_j \cdot \mathbf{r}^{f,b}$ then sample $[m_j^{f,b}]_h \xleftarrow{\$} \mathbb{Z}_{2^\ell}$ and compute $[m_j^{f,b}]_{1-h} = m_j^{f,b} - [m_j^{f,b}]_h$. Send all $[m_j^{f,b}]_i$ to $\mathbf{P_i} \in \mathcal{P}$.
- If $\mathbf{ready} = \mathsf{false}$, send $(\mathsf{failed}, \mathbf{P_i}, \mathsf{sid})$ to $\mathbf{P_i} \in \mathcal{P}$.

Figure 9. Two-party Preprocessing Functionality $\mathcal{F}_{\mathsf{2PC}}^{\mathsf{Pre}}$

Appendix C. Proof of Theorem 2

Proof: Let $[x]_1 = x + r \bmod 2^\ell$, where r is uniformly random in $\mathbb{Z}_{2^\ell}$, then $[x]_2 = 2^\ell - r$. if $y \in (0, 2^{\ell_x}]$, we decompose r as $r_1 \cdot y + r_2$, where $r_2 < y$. If $y \in [2^\ell - 2^{\ell_x}, 2^\ell)$, we decompose r as $r_1 \cdot (2^\ell - y) + r_2$, where $r_2 < 2^\ell - y$. We prove that if $2^{\ell_x} \leq r < 2^\ell - 2^{\ell_x}$, $\mathsf{Rec}([z]_1, [z]_2) \in \{z-1, z, z+1\}$. Consider the following four cases:

- **Case 1:** If $x \in [0, 2^{\ell_x}]$ and $y \in (0, 2^{\ell_x}]$, then $0 < x + r < 2^\ell$ and $[x]_1 = x + r$ without modulo. Let $x = x_1 \cdot y + x_2$, where $0 \leq x_1 \leq \lfloor x/y \rfloor$ and $0 \leq x_2 < y$. Then we have $x + r = (x_1 + r_1) \cdot y + x_2 + r_2 = (x_1 + r_1 + c) \cdot y + (x_2 + r_2 - c \cdot y)$, where $c = 0$ if $x_2 + r_2 < y$ and $c = 1$ otherwise. After the division, $[z]_1 = x_1 + r_1 + c$ and $[z]_2 = 2^\ell - r_1$. Therefore, $\mathsf{Rec}([z]_1, [z]_2) = 2^\ell + x_1 + c = z + c$.
- **Case 2:** If $x \in [2^\ell - 2^{\ell_x}, 2^\ell)$ and $y \in (0, 2^{\ell_x}]$, then $x + r > 2^\ell$ and $[x]_1 = x + r - 2^\ell$. Let $x = 2^\ell - x_1 \cdot y - x_2$, where $0 \leq x_1 \leq \lfloor (2^\ell - x)/y \rfloor$ and $0 \leq x_2 < y$. We have $x + r - 2^\ell = (r_1 - x_1) \cdot y + r_2 - x_2 = (r_1 - x_1 - c) \cdot y + (r_2 - x_2 + c \cdot y)$, where $c = 0$ if $r_2 > x_2$ and $c = 1$ otherwise. After the division, $[z]_1 = r_1 - x_1 - c$ and $[z]_2 = 2^\ell - r_1$. Therefore, $\mathsf{Rec}([z]_1, [z]_2) = 2^\ell - x_1 - c = z - c$.
- **Case 3:** If $x \in [0, 2^{\ell_x}]$ and $y \in [2^\ell - 2^{\ell_x}, 2^\ell)$, then $0 < x + r < 2^\ell$ and $[x]_1 = x + r$ without modulo. Let $x = x_1 \cdot (2^\ell - y) + x_2$, where $0 \leq x_1 \leq \lfloor x/(2^\ell - y) \rfloor$ and $0 \leq x_2 < 2^\ell - y$. Then we have $x + r = (x_1 + r_1) \cdot (2^\ell - y) + x_2 + r_2 = (x_1 + r_1 + c) \cdot (2^\ell - y) + (x_2 + r_2 - c \cdot (2^\ell - y))$, where $c = 0$ if $x_2 + r_2 < 2^\ell - y$ and $c = 1$ otherwise. After the division, $[z]_1 = 2^\ell - x_1 - r_1 - c$ and

$[z]_2 = r_1$. Therefore, $\mathsf{Rec}([z]_1, [z]_2) = 2^\ell - x_1 - c = z - c$.

- **Case 4:** If $x \in [2^\ell - 2^{\ell_x}, 2^\ell)$ and $y \in [2^\ell - 2^{\ell_x}, 2^\ell)$, then $x + r > 2^\ell$ and $[x]_1 = x + r - 2^\ell$. Let $x = 2^\ell - x_1 \cdot (2^\ell - y) - x_2$, where $0 \le x_1 \le \lfloor (2^\ell - x)/(2^\ell - y) \rfloor$ and $0 \le x_2 < 2^\ell - y$. We have $x + r - 2^\ell = (r_1 - x_1) \cdot (2^\ell - y) + r_2 - x_2 = (r_1 - x_1 - c) \cdot (2^\ell - y) + (r_2 - x_2 + c \cdot (2^\ell - y))$, where $c = 0$ if $r_2 > x_2$ and $c = 1$ otherwise. After the division, $[z]_1 = 2^\ell + x_1 - r_1 + c$ and $[z]_2 = r_1$. Therefore, $\mathsf{Rec}([z]_1, [z]_2) = 2^\ell + x_1 + c = z + c$.

