Guard-GBDT: Efficient Privacy-Preserving Approximated GBDT Training on Vertical Dataset (Full Version)

Anxiao Song
Xidian University
Xian, China
songanxiao@stu.xidian.edu.cn

Shujie Cui
Monash University
Melbourne, Australia
shujie.cui@monash.edu

Jianli Bai
Singapore Management University
Singapore, Singapore
jianlibai@smu.edu.sg

Ke Cheng
Xidian University
Xian, China
chengke@xidian.edu.cn

Yulong Shen
Xidian University
Xian, China
ylshen@mail.xidian.edu.cn

Giovanni Russello
University of Auckland
Auckland, New Zealand
g.russello@auckland.ac.nz

ABSTRACT

In light of increasing privacy concerns and stringent legal regulations, using secure multiparty computation (MPC) to enable collaborative GBDT model training among multiple data owners has garnered significant attention. Despite this, existing MPC-based GBDT frameworks face efficiency challenges due to high communication costs and the computation burden of non-linear operations, such as division and sigmoid calculations. In this work, we introduce Guard-GBDT, an innovative framework tailored for efficient and privacy-preserving GBDT training on large-scale vertical datasets. Guard-GBDT bypasses MPC-unfriendly division and sigmoid functions by using more streamlined approximations and reduces communication overhead by compressing the messages exchanged during gradient aggregation. We implement a prototype of Guard-GBDT and extensively evaluate its performance and accuracy on various real-world datasets. The results show that Guard-GBDT outperforms state-of-the-art HEP-XGB (CIKM'21) and SiGBDT (ASIA CCS'24) by 1.39× ~ 12.21× on LAN network and 2.39× ~ 4.17× on WAN network. Guard-GBDT also achieves comparable accuracy with SiGBDT and plaintext XGBoost (better than HEP-XGB), which exhibits a deviation of ±1% to ±2% only. Our implementation code is provided at https://github.com/NSS-01/Guard-GBDT.git.

PVLDB Reference Format:
Anxiao Song, Shujie Cui, Jianli Bai, Ke Cheng, Yulong Shen, and Giovanni Russello. Guard-GBDT: Efficient Privacy-Preserving Approximated GBDT Training on Vertical Dataset (Full Version). PVLDB, 14(1): XXX-XXX, 2020.
doi:XX.XX/XXX.XX

1 INTRODUCTION

Gradient Boosting Decision Tree (GBDT) and its variants such as XGBoost[11] and LightGBM[28] are tree-based machine learning (ML) algorithms known for their high performance and strong interpretability and have been widely used to increase productivity in industries such as finance[37], recommendation services[4], and threat analysis[21]. For a high-quality GBDT model, a business organization usually needs to train the model jointly by integrating more data from different owners. However, collecting data from multiple data owners directly for centralized model training, such as patient medical records from various hospitals, raises privacy concerns and can violate government regulations[17]. In this case, cross-silo privacy-preserving ML[33] offers an appealing solution to connect "data silos" among organizations, which enables multiple parties to collaboratively train a GBDT model with the protection of their privacy.

Secure multiparty computation (MPC) is a technique that allows multiple parties to compute a function while maintaining data privacy. In recent years, a series of works [1, 15, 19, 27, 30–32, 36, 40, 41] have used MPC to train privacy-preserving GBDT models, and they can be categorized into two main types according to data partitioning strategies: horizontal GBDT[1, 15, 30, 31, 36] and vertical GBDT [19, 27, 32, 40, 41]. Horizontal GBDT is a row-wise distributed algorithm where each party holds a subset of samples with identical features. In MPC-based horizontal GBDT, the training phase typically requires secure feature discretization and secure row-wise feature re-sorting over encrypted data to ensure privacy and correctness. However, this approach cannot be directly extended to vertical GBDT frameworks due to structural differences in data partitioning and feature ownership. In vertical GBDT, a column-wise algorithm, each party possesses independent features, thus feature re-sorting is unnecessary. Moreover, the split information for a tree node can be revealed to the party owning the corresponding feature while remaining hidden from other parties. This allows each party to perform certain operations, such as feature discretization (converting continuous values into discrete bins or categories) and comparing features with split thresholds, over unencrypted data. In this context, the primary focus of vertical GBDT is on efficiently determining the best feature split at each tree node. This process involves numerous complex operations, including division and gradient aggregation, to optimize the model's performance while maintaining data privacy. This paper focuses on vertical GDBT. However, existing MPC-based vertical GBDT approaches face two significant challenges when applied to large-scale datasets.

