Accio: Secure and Lean Gradient Boosting Decision Tree Training via Communication Optimization

Peiying Xu[1,2] and Li-Ping Wang[1,2*]
[1]*State Key Laboratory of Cyberspace Security Defense, Institute of Information Engineering, CAS*, Beijing, China
[2]*School of Cyber Security, University of Chinese Academy of Sciences*, Beijing, China
{xupeiying, wangliping}@iie.ac.cn

***Abstract*—Privacy-preserving machine learning (PPML) offers a promising framework for collaborative model training without user privacy leakage. And secure multi-party computation (MPC) stands out as one of the key technologies, as it allows distrustful parties to compute a function without revealing their private inputs. However, existing GBDT training frameworks suffer from communication inefficiency, primarily due to two issues: (1) high round complexity from linear operations based on general MPC techniques, and (2) heavy communication overhead from argmax computation that involves numerous comparisons. In this paper, we present *Accio*, a secure two-party GBDT training framework on vertically partitioned dataset. The main contributions of *Accio* are two-fold: the first part includes a leaner argmax protocol that reduces the online communication by nearly half, while the second part includes a secure protocol to efficiently realize node split. The experimental results demonstrate that *Accio* outperforms the state-of-the-art framework (i.e., SiGBDT) in both LAN and WAN settings.**

***Index Terms*—secure two-party computation, gradient boosting decision tree, privacy-preserving machine learning**

I. Introduction

Gradient Boosting Decision Tree (GBDT) and its variants (e.g. XGBoost [1], LightGBM [2], etc.) are extensively used in real-world applications due to their high performance and good interpretability for classification and regression tasks, such as fraud detection, online advertisement, and risk management. In reality, the data used for model training often belongs to various entities who seek to collaborate on training a model but are hesitant to divulge their confidential raw data. On the other hand, the evolving regulatory environment has placed greater restrictions (e.g. HIPPA and GDPR) on access to sensitive data.

The potential privacy issues surrounding cross-enterprise machine learning have motivated privacy-preserving machine learning (PPML) [3], which enables multiple parties to collaboratively compute machine learning algorithms without revealing any sensitive information. In particular, secure multi-party computation (MPC) is an important building block to safeguard both data and model privacy. To design efficient GBDT training systems, the mixed-mode MPC approach is commonly used. For example, frameworks Pivot [4] and HEP-XGB [5] use the Paillier cryptosystem for secure gradient aggregation, while they use secret sharing (SS) and a permutation-based approach, respectively, to conceal sensitive intermediate information. Squirrel [6] further applies lattice-based HE to an optimized method for gradient aggregation, and hides the sample distribution using correlated oblivious transfer (COT). Although these HE-based protocols have minimal round complexity, their online communication overhead is larger than the input size by a multiplicative factor related to the security parameter (e.g., 128 in practice). SiGBDT [7] targets optimizations for both costly linear computations (e.g., bucket aggregation) and complicated non-linear operations (e.g., argmax), and devises protocols with the almost "optimal online communication complexity" via function secret sharing (FSS) [8]. However, numerous comparisons are required to realize secure argmax computation, which introduces significant online communication overhead.

In this paper, we present *Accio*, a communication-optimized framework for GBDT training on vertically partitioned dataset. *Accio* enables two parties to efficiently train a GBDT model while concealing the sensitive intermediate information from semi-honest parties during the training process. In *Accio*, we propose two main optimizations to reduce online communication overhead as follows:

- For non-linear operations, we propose a leaner argmax protocol that reduces online communication by nearly half compared to the prior FSS-based counterpart. This improvement is driven by a reduction in the number of comparisons, which is achieved through a novel oblivious tree permutation technique.
- For linear operations, we develop a node split protocol to securely realize the sample space update. At its core, our protocol adopts the FSS-based select functionality and a local-update strategy, and thus avoids element-wise multiplications over the entire dataset, making it more communication-efficient than prior frameworks.

With a careful combination of the above optimizations, we implement a prototype of *Accio* and perform extensive evaluations. We demonstrate its efficiency and scalability through experiments conducted under a variety of training parameters and network settings. The comparison results show that *Accio* can perform secure GBDT training on large-scale datasets with smaller computational and communication overheads than the state-of-the-arts two-party GBDT training framework.

II. Preliminaries

A. Notation

We list the notations and their definitions in TABLE I.

TABLE I: Notations of *Accio*.

Notation	Definition
$[\ell]$	integer set $\{1, \ldots, \ell\}$ for $\ell \in \mathbb{N}$
$\odot$ / $\oplus$	element-wise multiplication / XOR
$1\{\mathbb{P}\}$	1 if the predicate $\mathbb{P}$ is true, 0 otherwise
$\mathbf{u}$, $\mathbf{u}[i]$ (or $\mathbf{u}_i$)	a row vector, the i-th element in a vector
$\langle \mathbf{u}, \mathbf{v} \rangle$	inner product of $\mathbf{u}$ and $\mathbf{v}$
$a\|_d$	d-th digit (from LSB as 0) of a positive integer a
$\pi : [n] \to [n]$	permutation (bijection) of the indices $1, \ldots, n$
$(\mathbf{v} \cdot \pi)_i := \mathbf{v}_{\pi(i)}$	permutation application action to $\mathbf{v}$ at index i
$[\![x]\!]_i$ (or $[\![x]\!]$)	arithmetic share of $x \in \mathbb{Z}_{2^\ell}$ owned by party $\mathcal{P}_i$
$[\![y]\!]^{\mathsf{B}}_i$ (or $[\![y]\!]^{\mathsf{B}}$)	boolean share of $y \in \mathbb{Z}_2$ owned by party $\mathcal{P}_i$
N, F, B	number of samples, features, and buckets

B. FSS Ideal Functionalities

We model the following FSS gates as ideal functionalities, invoked as sub-protocols in our protocols.

Preprocessing Functionality. The preprocessing functionality $\mathcal{F}^{\mathsf{Pre}}_{\mathsf{FSS}}$ outputs circuit-dependent correlated randomness for both parties, such as input and output offsets $([\![r^{\mathsf{in}}]\!], [\![r^{\mathsf{out}}]\!])$ (or $([\![r^{\mathsf{in}}]\!]^{\mathsf{B}}, [\![r^{\mathsf{out}}]\!]^{\mathsf{B}})$ for boolean values).

FSS Select Functionality. The FSS-based select functionality $\mathcal{F}^{\mathsf{Select}}_{\mathsf{FSS}}$ takes $\hat{x}$ ($\hat{x} = x + r^{\mathsf{in}}_x$) and $\hat{s}$ ($\hat{s} = s \oplus r^{\mathsf{in}}_s$) as inputs and outputs $[\![\hat{z}]\!]$ ($\hat{z} = z + r^{\mathsf{out}}$), where $z = x$ if $s = 1$, and $z = 0$ otherwise. We utilize the implementation from [9].

FSS dReLU Functionality. The FSS-based dReLU functionality $\mathcal{F}^{\mathsf{dReLU}}_{\mathsf{FSS}}$ takes $\hat{x}$ ($\hat{x} = x + r^{\mathsf{in}}_x$) as input and outputs $[\![\hat{z}]\!]^{\mathsf{B}}$ ($\hat{z} = z \oplus r^{\mathsf{out}}$), where $z = 1$ if $x \geq 0$ and $z = 0$ otherwise. This functionality can be easily achieved via distributed comparison function (DCF) [8].

C. Tree Permutation

We briefly review the *tree permutation* introduced in [10].

Let $T_h := (d, j) \mid 0 \leq d \leq h, 1 \leq j \leq 2^d$ denote the nodes of a full binary tree of height h, where (d, j) is the j-th node at depth d. The function $\mathsf{anc} : \{0, \ldots, h\} \times [2^h] \to T_h$, defined as $\mathsf{anc}(d, j) = (d, \lceil \frac{j}{2^{h-d}} \rceil)$, gives the ancestor at depth d of the j-th leaf. Let $C_h := \mathbb{Z}_2^{T_h}$ define the set of assignments of a boolean label to each node in the binary tree with height h. For $\mathbf{c} \in C_h$, let $\mathbf{c}_{d,j}$ be the label of node (d, j).And $\mathbf{c} \restriction_d \in C_d$ is taken from $\mathbf{c}$ with depth d. The *trace path* $\mathsf{path}_{\mathbf{c}}(i)$ records the labels in $\mathbf{c}$ for all non-leaf nodes on the path from the root to the i-th leaf, encoded as a h-bit vector. $\mathsf{Decom}(\cdot)$ is the bit decomposition function.