Finally, the probability that our assumption holds, i.e. the probability that r being in range $[2^{\ell_x}, 2^\ell)$, is $1 - 2^{\ell_x + 1 - \ell}$. $\square$

References

[1] J. H. Friedman, “Greedy function approximation: a gradient boosting machine,” *Annals of statistics*, pp. 1189–1232, 2001.

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

[3] G. Rushin, C. Stancil, M. Sun, S. Adams, and P. Beling, “Horse race analysis in credit card fraud—deep learning, logistic regression, and gradient boosted tree,” in *2017 SIEDS*. IEEE, 2017, pp. 117–121.

[4] Z. Shahbazi and Y.-C. Byun, “Product recommendation based on content-based filtering using xgboost classifier,” *Int. J. Adv. Sci. Technol*, vol. 29, 2019.

[5] X. Ling, W. Deng, C. Gu, H. Zhou, C. Li, and F. Sun, “Model ensemble for click prediction in bing search ads,” in *WWW ’17 Companion*, 2017.

[6] Y. Meng, N. Yang, Z. Qian, and G. Zhang, “What makes an online review more helpful: an interpretation framework using xgboost and shap values,” *Journal of Theoretical and Applied Electronic Commerce Research*, 2020.

[7] Z. Tian, J. Xiao, H. Feng, and Y. Wei, “Credit risk assessment based on gradient boosting decision tree,” *Procedia Computer Science*, vol. 174, pp. 150–160, 2020.

[8] European Parliament and Council of the European Union, “General data protection regulation.” [Online]. Available: https://data.europa.eu/eli/reg/2016/679/oj

[9] J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, “Federated learning: Strategies for improving communication efficiency,” *arXiv preprint arXiv:1610.05492*, 2016.

[10] Y. Huang, S. Gupta, Z. Song, K. Li, and S. Arora, “Evaluating gradient inversion attacks and defenses in federated learning,” *Advances in Neural Information Processing Systems*, vol. 34, pp. 7232–7241, 2021.

[11] X. Jin, P.-Y. Chen, C.-Y. Hsu, C.-M. Yu, and T. Chen, “Cafe: Catastrophic data leakage in vertical federated learning,” *Advances in Neural Information Processing Systems*, vol. 34, pp. 994–1006, 2021.

[12] J. Geiping, H. Bauermeister, H. Dröge, and M. Moeller, “Inverting gradients-how easy is it to break privacy in federated learning?” *Advances in Neural Information Processing Systems*, vol. 33, pp. 16 937–16 947, 2020.

[13] C. Fu, X. Zhang, S. Ji, J. Chen, J. Wu, S. Guo, J. Zhou, A. X. Liu, and T. Wang, “Label inference attacks against vertical federated learning,” in *31st USENIX Security Symposium*, 2022, pp. 1397–1414.

[14] A. C.-C. Yao, “How to generate and exchange secrets,” in *27th annual symposium on foundations of computer science*. IEEE, 1986, pp. 162–167.

[15] B. Pinkas, T. Schneider, N. Smart, and S. Williams, “Secure two-party computation is practical,” in *ASIACRYPT 2009*. Springer, 2009, pp. 250–267.

[16] D. Beaver, “Efficient multiparty protocols using circuit randomization,” in *Advances in Cryptology—CRYPTO’91: Proceedings 11*. Springer, 1992, pp. 420–432.

[17] K. Cheng, T. Fan, Y. Jin, Y. Liu, T. Chen, D. Papadopoulos, and Q. Yang, “Secureboost: A lossless federated learning framework,” *IEEE Intelligent Systems*, vol. 36, no. 6, pp. 87–98, 2021.

[18] Y. Wu, S. Cai, X. Xiao, G. Chen, and B. C. Ooi, “Privacy preserving vertical federated learning for tree-based models,” *VLDB Endowment*, vol. 13, no. 12, 2020.

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

[20] W. Lu, Z. Huang, Q. Zhang, Y. Wang, and C. Hong, “Squirrel: A scalable secure Two-Party computation framework for training gradient boosting decision tree,” in *32nd USENIX Security Symposium*, 2023, pp. 6435–6451.

[21] R. Canetti, “Universally composable security: A new paradigm for cryptographic protocols,” in *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*. IEEE, 2001, pp. 136–145.

[22] P. Mohassel and Y. Zhang, “Secureml: A system for scalable privacy-preserving machine learning,” in *IEEE symposium on security and privacy (SP)*, 2017.

[23] P. Mohassel and P. Rindal, “Aby3: A mixed protocol framework for machine learning,” in *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, 2018, pp. 35–52.

[24] Q. Yang, Y. Liu, T. Chen, and Y. Tong, “Federated machine learning: Concept and applications,” *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 10, no. 2, pp. 1–19, 2019.