Data inflation in private gradient aggregation. GBDT is an ensemble of decision trees, where each new tree uses aggregated first-order gradients $\mathbf{g}$ and second-order gradients $\mathbf{h}$ to guide tree node splits for optimal performance. In plaintext, $\mathbf{g}$ and $\mathbf{h}$ range from $[-1, 1]$. In the context of MPC, these gradients however are encoded into ℓ-bit fixed-point integers with 1-bit sign and ℓ_f-bit fraction, where only the fraction and sign carry significant data, as shown in Figure 1. Usually, $\ell = 64$ or $\ell = 32$ and $\ell_f = 16$. This encoding method leads to data inflation after encryption, which increases communication overhead. For example, approaches like

Table 1: Comparison with existing privacy-preserving vertical GBDT works.

Approach	System model	Primitive	Communication Round	Communication cost	Computation cost	Leakage
ELXGB[41]	2PC	PHE	$O(BF)$	$O(BNF\log qp)$	$C_{M_1}O(BFN)$	G, H, Tree shape
Pivot[40]	3PC/Multi-PC	PHE, ASS, GC	$O(BF(\ell^2+\ell+\log\ell))$	$O(BNF(\log qp+\ell^2))$	$(C_{M_1}+C_{M_2}+C_{c_1}+C_d)O(BNF)$	Tree shape
HEP-XGB[19]	2PC	PHE, ASS	$O(F\ell^2+\log(BF\ell))$	$O(BNF(\log q^2p+\ell^2))$	$(C_{M_1}+C_{M_2}+C_{c_2}+C_d)O(BNF)$	Tree shape
Squirrel[32]	2PC	FHE, ASS	$O(F\ell^2+\log(BF\ell))$	$O(BNF(\log_2 q+\ell^2))$	$(C_{M_3}+C_{M_2}+C_d+C_{c_2})O(BF)$	Tree shape
SiGBDT[27]	2PC	FSS, ASS	$O(F\ell^2+\log(BF))$	$O(BNF\ell^2)$	$(C_s+C_{M_2}+C_d+C_{c_3})O(BF)$	Tree shape
Guard-GBDT	**2PC**	**FSS, ASS**	$\boldsymbol{O(F+\log(BF))}$	$\boldsymbol{O(BNF\ell)}$	$\boldsymbol{(C_{M_2}+C_{c_3})O(BF)}$	**Tree shape**

The communication and computation costs summarised in the table are for training one tree node. G and H represent aggregated statistics corresponding to the first-order and second-order gradients, respectively; ℓ is the bit-length of additive secret sharing; q and p are two large prime numbers in homomorphic encryption. B, F, and N are the number of buckets, features, and samples, respectively; C_{M_1}, C_{M_2} and C_{M_3} are computation costs of multiplication in PHE, ASS, and FHE, respectively; C_{c_1}, C_{c_2} and C_{c_3} are computation costs of comparison in Garbled Circuits (GC), ASS, and FSS, respectively; C_s and C_d are computation costs of the *select* protocol of FSS and the division protocol, respectively; All other operations are local operations, and the computational cost is negligible.

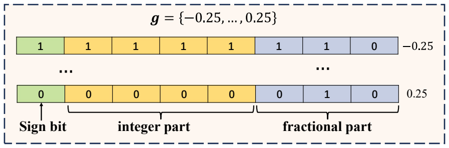


Figure 1: Fixed-point encoding for the first-order gradients. In plaintext, $g_i \in \mathbf{g}$ is in the range $[-1, 1]$. In MPC, g_i is encoded into a fixed-point number. The fraction and sign of g_i carry significant data. However, the integer part of g_i is filled with 1's for positive values or 0's for negative values, resulting in redundant information.

HEP-XGB [19] leverage partially homomorphic encryption (PHE) and additive secret sharing (ASS) to train the GDBT model, resulting in ciphertexts with 64× the traffic of plaintexts as PHE expands gradients to 2048-bit integers. To improve efficiency, Squirrel [32] utilizes lattice-based fully homomorphic encryption (FHE) with SIMD to pack multiple ciphertexts into one. However, it introduces frequent conversions between FHE and ASS, leading to additional communication. More recently, SiGBDT [27] introduces function secret sharing (FSS) to reduce communication overhead, using an ℓ-bit random value r to mask private $\mathbf{g}$ and $\mathbf{h}$ over the ring $\mathbb{Z}_{2^\ell}$. Despite this, $\ell - \ell_f$ bits of inflation persist. Thus, our first challenge is to develop a communication-friendly gradient aggregation protocol.