Tree permutation is defined by node swapping on a complete binary tree. Let $\mathbf{c} \in C_{h-1}$ and $\mathbf{v}$ be a vector of 2^h. The tree permutation of $(\mathbf{c}, \mathbf{v})$ defined by $\mathbf{r} \in C_{h-1}$ is given as follows:

$$(\mathbf{c}, \mathbf{v}) \cdot \mathbf{r} := (\mathbf{c} \circ \mathbf{r}, \mathbf{v} \cdot \pi_{\mathbf{r}}),$$

which consists of two core components:

- Permutation derived from a tree label $\mathbf{r} \in C_{h-1}$: defined as $\pi_{\mathbf{r}} : \mathbb{Z}_{2^h} \to \mathbb{Z}_{2^h}$ such that $\pi_{\mathbf{r}}(j)|_{h-d-1} := j|_{h-d-1} \oplus \mathbf{r}_{\mathsf{anc}(d,j)}$ for $0 \leq d < h$. The tree extension of $\pi_{\mathbf{r}}$ on T_{h-1}, which consists of permutations on vectors at each depth, is defined as $\pi_{\mathbf{r}} : T_{h-1} \to T_{h-1}$ such that $\pi_{\mathbf{r}}(d, j) := (d, \pi_{\mathbf{r}\restriction_{d-1}}(j))$.
- Binary operation on C_{h-1}: defined as $(\circ) : C_{h-1} \times C_{h-1} \to C_{h-1}$ such that $\mathbf{c} \circ \mathbf{r} = (\mathbf{c} \cdot \pi_{\mathbf{r}}) \oplus \mathbf{r}$, where $\mathbf{c} \cdot \pi_{\mathbf{r}}$ is given as $(\mathbf{c} \cdot \pi_{\mathbf{r}})_{d,j} := \mathbf{c}_{\pi_{\mathbf{r}}(d,j)}$.

We highlight the following properties that are crucial to our construction:

(1) $(C_{h-1}, \circ)$ forms a finite group, where the identity is the all-zero label $\mathbf{0}$, and the inverse of $\mathbf{r} \in C_{h-1}$ is given as $\mathbf{r} \cdot \pi_{\mathbf{r}}^{-1}$.
(2) For $\mathbf{c}, \mathbf{r} \in C_{h-1}$, it holds that $\mathbf{c} \circ \mathbf{r} = (\mathbf{c} \cdot \pi_{\mathbf{r}}) \oplus \mathbf{r} = (\mathbf{c} \oplus \mathbf{r} \cdot \pi_{\mathbf{r}}^{-1}) \cdot \pi_{\mathbf{r}}$.
(3) For $\mathbf{c}, \mathbf{r} \in C_{h-1}$, $\mathsf{path}_{\mathbf{c} \circ \mathbf{r}}(\pi_{\mathbf{r}}(j)) = \mathsf{Decom}(\pi_{\mathbf{r}}(j) - 1)$ if and only if $\mathsf{path}_{\mathbf{c}}(j) = \mathsf{Decom}(j - 1)$ holds for some $j \in [2^h]$.

D. Gradient Boosting Decision Tree

Given a dataset $\mathbf{X} \in \mathbb{R}^{N \times F}$, a GBDT model contains T decision trees of depth D. We assume that each tree is a full binary tree with $2^D - 1$ non-leaf nodes ($n_{\mathsf{non}} = 2^D - 1$) and 2^D leaf nodes ($n_{\mathsf{leaf}} = 2^D$). Each tree classifies a sample $\mathbf{x}_i \in \mathbb{R}^F$ to one leaf node and results at a weight. The final prediction result of $\mathbf{x}_i$ is the sum of T weights $\hat{y}_i = \sum_{t=1}^{T} f_t(\mathbf{x}_i)$.

We consider the approximate split finding algorithm used in most GBDT frameworks [1], [2], where split candidates are determined by percentiles in the distribution of each feature. Let $\mathcal{I}^{f,b}$ denote the indices of samples assigned to the b-th bucket of feature f. The total number of split candidates across B buckets is $F \times (B - 1)$.

A GBDT tree is constructed recursively from top to bottom. At the initial node, the gradient sum for each bucket can be computed as: $G^{f,b} = \sum_{i \in \mathcal{I}^{f,b}} g_i$. Then for each split candidate, buckets are partitioned into left and right child nodes. The gradient sums for the i-th candidate are: $G_L = \sum_{b \leq i} G^{f,b}$ and $G_R = \sum_{b < i \leq B} G^{f,b}$, with analogous definitions for the hessian sums H_L and H_R. Afterward, the Gain of each split candidate is computed as:

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

where γ and δ are public regularization parameters. The split with the maximum Gain is selected as the best one. Once the best split information is determined, update the indices set $\mathcal{I}^{f,b}$ for child nodes, and repeat the split procedure until depth D is reached. Let $\mathcal{I}^{leaf}$ denote the indices of samples falling into that leaf node, the leaf weight can be computed as:

$$\omega = \frac{\sum_{i \in \mathcal{I}^{leaf}} g_i}{\sum_{i \in \mathcal{I}^{leaf}} h_i + \gamma}.$$

III. Problem Statement

In this section, we describe the architecture and security model of *Accio*.

A. Architecture

The architecture of *Accio* involves two parties, $\mathcal{P}_0$ and $\mathcal{P}_1$, in a *vertical* setting. Suppose the (aligned) dataset $\mathbf{X} \in \mathbb{R}^{N \times F}$ is vertically partitioned as $\mathbf{X} = \mathbf{X}_0 \| \mathbf{X}_1$, where $\mathcal{P}_0$ holds $\mathbf{X}_0 \in \mathbb{R}^{N \times F_0}$ and the label $\mathbf{y} \in \mathbb{R}^N$, and $\mathcal{P}_1$ holds $\mathbf{X}_1 \in \mathbb{R}^{N \times F_1}$.

Formally, let Input_0 and Input_1 be the private input of $\mathcal{P}_0$ and $\mathcal{P}_1$ to the two-party protocol Π that privately implements the GBDT functionality $\mathcal{F}_{\mathsf{GBDT}}$ described in Algorithm 1. And the output Output_i (to $\mathcal{P}_i$ for $i \in \{0,1\}$) consists of shares of leaf weights for the t-th tree $\mathbf{w}^{(t)} = (w_1^{(t)}, \ldots, w_{2^D}^{(t)})$, and the best split candidate (f_*, v_*) for each non-leaf node.

$\mathcal{F}_{\mathsf{GBDT}}(\mathsf{Input}_0 = \{\mathbf{X}_0, \mathbf{y}\}, \mathsf{Input}_1 = \{\mathbf{X}_1\}, \mathsf{pp})$:

- $\mathsf{Output}_0 = \{\mathcal{C}_0^{(t)}, [\mathbf{w}^{(t)}]_0\}_{1 \le t \le T}$, where for $1 \le k \le n_{\mathsf{non}}$, $\mathcal{C}_0^{(t)}[k] = (f_*^k, b_*^k)$ if $f_*^k \le F_0$ and $\mathcal{C}_0^{(t)}[k] = \perp$ otherwise;
- $\mathsf{Output}_1 = \{\mathcal{C}_1^{(t)}, [\mathbf{w}^{(t)}]_1\}_{1 \le t \le T}$, where for $1 \le k \le n_{\mathsf{non}}$, $\mathcal{C}_1^{(t)}[k] = (f_*^k, b_*^k)$ if $f_*^k > F_0$ and $\mathcal{C}_1^{(t)}[k] = \perp$ otherwise.