[25] 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*. PMLR, 2017, pp. 1273–1282.

[26] Y. Lindell and B. Pinkas, “Privacy preserving data mining,” in *Annual International Cryptology Conference*. Springer, 2000, pp. 36–54.

[27] S. De Hoogh, B. Schoenmakers, P. Chen, and H. op den Akker, “Practical secure decision tree learning in a teletreatment application,” in *Financial Cryptography and Data Security*. Springer, 2014, pp. 179–194.

[28] M. Abspoel, D. Escudero, and N. Volgushev, “Secure training of decision trees with continuous attributes,” *Proceedings on Privacy Enhancing Technologies*, vol. 1, pp. 167–187, 2021.

[29] A. Law, C. Leung, R. Poddar, R. A. Popa, C. Shi, O. Sima, C. Yu, X. Zhang, and W. Zheng, “Secure collaborative training and inference for xgboost,” in *PPMLP’20*, 2020, pp. 21–26.

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

[31] F. Mo, H. Haddadi, K. Katevas, E. Marin, D. Perino, and N. Kourtellis, “Ppfl: privacy-preserving federated learning with trusted execution environments,” in *Proceedings of the 19th annual international conference on mobile systems, applications, and services*, 2021, pp. 94–108.

[32] F. Fu, Y. Shao, L. Yu, J. Jiang, H. Xue, Y. Tao, and B. Cui, “Vf2boost: Very fast vertical federated gradient boosting for cross-enterprise learning,” in *International Conference on Management of Data*, 2021, pp. 563–576.

[33] W. Chen, G. Ma, T. Fan, Y. Kang, Q. Xu, and Q. Yang, “Secureboost+: A high performance gradient boosting tree framework for large scale vertical federated learning,” *arXiv preprint arXiv:2110.10927*, 2021.

[34] L. Xie, J. Liu, S. Lu, T.-H. Chang, and Q. Shi, "An efficient learning framework for federated xgboost using secret sharing and distributed optimization," *ACM Transactions on Intelligent Systems and Technology*, vol. 13, no. 5, 2022.

[35] C. Dwork, A. Roth *et al.*, "The algorithmic foundations of differential privacy," *Foundations and Trends® in Theoretical Computer Science*, vol. 9, no. 3–4, pp. 211–407, 2014.

[36] C. Dwork, G. N. Rothblum, and S. Vadhan, "Boosting and differential privacy," in *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*. IEEE, 2010, pp. 51–60.

[37] S. Fletcher and M. Z. Islam, "Differentially private random decision forests using smooth sensitivity," *Expert Systems with Applications*, vol. 100, no. 78, pp. 16–31, 2017.

[38] S. Chatel, A. Pyrgelis, J. R. Troncoso-Pastoriza, and J.-P. Hubaux, "Sok: Privacy-preserving collaborative tree-based model learning," *Proceedings on Privacy Enhancing Technologies*, vol. 3, pp. 182–203, 2021.

[39] M. Keller, V. Pastro, and D. Rotaru, "Overdrive: making spdz great again," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2018, pp. 158–189.

[40] D. Demmler, T. Schneider, and M. Zohner, "Aby-a framework for efficient mixed-protocol secure two-party computation." in *NDSS*, 2015.

[41] R. E. Goldschmidt, "Applications of division by convergence," Ph.D. dissertation, Massachusetts Institute of Technology, 1964.

[42] M. Zhang, S. Vassiliadis, and J. G. Delgado-Frias, "Sigmoid generators for neural computing using piecewise approximations," *IEEE transactions on Computers*, vol. 45, no. 9, pp. 1045–1049, 1996.

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

[44] L. Breiman, J. H. Freidman, R. A. Olshen, and C. J. Stone, "Cart: Classification and regression trees," 1984.

[45] O.-E. Ørebæk and M. Geitle, "Exploring the hyperparameters of xgboost through 3d visualizations." in *AAAI-MAKE 2021*, 2021.

[46] J.-L. Watson, S. Wagh, and R. A. Popa, "Piranha: A {GPU} platform for secure computation," in *31st USENIX Security Symposium*, 2022, pp. 827–844.

[47] J. Ma, Y. Zheng, J. Feng, D. Zhao, H. Wu, W. Fang, J. Tan, C. Yu, B. Zhang, and L. Wang, "Secretflow-spu: A performant and user-friendly framework for privacy-preserving machine learning," in *USENIX ATC*, 2023.

[48] I.-C. Yeh, "Concrete compressive strength," UCI Machine Learning Repository, 2007, DOI: https://doi.org/10.24432/C5PK67.

[49] L. Candanedo, "Appliances energy prediction," UCI Machine Learning Repository, 2017, DOI: https://doi.org/10.24432/C5VC8G.

[50] W. Wolberg, O. Mangasarian, N. Street, and W. Street, "Breast cancer wisconsin (diagnostic)," UCI Machine Learning Repository, 1995, DOI: https://doi.org/10.24432/C5DW2B.

[51] I.-C. Yeh, "Default of credit card clients," UCI Machine Learning Repository, 2016, DOI: https://doi.org/10.24432/C55S3H.