Inefficient and complicated non-linear functions. GBDT algorithm involves complex non-linear operations, such as division and sigmoid functions, for each node training. Although these operations are easily computed in plaintext, they require substantial effort to be implemented under MPC. Previous works often utilize Goldschmidt[18] or Newton methods[20] to compute a division approximation through many iterations, and employ approximations based on Fourier series[32] or Taylor polynomials[43] to implement the sigmoid function. Both of them introduce heavy computation overheads, making the schemes still not practical for large-scale datasets. Thus, our second challenge is to design efficient solutions for these complex non-linear functions.

Our contributions. In this paper, we propose Guard-GBDT, a secure efficient two-party (2PC) framework for training GBDT models over large-scale vertically distributed datasets. As done in SiGBDT, Guard-GBDT leverages ASS and FSS to protect data as they can offer lower communication and simpler computations than PHE, FHE, and Garbled Circuits (GC) [13, 27]. Unlike SiGBDT, we use lookup-table-based approximations to eliminate inefficient divisions and sigmoid functions. Moreover, we present a novel division-free split gain metric. To further reduce communication, Guard-GBDT compresses the transmitted data during gradient aggregation by focusing on protecting significant data bits only, rather than entire data bits. As a result, as shown in Table 1, Guard-GBDT achieves lower communication and computation costs while maintaining comparable accuracy[1]. Our contributions and techniques can be summarized as follows:

- **MPC-friendly approximations for divisions and sigmoid functions.** Guard-GBDT eliminates the need for sigmoid function calculations and division operations by implementing a lookup table-based approximation. This approximation is used for both sigmoid and leaf prediction weights, making it particularly friendly to MPC.
- **Division-free split gain metric**. Guard-GBDT introduces a novel division-free split gain. It provides faster and more MPC-friendly operations than traditional division-based methods and renders an order-of-magnitude improvement in the computation of secure GBDT training.
- **Communication-friendly aggregation protocol.** Guard-GBDT designs a communication-efficient aggregation protocol that compresses intermediate communication messages by employing dynamically compact random bits to mask only the significant bits in private data, rather than using fixed-length randomness to mask all bits in existing approaches. This strategy reduces unnecessary data transmission, improving both communication efficiency and overall performance.
- **Extensive evaluations.** We implement a prototype of Guard-GBDT and extensively evaluate its performance on various real-world datasets. We compare Guard-GBDT with state-of-the-art GBDT training frameworks such as HEP-XGB and SiGBDT in both LAN and WAN networks and run our experiments with various training parameters. The experimental results show that Guard-GBDT outperforms the state-of-the-art frameworks by 1.39× ~ 12.21× on LAN and 2.39× ~ 4.19× on WAN.

[1] For fairness, we limit our comparison to secure vertical GBDT methods.

2 RELATED WORKS

MPC is a robust tool with strong security guarantees, which is widely utilized in core business areas to construct privacy-preserving computing tasks. Existing works use ASS or homomorphic encryption (HE) protocols to compute linear functions, and garbled circuit (GC) or FSS is used to compute non-linear functions. These works also adopt an offline-online computation paradigm, shifting most computation and communication overhead to the offline stage to achieve a more efficient online phase. Lindell et al.[30] proposed the first MPC-based GBDT using Oblivious Transfer (OT) and GC on a horizontally partitioned dataset between two parties. However, this approach suffers from heavy online communication rounds and overhead. Subsequently, Hoogh et al.[15] and Abspoel et al.[1] utilized ASS protocols to construct privacy-preserving GBDTs, aiming to enhance performance.

To collaboratively build models among different organizations holding data on the same samples but with different features, the MPC-based vertical GBDT framework has recently become a vibrant area of research. SecureBoost[12] first was implemented using Paillier HE and GC to train a GBDT model on two-party vertical datasets, where the gradients are summed using Paillier HE by the passive parties and decrypted by the active party. However, it incurs high communication and computational costs during each training epoch and potentially leaks data labels through the weights sent to the passive party. Zhu et al. [41] introduced a novel vertical GBDT framework, ELXGB, where clients locally compute partial gradient sums, encrypt them using HE, and forward them to an aggregation server. To safeguard data labels, noise is added via Differential Privacy (DP). Despite these protections, data labels might still be recoverable through model inversion attacks, and the training scheme incurs increasing communication costs over more training epochs. Following these developments, Pivot[40] and HEP-XGB[19] employed HE and ASS to enhance computational efficiency. Despite these improvements, the operations involved in frequent encryption and decryption operations of HE remain prohibitively expensive for GBDT training. Building on these advancements, Squirrel[32] and SiGBDT[27] have utilized lattice-based homomorphic encryption (Learning with Errors and its ring variant) and FSS to further optimize communication and computational complexity, thus achieving superior HE performance. However, transmitting their encrypted gradients still imposes a significant burden on network bandwidth due to data inflation after encryption.