Algorithm 1: The GBDT functionality $\mathcal{F}_{\mathsf{GBDT}}$

B. Security Model

Accio is designed for the two-party semi-honest setting with the preprocessing model. We let a trusted dealer generate all correlated randomness in the preprocessing stage. Then in the online stage, both parties follow the execution of the protocol and only one of them is corrupted by an adversary.

Let $\mathcal{F} = (\mathcal{F}_0, \mathcal{F}_1) : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^*$ be a two-party ideal functionality, and Π a two-party protocol for computing $\mathcal{F}$. $\mathsf{REAL}_{\Pi,\mathcal{A}}$ denotes the output distribution of $\mathcal{A}$ executing Π in the real world, and $\mathsf{IDEAL}_{\mathcal{F},\mathcal{S}}$ denote the output distribution generated by a simulator $\mathcal{S}$ connecting to $\mathcal{F}$ in the ideal world.

Definition 1 (Privacy). *We say that a protocol Π privately realizes a functionality $\mathcal{F}$ if for any non-uniform probabilistic PPT adversary $\mathcal{A}$, there exists a non-uniform PPT simulator $\mathcal{S}$, such that*

$$\mathsf{IDEAL}_{\mathcal{F},\mathcal{S}} \overset{c}{\approx} \mathsf{REAL}_{\Pi,\mathcal{A}},$$

where $\overset{c}{\approx}$ denotes computational indistinguishability.

Accio is constructed following the composition theorem [11]. Provably secure sub-protocols invoked in our protocols are abstracted as ideal functionalities in a hybrid model. In this paper, a protocol invoking a functionality $\mathcal{F}$ is said to be in "$\mathcal{F}$-hybrid mode".

In the context of training, only an indicator bit $c \in \{0,1\}$ pointing to the party $\mathcal{P}_c$ who holds the best split candidate is allowed to be revealed. $\mathcal{P}_c$ records the split information, while P_{1-c} is invisible to the index of the best split candidate. Once a non-leaf node is split, parties have to securely update the sample space for each child node to conceal the relative order of samples. Correspondingly, the gradient and the hessian for each child node are required to be securely updated. When a leaf node is reached, the leaf weight is securely computed between parties. During the training process, the leaf weights and the intermediate values are not revealed.

IV. The Proposed *Accio*

We first introduce two key optimizations and then present our *Accio* framework through a combination of them.

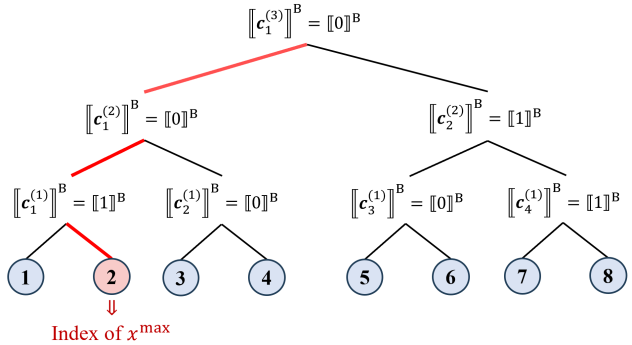


Fig. 1: A Toy Example of the Special Trace Path.

A. Communication Optimization for Secure Argmax

To identify the best split feature and the threshold for each non-leaf node, the secure argmax computation is required to compare and find the maximum Gain over all split candidates.

The argmax functionality takes a (secret-shared) vector $[\![\mathbf{x}]\!]$ (where $\mathbf{x} = (x_1, \ldots, x_N)$) of size N as input, and outputs $[\![\mathbf{z}]\!]^B$ with $\mathbf{z}$, an indicator vector pointing out the position of the max value $x^{\max}$, i.e., $\mathbf{z}_i = 1\{\mathbf{x}_i = \max(\mathbf{x})\}$. For simplicity, we assume that N is a power of two and each x_i is different.

Our protocol is inspired by the silent FSS argmax protocol Π_{SiArg} in [7]. In Π_{SiArg}, parties perform $N-1$ comparisons among N elements $[\![\mathbf{x}]\!]$ to determine $[\![x^{\max}]\!]$, then perform N parallel comparisons between each element of $[\![\mathbf{x}]\!]$ and $[\![x^{\max}]\!]$ to acquire $[\![\mathbf{z}]\!]^B$.

Our intuition is that the latter N comparisons in Π_{SiArg} can be eliminated. That is, outputs of $N-1$ comparisons, denoted by $[\![\mathbf{c}^{(1)}]\!]^B, \ldots, [\![\mathbf{c}^{(\log N)}]\!]^B$, can be organized into a binary tree with N leaves, where the leaves (at height 0) correspond to elements in $\mathbf{x}$ and the nodes at height j ($j \in [\log N]$) are labeled by $\mathbf{c}^{(j)}$. The boolean label of each non-leaf node indicates the comparison result between its child nodes, i.e., 0 indicates the left child is greater and 1 otherwise. This implies that $\mathsf{path}_{\mathbf{c}}(i) = \mathsf{Decom}(i-1)$ holds if and only if $\mathbf{x}_i = x^{\max}$. The argmax result $\mathbf{z} \in \{0,1\}^N$ therefore indicates the leaf label, subject to $\mathbf{z}_i = 1\{\mathsf{path}_{\mathbf{c}}(i) = \mathsf{Decom}(i-1)\}$ for $i \in [N]$. We demonstrate a toy example of such a special trace path (marked in red) with $n = 8$ and $x^{\max} = x_2$ in Fig. 1.

To do this, we use an oblivious tree permutation technique, which ensures that the equality relationship of the special trace path remains unchanged after permutation. Loosely speaking, parties perform an oblivious tree permutation on the (masked) tree label $\hat{\mathbf{c}}$, and reveal the permuted one $\mathbf{t}$ to $\mathcal{P}_1$, who then locally computes the permuted argmax result $\hat{\mathbf{z}}$ by checking the equality of $\mathsf{path}_{\mathbf{t}}(i)$ and $\mathsf{Decom}(i-1)$ for $i \in [N]$. Finally, parties receive the argmax result $[\![\mathbf{z}]\!]^B$ through the inverse tree permutation.

We work with FSS select ($\mathcal{F}_{\mathsf{FSS}}^{\mathsf{Select}}$) and comparison ($\mathcal{F}_{\mathsf{FSS}}^{\mathsf{dReLU}}$) functionalities to build the secure argmax protocol in Algorithm 2. We divide $\Pi_{\mathsf{2PC}}^{\mathsf{SeArg}}$ into two parts:

- In the *MaxReduce* part, parties perform $N-1$ secure comparisons $\mathcal{F}_{\mathsf{FSS}}^{\mathsf{dReLU}}$ over $\log N$ rounds. During the execution