3 PRELIMINARIES

3.1 Additive Secret Sharing

In this work, we use 2-out-of-2 additive secret sharing[10] (ASS) to implement secure linear operations. ASS includes two different types of secret sharing: one is arithmetic secret sharing $\langle x \rangle$, which is used for linear arithmetic operations; the other is boolean secret sharing $[[x]]$, which is applied for boolean operations. The details of each type in ASS are as follows:

Arithmetic sharing. For a given ℓ-bit value $x \in \mathbb{Z}_{2^\ell}$, its arithmetic sharing is denoted as $\langle x \rangle = (\langle x \rangle_0, \langle x \rangle_1)$, where $\langle x \rangle_b$ is held by party P_b, $x = (\langle x \rangle_0 + \langle x \rangle_1) \mod 2^\ell$, and $b \in \{0, 1\}$. For brevity, we omit the operation of *mod* 2^ℓ in the rest of the paper. We have the following secure primitives:

- $\langle x \rangle = \mathbf{Share}(x)$: The party holding x, say P_b, chooses a random value $r_x \in \mathbb{Z}_{2^\ell}$, set $\langle x \rangle_b = x - r_x \mod 2^\ell$, and send r_x to the other party P_{1-b} as $\langle x \rangle_{1-b}$.
- $x = \mathbf{Open}(\langle x \rangle)$: Given $\langle x \rangle$, each party P_b exchanges their respective secret sharing shares $\langle x \rangle_b$ to recover x by doing $x = \langle x \rangle_0 + \langle x \rangle_1$.
- $\langle z \rangle = \langle x \rangle + \langle y \rangle$: Given $\langle x \rangle$ and $\langle y \rangle$, each party P_b gets the secret sharing of $z = x + y$ by locally computing $\langle z \rangle_b = \langle x \rangle_b + \langle y \rangle_b$.
- $\langle z \rangle = \langle x \rangle + c$: Given $\langle x \rangle$ and a constant c, each party P_b gets the secret sharing of $z = x + c$ by locally computing $\langle z \rangle_b = \langle x \rangle_b + b \cdot c$.
- $\langle z \rangle = \langle x \rangle \cdot \langle y \rangle$: Given $\langle x \rangle$ and $\langle y \rangle$, the secret sharing of $z = x \cdot y$ can be computed using Beaver multiplication triplet technique [3], i.e., $\langle r_z \rangle = \langle r_x \rangle \cdot \langle r_y \rangle$ are pre-generated in offline phase by P_0 and P_1. In online phase, each party P_b locally computes masked $\langle \hat{x} \rangle_b = \langle x \rangle_b - \langle r_x \rangle_b$ and $\langle \hat{y} \rangle_b = \langle y \rangle_b - \langle r_y \rangle_b$. Next, P_0 and P_1 run $\hat{x} = \mathbf{Open}(\langle \hat{x} \rangle)$ and $\hat{y} = \mathbf{Open}(\langle \hat{y} \rangle)$. Finally, each party P_b computes $\langle z \rangle_b = b \cdot \hat{x} \cdot \hat{y} + \hat{y} \cdot \langle x \rangle_b + \hat{x} \cdot \langle y \rangle_b + \langle r_c \rangle_b$. It requires 1 round and 2ℓ bits of communication to exchange shares of $\hat{x}$ and $\hat{y}$.
- $\langle z \rangle = c \cdot \langle x \rangle$: Given $\langle x \rangle$ and a constant c, P_b gets the secret sharing of $z = c \cdot x$ by locally computing $\langle z \rangle_b = c \cdot \langle x \rangle_b$.