Private inputs: Parties hold $[\![\mathbf{x}]\!]$.
Public inputs: Public parameters. $|\mathbf{x}| = N$, and N is a power of 2.
Outputs: Parties output $[\![\mathbf{z}]\!]^{\mathsf{B}}$, where $\mathbf{z} \in \{0,1\}^N$ and $\mathbf{z}[i] = 1\{x_i = \mathsf{max}(\mathbf{x})\}$.
Initialization: Parties set $\mathsf{csize} = N$ and $\mathsf{cnt} = 1$.
Preprocessing:
1: Parties receive $[\![\mathbf{u}]\!]$ from $\mathcal{F}_{\mathsf{FSS}}^{\mathsf{Pre}}$ as (shared) masks for the arithmetic inputs.
2: Parties receive $[\![\mathbf{r}]\!]^{\mathsf{B}}$ from $\mathcal{F}_{\mathsf{FSS}}^{\mathsf{Pre}}$ as (shared) masks for the boolean inputs.
3: $\mathcal{P}_0$ receives $\hat{\mathbf{r}}$, $[\![(\mathbf{m} \cdot \pi_{\hat{r}^{-1}})]\!]_0^{\mathsf{B}}$ and $\mathbf{\Delta}$, and $\mathcal{P}_1$ receives $\mathbf{m}$, $[\![(\mathbf{m} \cdot \pi_{\hat{r}^{-1}})]\!]_1^{\mathsf{B}}$ and (α, β) from $\mathcal{F}_{\mathsf{FSS}}^{\mathsf{Pre}}$, where $\mathbf{m} \in C_{\log N - 1}$, $\hat{\mathbf{r}} = \mathbf{r} \oplus \mathbf{m}$, $\hat{\mathbf{r}}^{-1} := \hat{\mathbf{r}} \cdot \pi_{\hat{\mathbf{r}}}^{-1}$, and $\mathbf{\Delta}, \alpha, \beta \in \{0,1\}^N$ such that $\mathbf{\Delta} := \pi_{\hat{\mathbf{r}}^{-1}}^{-1}(\alpha) \oplus \beta$.
Protocol:
1: Parties set $[\![\hat{\mathbf{x}}]\!] = [\![\mathbf{x}]\!] + [\![\mathbf{u}]\!]$, and $[\![\hat{\mathbf{x}}^{\mathsf{tmp}}]\!] = [\![\hat{\mathbf{x}}]\!]$.
2: **[MaxReduce].** For $\mathsf{csize} \neq 1$, parties do:
(1) Set $[\![\hat{\mathbf{x}}^{\mathsf{right}}]\!] = \{[\![\hat{x}_2^{\mathsf{tmp}}]\!], [\![\hat{x}_4^{\mathsf{tmp}}]\!], \cdots, [\![\hat{x}_{\mathsf{csize}}^{\mathsf{tmp}}]\!]\}$, and $[\![\hat{\mathbf{x}}^{\mathsf{left}}]\!] = \{[\![\hat{x}_1^{\mathsf{tmp}}]\!], [\![\hat{x}_3^{\mathsf{tmp}}]\!], \cdots, [\![\hat{x}_{\mathsf{csize}-1}^{\mathsf{tmp}}]\!]\}$. Compute $[\![\hat{\mathbf{x}}^{\mathsf{diff}}]\!] = [\![\hat{\mathbf{x}}^{\mathsf{right}}]\!] - [\![\hat{\mathbf{x}}^{\mathsf{left}}]\!]$, then reconstruct $\hat{\mathbf{x}}^{\mathsf{diff}}$.
(2) Call $\mathcal{F}_{\mathsf{FSS}}^{\mathsf{dReLU}}$ with $\hat{\mathbf{x}}^{\mathsf{diff}}$ as input, receive $[\![\hat{\mathbf{c}}^{(\mathsf{cnt})}]\!]^{\mathsf{B}}$ as output. Reconstruct $\hat{\mathbf{c}}^{(\mathsf{cnt})} = [\![\hat{\mathbf{c}}^{(\mathsf{cnt})}]\!]_0^{\mathsf{B}} \oplus [\![\hat{\mathbf{c}}^{(\mathsf{cnt})}]\!]_1^{\mathsf{B}}$, where $\hat{\mathbf{c}}^{(\mathsf{cnt})}$ has a size of $\frac{\mathsf{csize}}{2}$.
(3) Call $\mathcal{F}_{\mathsf{FSS}}^{\mathsf{Select}}$ with $\hat{\mathbf{c}}^{(\mathsf{cnt})}$ and $\hat{\mathbf{x}}^{\mathsf{diff}}$ as inputs, receive $[\![\hat{\mathbf{q}}]\!]$ as output, where $[\![\hat{\mathbf{q}}]\!]$ has a size of $\frac{\mathsf{csize}}{2}$.
(4) Compute and update $[\![\hat{\mathbf{x}}^{\mathsf{tmp}}]\!] = [\![\hat{\mathbf{q}}]\!] + [\![\hat{\mathbf{x}}^{\mathsf{left}}]\!]$. Then set $\mathsf{csize} = \mathsf{csize}/2$ and $\mathsf{cnt} = \mathsf{cnt} + 1$.
3: **[MaxLocate].** If $\mathsf{csize} = 1$, then:
(1) $\mathcal{P}_0$ treats $\{\hat{\mathbf{c}}^{(i)}\}_{i \in [\mathsf{cnt}]}$ as a tree-structured label $\hat{\mathbf{c}} \in C_{\log N - 1}$, then computes and sends $[\![\mathbf{t}]\!]_0^{\mathsf{B}} = (\hat{\mathbf{c}} \cdot \pi_{\hat{\mathbf{r}}^{-1}}) \oplus [\![(\mathbf{m} \cdot \pi_{\hat{r}^{-1}})]\!]_0^{\mathsf{B}}$ to $\mathcal{P}_1$.
(2) $\mathcal{P}_1$ computes $\mathbf{t} = [\![\mathbf{t}]\!]_0^{\mathsf{B}} \oplus [\![(\mathbf{m} \cdot \pi_{\hat{r}^{-1}})]\!]_1^{\mathsf{B}}$.
(3) $\mathcal{P}_1$ sets $\mathbf{w} \in \{0,1\}^N$, where $\mathbf{w}[j] = 1\{\mathsf{path}_{\mathbf{t}}(j) = \mathsf{Decom}(j-1)\}$ for $j \in [N]$, then sends $\hat{\mathbf{w}} = \mathbf{w} \oplus \alpha$ to $\mathcal{P}_0$. $\mathcal{P}_1$ sets $[\![\mathbf{z}]\!]_1^{\mathsf{B}} = \beta$.
(4) $\mathcal{P}_0$ computes $[\![\mathbf{z}]\!]_0^{\mathsf{B}} = \hat{\mathbf{w}} \cdot \pi_{\hat{\mathbf{r}}^{-1}}^{-1} \oplus \mathbf{\Delta}$.

Algorithm 2: Two-party secure argmax $\Pi_{\mathsf{2PC}}^{\mathsf{SeArg}}$

of the j-th round ($j \in [\log N]$), parties reconstruct the (masked) results $\hat{\mathbf{c}}^j$ from the comparison outputs $[\![\mathbf{c}^j]\!] = \{[\![c_1^j]\!], \ldots, [\![c_{\frac{N}{2^j}}^j]\!]\}$, and update comparison candidates by calling $\mathcal{F}_{\mathsf{FSS}}^{\mathsf{Select}}$.

- Then in the *MaxLocate* part, unlike prior protocols that require N comparisons to determine the index of the max value, we leverage oblivious tree permutation to avoid any comparison. Specifically, the (masked) comparison results $\{\hat{\mathbf{c}}^{(i)}\}_{i \in [\log N]}$ correspond to the tree label $\hat{\mathbf{c}}$ (of depth $\log N - 1$), where $\hat{\mathbf{c}} = \mathbf{c} \oplus \mathbf{r}$. Parties perform an oblivious tree permutation on $\hat{\mathbf{c}}$ using a random permutation derived from $\mathbf{r} \cdot \pi_{\mathbf{r}}^{-1}$, and reveal the result $\mathbf{t}$ to $\mathcal{P}_1$. For each $i \in [N]$, $\mathcal{P}_1$ computes a leaf label vector $\mathbf{w}$ (of length N) by checking whether $\mathsf{path}_{\mathbf{t}}(j) = \mathsf{Decom}(j-1)$ holds. Finally, parties perform an oblivious tree permutation on the leaf labels via the inverse permutation to obtain the (secret-shared) argmax result $[\![\mathbf{z}]\!]^{\mathsf{B}}$.

Theorem 1. *Protocol Π_{SeArg} shown in Algorithm 2 securely realizes $\mathcal{F}_{\mathsf{2PC}}^{\mathsf{SeArg}}$ in the $\mathcal{F}_{\mathsf{FSS}}^{\mathsf{dReLU}}$, $\mathcal{F}_{\mathsf{FSS}}^{\mathsf{Select}}$-hybrid model, in the presence of a semi-honest adversary who can corrupt P_0 or P_1 with static corruption.*

Proof Sketch. We first show the correctness. By construction, there is a trace path on the (tree-structured) comparison results $\mathbf{c}$ such that $\mathsf{path}_{\mathbf{c}}(i^*) = \mathsf{Decom}(i^* - 1)$ for some $i^* \in [N]$, which implies $x^{\mathsf{max}} = x_{i^*}$. The result $\mathbf{z} \in \{0,1\}^N$ therefore indicates the leaf label, subject to $\mathbf{z}[i] = 1\{i = i^*\}$ for $i \in [N]$.