Boolean sharing For a one-bit value x, its boolean sharing is denoted as $[[x]] = ([[x]]_0, [[x]]_1)$, where $[[x]]_b$ is held by party P_b and $x = [[x]]_0 \oplus [[x]]_1$. Boolean sharing is similar to arithmetic secret sharing but with a key difference: in boolean sharing, the addition (+) and multiplication ($\cdot$) are replaced by bit-wise operations $\oplus$ (XOR) and $\wedge$ (AND).

3.2 Function Secret Sharing

Our work also introduces a two-party secret sharing (FSS) scheme to construct secure comparison operations. FSS can split a private function $f(x)$ into succinct function keys such that every key does not reveal private information about $f(x)$. If each key is evaluated at a specific value x, each party P_b ($b \in \{0, 1\}$) can produce a secret share $\langle f_b(x) \rangle_b$ corresponding to the private function $f(x)$. The two-party FSS-based scheme is formally described as follows.

Definition 1 (Syntax of FSS[7]). *A function secret sharing (FSS) scheme consists of two algorithms, that is,* ***FSS.Gen*** *and* ***FSS.Eval****, with the following syntax:*

- $(\mathcal{K}_0, \mathcal{K}_1) \leftarrow$ ***FSS.Gen***$(1^\lambda, f)$*: Given a function description of* f *and a security parameter* 1^λ*, outputs two FSS secret keys* $\mathcal{K}_0, \mathcal{K}_1$.
- $\langle f(x) \rangle_b \leftarrow$ ***FSS.Eval***$(\mathcal{K}_b, x)$*: Given an FSS key* $\mathcal{K}_b$ *and a public input* x*, outputs a secret share* $\langle f(x) \rangle_b$ *of the function* $f(x)$ *such that* $f(x) = \langle f(x) \rangle_0 + \langle f(x) \rangle_1$.

where correctness and security of FSS have been proved in studies proposed by Boyle[6–8].

Definition 2 (Distributed Comparison Function [7]). *Distributed comparison function (DCF) is a specific instance of FSS designed to compute comparison functions. It is built on a special interval function* $f_\alpha^<$ *that can output* 1 *if* $x < \alpha$ *and* 0 *otherwise, where* α *is random value from* $\mathbb{Z}_{2^\ell}$*. In the key generation algorithm of DCF,* ***FSS.Gen*** $(1^\lambda, f_\alpha^<)$ *generates a key* $\mathcal{K}_b$ *on the interval function* $f_\alpha^<$ *for each party*

P_b ($b \in \{0,1\}$). In evaluation of the key, each party P_b ($b \in \{0,1\}$) locally runs ***FSS.Eval ($\mathcal{K}_b, x$)*** *algorithm. This algorithm takes as input the key $\mathcal{K}_b$ and a public value x, and outputs a secret share y_b of $x < \alpha$ such that $y_0 + y_1 = 1$ if $x < \alpha$, otherwise, $y_0 + y_1 = 0$. For clarity, we abbreviate the generation and evaluation algorithms of DCF as $\mathcal{K}_b = DCF.Gen(\alpha)$ and $\langle y \rangle_b = DCF.Eval(\mathcal{K}_b, x)$, respectively.*

3.3 Threat Model

Similar to previous privacy-preserving GBDT works[1, 12, 19, 25–27], Guard-GBDT employs a two-party secure computation in the preprocessing model that has gained significant attention in secure machine learning. That is, two parties, P_0 and P_1, with inputs x_0 and x_1, aim to compute a public function $y = f(x_0, x_1)$ without revealing any additional information beyond the output y. The randomness independent of the inputs x_0 and x_1 can be generated offline in several ways–through a secure third party (STP), generic 2PC protocols, or specialized 2PC protocols. In this work, we employ the first method. Our proposed protocols follow a standard simulation paradigm against a static and semi-honest probabilistic polynomial-time (PPT) adversary that corrupts one of the two parties, as described in Definition 3.

Definition 3 (Semi-honest Security [9, 29]). *We assume that Π is a two-party protocol for computing a function $f : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}^* \times \{0,1\}^*$, where $f = (f_0, f_1)$. For input pair $(x, y) \in \{0,1\}^n$, the output pair is $(f_0(x,y), f_1(x,y))$. The view of P_b during an execution of Π on (x,y) is denoted by $view_b^{\Pi}(x,y) = (x, r_b, m_1, \ldots, m_t)$, where r_b represents the internal randomness of P_b and m_i represents the i-th message passed between the parties. The output of P_b during an execution of Π on (x,y) is denoted by O_b^{Π}. Let the joint output of two parties be $O^{\Pi} = (O_0^{\Pi}, O_1^{\Pi})$. We say that Π privately computes $f(x,y)$ if there exist PPT simulators $\mathcal{S}_0$ and $\mathcal{S}_1$ such that:*

$$\{\mathcal{S}_0(x, f_0(x,y)), f(x,y)\} \overset{c}{\equiv} \{view_0^{\Pi}(x,y), O^{\Pi}(x,y)\} \quad (1)$$

$$\{\mathcal{S}_1(y, f_1(x,y)), f(x,y)\} \overset{c}{\equiv} \{view_1^{\Pi}(x,y), O^{\Pi}(x,y)\} \quad (2)$$