In the protocol, $\mathcal{P}_1$ recovers $\mathbf{t} = (\hat{\mathbf{c}} \cdot \pi_{\hat{\mathbf{r}}^{-1}}) \oplus [(\mathbf{m} \cdot \pi_{\hat{r}^{-1}})]_0^{\mathsf{B}} \oplus [(\mathbf{m} \cdot \pi_{\hat{r}^{-1}})]_1^{\mathsf{B}} = (\hat{\mathbf{c}} \oplus \mathbf{m}) \cdot \pi_{\hat{\mathbf{r}}^{-1}} = (\mathbf{c} \oplus \hat{\mathbf{r}}) \cdot \pi_{\hat{\mathbf{r}}^{-1}}$. By property (2) of tree permutation, $(\mathbf{c} \oplus \hat{\mathbf{r}}) \cdot \pi_{\hat{\mathbf{r}}^{-1}}$ is equal to $\mathbf{c} \circ \hat{\mathbf{r}}^{-1}$. Thus, we have $\mathsf{path}_{\mathbf{c} \circ \hat{\mathbf{r}}^{-1}}(\pi_{\hat{\mathbf{r}}^{-1}}(i^*)) = \mathsf{Decom}(\pi_{\hat{\mathbf{r}}^{-1}}(i^*) - 1)$ according to property (3). This implies that the unique non-zero element in $\mathbf{w}$ is indeed $\pi_{\hat{\mathbf{r}}^{-1}}(i^*)$. The argmax result $\mathbf{z}$ can be recovered by computing $\mathbf{z} = [\mathbf{z}]_0^{\mathsf{B}} \oplus [\mathbf{z}]_1^{\mathsf{B}} = (\mathbf{w} \oplus \alpha) \cdot \pi_{\hat{\mathbf{r}}^{-1}}^{-1} \oplus \mathbf{\Delta} \oplus \beta = \mathbf{w} \cdot \pi_{\hat{\mathbf{r}}^{-1}}^{-1} \oplus \alpha \cdot \pi_{\hat{\mathbf{r}}^{-1}}^{-1} \oplus \mathbf{\Delta} \oplus \beta = \mathbf{w} \cdot \pi_{\hat{\mathbf{r}}^{-1}}^{-1}$. With an inverse permutation $\pi_{\hat{\mathbf{r}}^{-1}}^{-1}$ on $\mathbf{w}$ (by property (1)), $\mathbf{z}$ is obtained.

We next show the semi-honest security. Consider the case $\mathcal{P}_1$ is corrupted. The view of $\mathcal{P}_1$ consists of the messages from $\mathcal{P}_0$, including $[\![\hat{\mathbf{x}}^{\mathsf{diff}}]\!]_0$, $\{[\![\hat{\mathbf{c}}^{(i)}]\!]_0^{\mathsf{B}}\}_{i \in [\log N]}$ and $[\![\mathbf{t}]\!]_0^{\mathsf{B}}$. The security of $\mathcal{F}_{\mathsf{FSS}}^{\mathsf{dReLU}}$ and $\mathcal{F}_{\mathsf{FSS}}^{\mathsf{Select}}$ ensures that the masked values $[\![\hat{\mathbf{x}}^{\mathsf{diff}}]\!]_0$ and $\{[\![\hat{\mathbf{c}}^{(i)}]\!]_0^{\mathsf{B}}\}_{i \in [\log N]}$ are uniformly random. And the randomness of $\mathbf{m}$ ensures that the revealed values $(\mathbf{c} \oplus \hat{\mathbf{r}}) \cdot \pi_{\hat{\mathbf{r}}^{-1}}$ do not leak any information about $\mathbf{c}$, as $\hat{\mathbf{r}} = \mathbf{r} \oplus \mathbf{m}$.

Consider the case $\mathcal{P}_0$ is corrupted. The view of $\mathcal{P}_0$ consists of the messages from $\mathcal{P}_1$, including $[\![\hat{\mathbf{x}}^{\mathsf{diff}}]\!]_1$, $\{[\![\hat{\mathbf{c}}^{(i)}]\!]_1^{\mathsf{B}}\}_{i \in [\log N]}$ and $\hat{\mathbf{w}}$. Similarly, $[\![\hat{\mathbf{x}}^{\mathsf{diff}}]\!]_1$ and $\{[\![\hat{\mathbf{c}}^{(i)}]\!]_1^{\mathsf{B}}\}_{i \in [\log N]}$ are uniformly random. The randomness of α ensures that $\mathbf{w}$ is concealed, so $\mathcal{P}_0$ can learn nothing about the (permuted) output.

Complexity. The FSS-based protocol Π_{SiArg} in [7] requires $2N\ell + N$ bits within $2\log N + 1$ online communication rounds, of which $N\ell$ bits are consumed to identify the index of x^{max} by performing N comparisons.

Our protocol $\Pi_{\mathsf{2PC}}^{\mathsf{SeArg}}$ can be executed within $2\log N + 1$ communication rounds, where overall $N\ell + 2N$ bits are exchanged. The required communication overhead in $\Pi_{\mathsf{2PC}}^{\mathsf{SeArg}}$ is almost half that of Π_{SiArg}. This benefits from the oblivious tree permutation, which cuts the total number of comparisons from $2N - 1$ to $N - 1$, thereby yielding a saving of $N\ell - N$ bits in online communication overhead.

B. Communication Optimization for Node Split

Once the best split feature and the threshold are determined, parties need to perform the node split computation by updating the gradient statistics $[\![\mathbf{g}^{2j}]\!]$ and $[\![\mathbf{g}^{2j+1}]\!]$ for the child nodes $2j$ and $2j + 1$ of the current node j.

We adopt the local-update strategy introduced in [6]. That is, let $\mathcal{P}_i$ hold $\mathbf{s}_i^j \in \{0,1\}^N$ for each node j, where $\mathbf{s}_i^j[k] = 1$ indicates that the k-th sample is available on node j. For the root node, parties hold $\mathbf{s}_0^1 = \mathbf{s}_1^1 = \mathbf{1}^N$ since all the samples are available. For the subsequent nodes, since the feature holder $\mathcal{P}_c$ can learn the best split identifier (f_*^j, b_*^j) on the node j, she can locally update the indicator $\mathbf{s}^j$ as follows: $\mathcal{P}_c$ constructs a vector $\mathbf{s}_*^j \in \{0,1\}^N$, where $\mathbf{s}_*^j[k] = 0$ if the f_*^j-th feature of the k-th sample has a larger value than a threshold defined by the b_*^j-th bucket, vice versa $\mathbf{s}_*^j[k] = 1$. Then, $\mathcal{P}_c$ updates the indicator for child nodes as $\mathbf{s}_c^{2j} = \mathbf{s}_c^j \wedge \mathbf{s}_*^j$ and $\mathbf{s}_c^{2j+1} = \mathbf{s}_c^j \oplus \mathbf{s}_c^{2j}$.

The other party $\mathcal{P}_{1-c}$ keeps $\mathbf{s}_{1-c}^{2j} = \mathbf{s}_{1-c}^{2j+1} = \mathbf{s}_{1-c}^{j}$, as she has no information about the split.

We present the secure node split protocol in Algorithm 3. By calling the FSS-based sub-protocol $\mathcal{F}_{\mathsf{FSS}}^{\mathsf{select}}$ with $\mathbf{s}_*^j$ and $[\![\mathbf{g}^j]\!]$ as inputs, parties receive the output $[\![\mathbf{g}^{2j}]\!]$.