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

3.4 Gradient Boosting Decision Tree

GBDT is a boosting-based machine learning algorithm that ensembles a sequence of decision trees in an additive manner [23]. In the model, each decision tree consists of internal nodes, edges, and leaf nodes: each internal node represents a test (e.g., *Age* < 20) between a feature and a threshold, each edge represents the outcome of the test, and each leaf node represents a prediction weight. In this paper, we assume that all trees are complete binary trees for maximum computational overhead. This means that each tree has $2^D - 1$ internal nodes and 2^D leaf nodes when the tree depth is denoted as D. Given a sample set $\mathbf{X} \in \mathbb{R}^{N \times F}$ with N samples and F features, each sample $\mathbf{X}[i]$ would be classified into one leaf node of each tree, then the GBDT model sums the prediction weights of all trees as the final prediction $\hat{\mathbf{y}}[i] = \sum_t^T \mathcal{T}_t(\mathbf{X}[i])$, where T is the number of decision trees and $\mathcal{T}_t(\mathbf{X}[i])$ is the prediction weight, denoted as w_t, of the t-th tree.

The GBDT training algorithm's main task is to determine the *best-split candidate* (i.e., a pair of the feature and threshold) for each tree node by evaluating all possible thresholds for each feature. However, testing all possible thresholds is computationally intractable in practice. Existing GBDT approaches[11, 38] accelerate the training process by discretizing numeric features. For example, the feature *Age* $\in [0, 100]$ can be discretized into three buckets, such as $[0, 20]$, $[20, 60]$, and $[60, 100]$. During training, the algorithm evaluates each bucket boundary (e.g., 20 and 60) sequentially as a potential split candidate. To simplify the presentation, we assume that each feature is discretized into B buckets, resulting in $F \cdot (B-1)$ split candidates for all features. We use $\mathbf{Bucket} \in \mathbb{R}^{F \times (B-1)}$ to represents all potential split candidates, where $\mathbf{Bucket}[z, u]$ be the u-th threshold of the z-th feature.