Private inputs: The feature holder $\mathcal{P}_c$ ($c \in \{0,1\}$) holds $\mathcal{I}^{f,b}$ and $\mathbf{m}^{f,b} \in \{0,1\}^N$, where $|\mathcal{I}^{f,b}| \leq N$ for $1 \leq b \leq B$, $1 \leq f \leq F_c$, and $\mathbf{m}^{f,b}[k] = 1\{k \in \mathcal{I}^{f,b}\}$. At a node j, parties $\mathcal{P}_i$ ($i \in \{0,1\}$) hold $\mathbf{s}_i^j$ and $[\![\mathbf{g}^j]\!]_i$, where $\mathbf{g}^j = (g_1^j, \ldots, g_N^j)$.
Public inputs: Public parameters.
Outputs: Parties output $[\![\mathbf{g}^{2j}]\!]$ and $[\![\mathbf{g}^{2j+1}]\!]$.
Preprocessing:
1: Parties receive $[\![\mathbf{u}]\!]$ from $\mathcal{F}_{\mathsf{FSS}}^{\mathsf{Pre}}$ as (shared) masks for the arithmetic inputs, and $\mathcal{P}_c$ receives $\mathbf{r}$ for the boolean inputs.
2: Parties receive $[\![\mathbf{v}]\!]$ from $\mathcal{F}_{\mathsf{FSS}}^{\mathsf{Pre}}$ as (shared) masks for the outputs.
Protocol:
1. $\mathcal{P}_c$ sets $\mathbf{s}_*^j \in \{0,1\}^N$ where $\mathbf{s}_*^j[k] = 1\{\sum_{b \leq b_*^j} \mathbf{m}^{f_*^j, b}[k] > 0\}$, and locally updates $\mathbf{s}_c^{2j} = \mathbf{s}_c^j \wedge \mathbf{s}_*^j$ and $\mathbf{s}_c^{2j+1} = \mathbf{s}_c^j \oplus \mathbf{s}_c^{2j}$.
2: $\mathcal{P}_c$ computes $\hat{\mathbf{s}}_*^j = \mathbf{s}_*^j \oplus \mathbf{r}$, and sends $\hat{\mathbf{s}}_*^j$ to $\mathcal{P}_{1-c}$.
3: $\mathcal{P}_{1-c}$ sets $\mathbf{s}_{1-c}^{2j} = \mathbf{s}_{1-c}^{2j+1} = \mathbf{s}_{1-c}^j$.
4: Compute $[\![\hat{\mathbf{g}}^j]\!] = [\![\mathbf{g}^j]\!] + [\![\mathbf{u}]\!]$, then reconstruct $\hat{\mathbf{g}}^j = [\![\hat{\mathbf{g}}^j]\!]_c + [\![\hat{\mathbf{g}}^j]\!]_{1-c}$.
5. Call $\mathcal{F}_{\mathsf{FSS}}^{\mathsf{Select}}$ with $\hat{\mathbf{s}}_*^j$, $\hat{\mathbf{g}}^j$ as inputs, receive $[\![\hat{\mathbf{g}}^{2j}]\!]$ as output.
6: Compute $[\![\mathbf{g}^{2j}]\!] = [\![\hat{\mathbf{g}}^{2j}]\!] - [\![\mathbf{r}^{\mathsf{out}}]\!]$, $[\![\mathbf{g}^{2j+1}]\!] = [\![\mathbf{g}^j]\!] - [\![\mathbf{g}^{2j}]\!]$.

Algorithm 3: Two-party secure Node Split $\Pi_{\mathsf{2PC}}^{\mathsf{SeNS}}$

Theorem 2. *Protocol $\Pi_{\mathsf{2PC}}^{\mathsf{SeNS}}$ shown in Algorithm 3 securely realizes $\mathcal{F}_{\mathsf{2PC}}^{\mathsf{SeArg}}$ in the $\mathcal{F}_{\mathsf{FSS}}^{\mathsf{Select}}$-hybrid model, in the presence of a semi-honest adversary who can corrupt P_0 or P_1 with static corruption.*

Proof Sketch. For correctness, the ground sample indicator is $\mathbf{s}^{2j} = \mathbf{s}^j \wedge \mathbf{s}_*^j$ for the left child, and $\mathbf{s}^{2j+1} = \mathbf{s}^j \oplus \mathbf{s}^{2j}$ for the right child. The sample space of the left child is correctly updated according to: $\mathbf{g}^{2j} = \mathbf{s}^{2j} \odot \mathbf{g} = (\mathbf{s}^j \wedge \mathbf{s}_*^j) \odot \mathbf{g} = \mathbf{s}_*^j \odot \mathbf{g}^j$. And for the right child, $[\![\mathbf{g}^{2j+1}]\!] = [\![\mathbf{g}^j]\!] - [\![\mathbf{g}^{2j}]\!]$ holds.

For security, in the online phase, $\mathbf{s}_*^j$ can be directly derived from the best split feature and the threshold (f_*^j, v_*^j) which are only known to the feature holder P_c. the update of child indicators just involves local computation. As a result, the privacy follows trivially in the $\mathcal{F}_{\mathsf{FSS}}^{\mathsf{Select}}$-hybrid.

Complexity. Since $\hat{\mathbf{g}}^j$ are typically already reconstructed in the previous bucket aggregation phase (as in [7]), only $\mathbf{s}_*^j$ has to be masked by P_c and sent to P_{1-c} in the online phase. Our protocol requires only one round online communication with overall N bits to send $\hat{\mathbf{s}}_*^j$. In contrast, [6] uses the Ferret COT [12] to compute the multiplications in $\mathbf{s}_*^j \odot \mathbf{g}^j$, and thus about $O(2N\ell)$ bits are exchanged within a round. Meanwhile, [7] calls $\mathcal{F}_{\mathsf{FSS}}^{\mathsf{Select}}$ twice for successively updating $\mathbf{s}^{2j}$ and $\mathbf{g}^{2j}$, and requires 2 rounds of $4N$ bits. We conclude that our method is more efficient than the previous works in terms of computation and communication.

C. Our Accio Framework

In Algorithm 4, we show how to build the *Accio* framework using the sub-protocols constructed above. Specifically, *Accio* comprises a preprocessing phase, and an online training phase.

Preprocessing. Initially, parties locally partition their samples into buckets for each of the features they holds (*Step 1*), and set up correlated randomness (*Step 2*).

Online Training. At any non-leaf node, parties first compute Gain for all split candidates (*Step 11-15*). Then, the best split is identified by computing the maximum Gain at the current node (*Step 16-18*). With the best split identifiers, parties update the gradient statistics for the child nodes (*Step 19-20*).

Then for each leaf node, parties compute the leaf weight $[\![w]\!]$ (*Step 23*), and update the (shared) prediction score $[\![\check{\mathbf{y}}]\!]$ (*Step 24-26*). Finally, each party keeps the split identifiers it holds along with the (secret-shared) leaf weights (*Step 27*).

V. Evaluations

A. Evaluation Setup

Tested Environment. All our experiments were conducted on Intel(R) Xeon(R) Gold 6230R CPU at 2.1GHz with 256GB RAM. Each party in *Accio* is simulated by a separate process with 6 threads. We run our benchmarks in two network settings using the `tc` tool[1]: LAN (1Gbps and 0.2ms of round-trip time) and WAN (100Mbps and 40ms of round-trip time), aligning with the configuration utilized in [7].

Implementation. We set the arithmetic sharing length $\ell = 64$ and the fraction precision $f = 20$ for fixed-point computation. By approximating the Sigmoid as a piece-wise linear function, $\mathcal{F}_{\mathsf{2PC}}^{\mathsf{Grad}}$ and $\mathcal{F}_{\mathsf{2PC}}^{\mathsf{Hess}}$ are realized via parallel executions of $\mathcal{F}_{\mathsf{FSS}}^{\mathsf{dReLU}}$. We implement $\mathcal{F}_{\mathsf{2PC}}^{\mathsf{Gain}}$ and $\mathcal{F}_{\mathsf{2PC}}^{\mathsf{Leaf}}$ via the division sub-protocol $\mathcal{F}_{\mathsf{2PC}}^{\mathsf{recip}}$ [13]. $\mathcal{F}_{\mathsf{2PC}}^{\mathsf{BestParty}}$ is realized using the sub-protocol $\mathcal{F}_{\mathsf{FSS}}^{\mathsf{dReLU}}$.

Baseline. For accuracy verification, we compare *Accio* with the plaintext XGBoost[2] library. For efficiency evaluation, we compare *Accio* with the state-of-the-art two-party framework SiGBDT [7]. For all experiments, we only focus on the online training stage in two network settings.

B. Microbenchmarks

As shown in TABLE II, *Accio*'s argmax protocol reduces (online) running time and communication significantly compared with existing works. As detailed in Section IV-A, *Accio*'s argmax protocol benefits from the application of oblivious tree permutation technique, and achieves its enhanced performance by nearly halving the invocations to comparative instances.

We also benchmark the node splits in Squirrel [6] and SiGBDT [7], which are COT-based and FSS-based approaches, respectively. As shown in TABLE II, our node splits are more (online) efficient in terms of computation time and communication cost than existing works.

[1]https://man7.org/linux/man-pages/man8/tc.8.html
[2]https://github.com/dmlc/xgboost.git

Input: $\mathsf{Input}_0 = (\mathbf{X}_0 \in \mathbb{R}^{N\times F_0}, \mathbf{y} \in \mathbb{R}^N)$ and $\mathsf{Input}_1 = (\mathbf{X}_1 \in \mathbb{R}^{N\times F_1})$. Hyperparameters $\mathsf{pp} = (T > 0, D > 0, B > 0, \gamma > 0)$. Secretly shared stateful values $\mathsf{state} = (\llbracket \mathbf{g} \rrbracket, \llbracket \mathbf{h} \rrbracket, \llbracket \tilde{\mathbf{y}}_0 \rrbracket)$.

Output: Output_0^{Π} and Output_1^{Π} (as defined in Algorithm 1).

[Initialization.] $\mathcal{P}_i$ locally partitions her samples $\mathbf{X}_i$ into buckets, written as $\mathbf{m}^{f,b} \in \{0,1\}^N$ where $\mathbf{m}^{f,b}[j] = 1\{j \in \mathcal{I}^{f,b}\}$ and $\mathcal{I}^{f,b}$ denotes the indices of samples sorted into the b-th bucket of feature f for $f \in [F_i]$ and $b \in [B]$.

Both parties run $\mathcal{F}_{\mathsf{FSS}}^{\mathsf{Pre}}$, and receive correlated randomness.

for $t = 1, 2, \ldots, T$ **do**

  if $t > 1$ **then**

    Jointly compute $\llbracket \mathbf{g} \rrbracket \leftarrow \mathcal{F}_{\mathsf{2PC}}^{\mathsf{Grad}}(\llbracket \mathbf{y} \rrbracket, \llbracket \tilde{\mathbf{y}} \rrbracket)$ and $\llbracket \mathbf{h} \rrbracket \leftarrow \mathcal{F}_{\mathsf{2PC}}^{\mathsf{Hess}}(\llbracket \mathbf{y} \rrbracket, \llbracket \tilde{\mathbf{y}} \rrbracket)$.

  end

  for *non-leaf nodes* $j = 1, 2, \ldots, n_{\mathsf{non}}$ **do**

    if $j = 1$ **then**

      $\mathcal{P}_i$ sets $\mathbf{s}_i^{(1)} = \mathbf{1}^N$ with all 1s, and sets $\llbracket \mathbf{g}^{(1)} \rrbracket_i = \llbracket \mathbf{g} \rrbracket_i$ and $\llbracket \mathbf{h}^{(1)} \rrbracket_i = \llbracket \mathbf{h} \rrbracket_i$.

    end

    [Compute Gain **for all split candidates.] for** $f \in [F]$ *and* $b \in [B]$ **do**

      Jointly run $\llbracket G^{f,b} \rrbracket$(and $\hat{\mathbf{g}}^{(j)}$) $\leftarrow \mathcal{F}_{\mathsf{2PC}}^{\mathsf{Buc}}(\llbracket \mathbf{g}^{(j)} \rrbracket, \mathbf{m}^{f,b})$, and $\llbracket H^{f,b} \rrbracket$(and $\hat{\mathbf{h}}^{(j)}$) $\leftarrow \mathcal{F}_{\mathsf{2PC}}^{\mathsf{Buc}}(\llbracket \mathbf{h}^{(j)} \rrbracket, \mathbf{m}^{f,b})$.

    end

    $\mathcal{P}_i$ locally partitions and aggregates both $\llbracket G^{f,b} \rrbracket_i$ and $\llbracket H^{f,b} \rrbracket_i$ into left and right parts according to $F \times (B-1)$ split candidates, written as $(\llbracket \mathbf{G}^L \rrbracket, \llbracket \mathbf{G}^R \rrbracket)$ and $(\llbracket \mathbf{H}^L \rrbracket, \llbracket \mathbf{H}^R \rrbracket)$.

    Jointly run $\llbracket \mathbf{Gain}^{(j)} \rrbracket \leftarrow \mathcal{F}_{\mathsf{2PC}}^{\mathsf{Gain}}(\llbracket \mathbf{G}^L \rrbracket, \llbracket \mathbf{G}^R \rrbracket, \llbracket \mathbf{H}^L \rrbracket, \llbracket \mathbf{H}^R \rrbracket)$.

    [Find the best split with maximum Gain**.]** Jointly compute $\llbracket \mathbf{z}^{(j)} \rrbracket^{\mathsf{B}} \leftarrow \mathcal{F}_{\mathsf{2PC}}^{\mathsf{SeArg}}(\llbracket \mathbf{Gain}^{(j)} \rrbracket)$ (see Algorithm 2).

    Open an indicator $\llbracket c \rrbracket^{\mathsf{B}} \leftarrow \mathcal{F}_{\mathsf{2PC}}^{\mathsf{BestParty}}(\llbracket \mathbf{z}^{(j)} \rrbracket^{\mathsf{B}}, F_0)$ to both parties, where $c \in \{0,1\}$ pointing to $\mathcal{P}_c$ who hold the best split feature.

    Open the split identifier $(f_*^{(j)}, b_*^{(j)})$ to $\mathcal{P}_c$ who then writes them into $\mathcal{C}_c[j]$, while $\mathcal{P}_{1-c}$ writes $\perp$ to $\mathcal{C}_{1-c}[j]$.

    [Split the current node by sample space update.] Jointly compute $\llbracket \mathbf{g}^{(2j)} \rrbracket$ and $\llbracket \mathbf{g}^{(2j+1)} \rrbracket$ using $\mathcal{F}_{\mathsf{2PC}}^{\mathsf{SeNS}}$ (see Algorithm 3).

    Jointly compute $\llbracket \mathbf{h}^{(2j)} \rrbracket$ and $\llbracket \mathbf{h}^{(2j+1)} \rrbracket$ using $\mathcal{F}_{\mathsf{2PC}}^{\mathsf{SeNS}}$ (see Algorithm 3).

  end

  for *leaf nodes* $j = n_{\mathsf{non}} + 1, \ldots, n_{\mathsf{non}} + n_{\mathsf{leaf}}$ **do**

    Jointly compute the weight $\llbracket w^{(j)} \rrbracket$ using $\mathcal{F}_{\mathsf{2PC}}^{\mathsf{Leaf}}$.

    Jointly compute $\llbracket \mathbf{s}^{(j)} \rrbracket$ using $\mathcal{F}_{\mathsf{2PC}}^{\mathsf{AND}}$ with $\mathbf{s}_0^{(j)}$ and $\mathbf{s}_1^{(j)}$ as inputs, and $\llbracket \mathbf{v}^{(j)} \rrbracket \leftarrow \mathcal{F}_{\mathsf{FSS}}^{\mathsf{Select}}(\llbracket \mathbf{s}^{(j)} \rrbracket, \llbracket w^{(j)} \rrbracket)$.

  end

  Update the predication scores $\llbracket \tilde{\mathbf{y}} \rrbracket = \sum_{k=n_{\mathsf{non}}+1}^{n_{\mathsf{non}}+n_{\mathsf{leaf}}} \llbracket \mathbf{v}^{(j)} \rrbracket + \llbracket \tilde{\mathbf{y}} \rrbracket$.

  $\mathcal{P}_i$ writes $\llbracket w^{(j)} \rrbracket_i$ to Output_i^{Π}.

end

Algorithm 4: Secure GBDT Training Framework *Accio*

TABLE II: Compare our optimizations with existing works.