Algorithm 1 Training algorithm of GBDT

```
Input: Training samples $\mathbf{X} = \{\mathbf{x}_0, \ldots, \mathbf{x}_{N-1}\}$; Label set of samples $\mathbf{y} = \{y_0, \ldots, y_{N-1}\}$; Maximum tree depth $D$
Output: A GBDT model $\mathcal{M}^{(T)} = \{\mathcal{T}_0, \ldots, \mathcal{T}_{T-1}\}$
Function: GBDT.TraingModel($\mathbf{X}, \mathbf{y}, D$):
Bucket ← GBDT.Distinct($\mathbf{X}$), $\mathbf{y}^{(0)} = \mathbf{0}$
for $t \in [1, T-1]$ do
    if t=1 then
        $\hat{\mathbf{y}}^{(0)} = \mathbf{0}$
    else
        $\hat{\mathbf{y}}^{(t-1)} = \sum_t^{T-1} \mathcal{T}_t(\mathbf{X})$
    end if
    $\mathbf{p} = \mathrm{sigmoid}(\hat{\mathbf{y}}^{(t-1)})$
    $\mathbf{g} = \mathbf{y} - \mathbf{p}; \mathbf{h} = \mathbf{p} \cdot (1 - \mathbf{p})$
    GBDT.BulidTree($\mathcal{T}_t.root, \mathbf{X}, \mathbf{g}, \mathbf{h}, \mathbf{Bucket}, D, 0$)
    $\mathcal{M}$.push($\mathcal{T}_t$)
end for
return $\mathcal{M}$
Function: GBDT.BulidTree($\mathcal{T}.root, \mathbf{X}, \mathbf{g}, \mathbf{h}, \mathbf{Bucket}, D, d$):
if current deepth $d < D$ then
    $(z_*, u_*) \leftarrow$ GBDT.BestSplit($\mathbf{X}$)
    $\mathcal{T}.root.value \leftarrow (z_*, u_*)$
    $\mathbf{X}_L \leftarrow \{\mathbf{X}[i] \mid \mathbf{X}[i, z_*] < \mathbf{Bucket}[z_*, u_*]\}$;
    $\mathbf{X}_R \leftarrow \mathbf{X} - \mathbf{X}_L$
    GBDT.BulidTree($\mathcal{T}.root.left\_child, \mathbf{X}_L, \mathbf{g}, \mathbf{h}, \mathbf{Bucket}, D, d+1$)
    GBDT.BulidTree($\mathcal{T}.root.right\_child, \mathbf{X}_R, \mathbf{g}, \mathbf{h}, \mathbf{Bucket}, D, d+1$)
else
    $G_X = \sum_{i \in \mathbf{X}} \mathbf{g}[i]; H_X = \sum_{i \in \mathbf{X}} \mathbf{h}[i]$
    $\mathcal{T}.leaf.value \leftarrow w = -G_X / H_X$
end if
Function: GBDT.BestSplit($\mathbf{X}$) :
for $z \in [1, F]$ do
    for $u \in [1, B-1]$ do
        $\mathbf{X}_L \leftarrow \{\mathbf{X}[i] \mid \mathbf{X}[i, z] < \mathbf{Bucket}[z, u]\}$;
        $\mathbf{X}_R \leftarrow \mathbf{X} - \mathbf{X}_L$;
        $G_X = \sum_{i \in \mathbf{X}} \mathbf{g}[i]; H_X = \sum_{i \in \mathbf{X}} \mathbf{h}[i]$
        $G_L = \sum_{i \in \mathbf{X}_L} \mathbf{g}[i]; H_L = \sum_{i \in \mathbf{X}_L} \mathbf{h}[i]$
        $G_R = G_X - G_L; H_R = H_X - H_R$
        $\mathcal{G}^{(u,z)} = \frac{1}{2}(\frac{(G_L)^2}{H_L+\gamma} + \frac{(G_R)^2}{H_R+\gamma} - \frac{(G_X)^2}{H_X+\gamma})$
    end for
end for
$(z_*, u_*) \leftarrow Argmax(\{\mathcal{G}^{(1,1)}, \ldots, \mathcal{G}^{(F,B-1)}\})$
return $(z_*, u_*)$
```

Algorithm 1 describes the GBDT training process. The trees are trained in sequence, where the t-th tree is trained based on the predicates of the previous $(t-1)$ trees, and each tree is trained with the full dataset $\mathbf{X}$. Assume now we are going to train the t-th tree. Given $\mathbf{X}$, we first input them into the first $t-1$ trees to get the predicates for the N data samples, denoted as $\hat{\mathbf{y}}^{(t-1)}$. Second, we calculate the first-order gradient $\mathbf{g}$ and second-order gradient $\mathbf{h}$ of all samples as follows:

$$\mathbf{g} = \partial_{\hat{\mathbf{y}}^{(t-1)}} \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}^{(t-1)}); \mathbf{h} = \partial^2_{\hat{\mathbf{y}}^{(t-1)}} \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}^{(t-1)}) \tag{3}$$

where $\mathcal{L}(\cdot)$ is a task loss function that measures the difference between $\hat{\mathbf{y}}^{(t-1)}$ and the ground label $\mathbf{y}$. Without loss of generality, we focus on the binary classification tasks that use cross-entropy loss function, which are more complex than regression tasks[2]. In this task, the GBDT model's prediction is represented as:

$$\mathbf{p} = \text{sigmoid}(\hat{\mathbf{y}}^{(t-1)}) = \frac{1}{1 + \exp(-\hat{\mathbf{y}}^{(t-1)})} \tag{4}$$

Thus, we have the first-order gradient $\mathbf{g} = \mathbf{y} - \mathbf{p}$ and the second-order gradient $\mathbf{h} = \mathbf{p} \cdot (1 - \mathbf{p})$.