Argmax (10^4 inputs)	[6] (Squirrel)	[7] (SiGBDT)	Ours
Times-LAN (s)	6.64	0.52	0.27
Times-WAN (s)	18.9	2.11	1.72
Comm. (MB)	4.95	0.16	0.083
Node split (10^6 inputs)	**[6] (Squirrel)**	**[7] (SiGBDT)**	**Ours**
Times-LAN (s)	2.74	1.94	1.02
Times-WAN (s)	9.90	3.79	1.96
Comm. (MB)	15.26	0.50	0.13

TABLE III: Accuracy of *Accio* vs. XGBoost on four real-world datasets for classification (Cls.) and regression (Reg.).

Task	Dataset	N	F	Metric	*Accio*	Plaintext
Cls.	Cancer	569	32	ACC	0.98	0.98
	Phishing	11055	67		0.96	0.96
Reg.	Diabetes	442	10	RMSE	52.89	52.95
	Concrete	1030	8		5.17	5.30

C. Accuracy of Accio

We use the plaintext XGBoost library to train a model over public datasets as baseline, then use *Accio* to train a model on the same datasets. We employ four real-world datasets from the UCI repository[3], and list the detailed information of these datasets in TABLE III. We set the training parameters as follows: tree number $T = 20$, max depth $D = 4$ and max bucket size $B = 16$. Each dataset is divided into a training set and a test set with a ratio of $4:1$. In *Accio*, features are evenly allocated to both parties. We choose Root Mean Square Error (RMSE) as the evaluation metric for regression task, and accuracy (ACC) for classification task.

We show the average accuracy of the five runs in TABLE III. The accuracy of *Accio* and the plaintext model are almost the same, which proves that *Accio* can accurately train a GBDT.

D. Efficiency of Accio

To evaluate the efficiency, we generate random synthetic datasets with identical hyperparameters to [7]: sample amount $N = 10,000$, feature size $F = 10$, max depth $D = 4$, and max bucket size $B = 8$. We compare the efficiency of *Accio* with the state-of-the-art two-party framework SiGBDT [7].

1) End-to-End Training Time: We set the tree number $T = 10$, and measure the end-to-end training time of *Accio*

[3]https://archive.ics.uci.edu

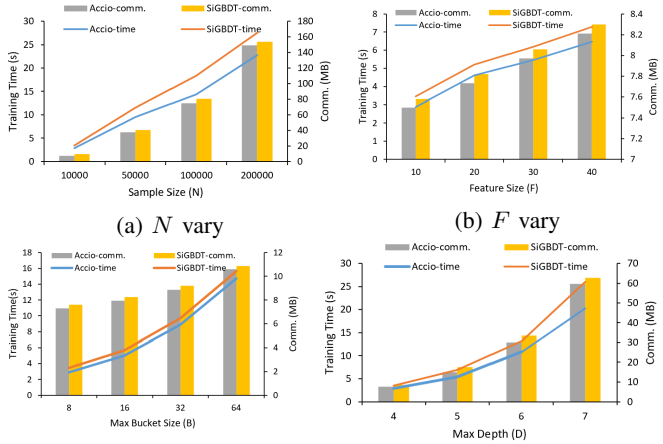
compared to the baseline framework. In TABLE IV, we give the average running time per tree. *Accio* is faster than SiGBDT in both LAN and WAN settings.

TABLE IV: End-to-end running time (s) and total communication cost (MB) per tree.

Framework	Parameters	Settings	Times	Comm.
Accio	$N = 10000, F = 10$	LAN	2.88	7.50
SiGBDT	$B = 8, D = 4$		3.45	7.58
Accio	$\mathbf{N = 50000}, F = 10$	LAN	9.50	37.32
SiGBDT	$B = 8, D = 4$		11.48	37.68
Accio	$N = 10000, \mathbf{F=20}$	LAN	4.60	7.73
SiGBDT	$B = 8, D = 4$		5.50	7.82
Accio	$N = 10000, F = 10$	LAN	5.02	7.95
SiGBDT	$\mathbf{B=16}, D = 4$		5.68	8.05
Accio	$N = 10000, F = 10$	LAN	5.40	14.96
SiGBDT	$B = 8, \mathbf{D=5}$		6.17	15.42
Accio	$N = 10000, F = 10$	**WAN**	12.19	7.50
SiGBDT	$B = 8, D = 4$		14.18	7.58

2) Total Communication: In TABLE IV, we also record the online communication cost of both frameworks for training one tree. Theoretically, the two frameworks have identical round complexity. TABLE IV demonstrates that *Accio* incurs slightly lower communication costs, and this can be attributed to our communication optimizations for linear and non-linear operations. Indeed, the communication costs rise exponentially with the depth of the tree D, and our advantage will become increasingly apparent as the training size (especially max bucket size and max depth size) increases.

3) Scalability Test: To analyze the scalability of *Accio*, we vary training parameters based on the default settings and study the impact of increased training size on each framework. As shown in Fig. 2, the running time and communication costs of both frameworks increase at roughly the same rate.



(a) N vary (b) F vary

(c) B varies (d) D varies

Fig. 2: Effect of parameters in *Accio*.

VI. Conclusion

In this paper, we propose *Accio*, a two-party framework for secure GBDT training in the preprocessing model. *Accio* is built upon two key optimizations. First, we devise a leaner argmax protocol based on oblivious tree permutation, which reduces the number of required comparisons, saving nearly half of the online communication cost compared to its FSS-based counterpart. Second, we design a node splitting protocol with a local update strategy, enabling parties to locally update the sample space of child nodes, thereby reducing the total number of online rounds. Experimental results show that *Accio* outperforms the state-of-the-art SiGBDT framework in both LAN and WAN settings, while achieving accuracy comparable to plaintext XGBoost. Future work will focus on eliminating the trusted dealer and reducing preprocessing overhead, while extending Accio to support malicious security.

Acknowledgment

This research was supported by the National Natural Science Foundation of China under Grant No.62372446.

References

[1] 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.

[2] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017*, 2017, pp. 3146–3154.

[3] P. Mohassel and Y. Zhang, "Secureml: A system for scalable privacy-preserving machine learning," in *2017 IEEE Symposium on Security and Privacy, SP 2017*, 2017, pp. 19–38.

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

[5] 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 *CIKM '21: The 30th ACM International Conference on Information and Knowledge Management*, 2021, pp. 443–452.

[6] 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, USENIX Security 2023*, 2023, pp. 6435–6451.

[7] Y. Jiang, F. Mei, T. Dai, and Y. Li, "Sigbdt: Large-scale gradient boosting decision tree training via function secret sharing," in *Proceedings of the 19th ACM Asia Conference on Computer and Communications Security*, 2024.

[8] E. Boyle, N. Chandran, N. Gilboa, D. Gupta, Y. Ishai, N. Kumar, and M. Rathee, "Function secret sharing for mixed-mode and fixed-point secure computation," in *Advances in Cryptology - EUROCRYPT 2021 - 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, ser. Lecture Notes in Computer Science, vol. 12697, 2021, pp. 871–900.

[9] N. Jawalkar, K. Gupta, A. Basu, N. Chandran, D. Gupta, and R. Sharma, "Orca: Fss-based secure training and inference with gpus," in *IEEE Symposium on Security and Privacy, SP 2024*, 2024, pp. 597–616.

[10] N. Cheng, N. Gupta, A. Mitrokotsa, H. Morita, and K. Tozawa, "Constant-round private decision tree evaluation for secret shared data," *Proc. Priv. Enhancing Technol.*, vol. 2024, no. 1, pp. 397–412, 2024.

[11] R. Canetti, "Security and composition of cryptographic protocols: A tutorial," in *Secure Multi-Party Computation*, ser. Cryptology and Information Security Series, 2013, vol. 10, pp. 61–119.

[12] K. Yang, C. Weng, X. Lan, J. Zhang, and X. Wang, "Ferret: Fast extension for correlated OT with small communication," in *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 1607–1626.

[13] O. Catrina and A. Saxena, "Secure computation with fixed-point numbers," in *Financial Cryptography and Data Security, 14th International Conference, FC 2010*, ser. Lecture Notes in Computer Science, vol. 6052, 2010, pp. 35–50.