The t-th tree is trained top-down in a recursive fashion. The main task of training is to assign a feature-threshold pair to each internal node based on a gradients-guided criterion, which determines how well the pair classifies the current samples, and assign a value to each leaf. Starting with the root node, to evaluate a split candidate $\mathbf{Bucket}[z, u]$, we first partition the data samples into subset $\mathbf{X}_L = \{\mathbf{X}[i] \mid \mathbf{X}[i, z] \leq \mathbf{Bucket}[\text{z,u}]\}$ and $\mathbf{X}_R = \mathbf{X} - \mathbf{X}_L$, where $\mathbf{X}[i, z]$ represents the z-th feature of the i-th sample. Next, we aggregate the gradient statistics of parent and child nodes as follows:

$$\begin{aligned} G_X &= \sum_{i \in \mathbf{X}} \mathbf{g}[i]; \qquad H_X = \sum_{i \in \mathbf{X}} \mathbf{h}[i] \\ G_L &= \sum_{i \in \mathbf{X}_L} \mathbf{g}[i]; \qquad H_L = \sum_{i \in \mathbf{X}_L} \mathbf{h}[i] \\ G_R &= \sum_{i \in \mathbf{X}_R} \mathbf{g}[i]; \qquad H_X = \sum_{i \in \mathbf{X}_R} \mathbf{h}[i] \end{aligned} \tag{5}$$

Finally, we can evaluate a gain $\mathcal{G}$ for the split candidate $\mathbf{Bucket}[z, u]$ as follows:

$$\mathcal{G} = \frac{1}{2}(\frac{(G_L)^2}{H_L + \gamma} + \frac{(G_R)^2}{H_R + \gamma} - \frac{(G_X)^2}{H_X + \gamma}) \tag{6}$$

where $\gamma > 0$ is a (public) constant denoted as regularization parameters. The GBDT algorithm iterates all (z, u) pairs to find the split candidate that gives the maximum gain. Once the best feature and threshold are determined, the samples will be split into two partitions accordingly and used to train the nodes in the next level, and so forth. When a GBDT tree $\mathcal{T}_t$ stops growing, the prediction weight w for a leaf node can be computed as:

$$w = -\frac{G_{Leaf}}{H_{Leaf} + \gamma} \tag{7}$$

where G_{Leaf} and H_{Leaf} denotes the gradient statistics of the leaf.

4 GUARD-GBDT CONSTRUCTION

This work focuses on training a GBDT model securely between two untrusted parties who share the dataset vertically. Specifically, a training dataset $\mathbf{X} = \mathbf{X}_0 \| \mathbf{X}_1 \in \mathbb{R}^{N \times F}$ is vertically partitioned, and the party P_b ($b \in \{0, 1\}$) locally holds a dataset $\mathbf{X}_b \in \mathbb{R}^{N \times F_b}$ with N samples and F_b features. For simplicity, we assume that the features held by P_0 come before those held by P_1, and P_1 holds whole label vector $\mathbf{y}$.

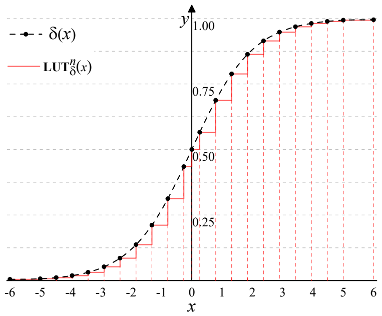


Figure 2: Our *sigmoid* function with multiple segments

For the parties who do not trust each other, all steps of GBDT training should be performed secretly without leaking anything to the other party. Basically, after training, each party P_b can only know the tree parts that involve the features owned by P_b. Any data owned or generated by one party should be hidden from the other. For instance, how the data samples are partitioned by the node owned by P_b should be unknown to P_{1-b}. MPC can be employed to achieve security goals, where the data that needs to be shared between the two parties is protected with secret sharing techniques. However, it raises data inflation and is inefficient in processing non-linear functions. Guard-GBDT is an MPC-friendly and communication-efficient GBDT framework that address the two issues.

4.1 Overview of Guard-GBDT

To streamline the use of non-linear functions, inspired by gradient quantization techniques, we introduce a segmented division-free approximation for the sigmoid function and leaf weights. Each segment's approximation is stored in a lookup table, which can be processed offline. This design enables online execution of our approximation with minimal communication, requiring only a single communication round and $n\ell$ bits of overhead, where n is a constant that represents the number of segments. Note that the overhead of previous works[19, 27, 32] for the sigmoid function is $O(\ell^2 + \ell)$. Besides, we also propose a division-free split gain to determine the best-split candidate for each tree node. Compared with Eq. 6, our gain requires only 5 communication round and 9ℓ bits of overhead,

To address data inflation, Guard-GBDT introduces compact bit-width random values to protect only the significant data bits, eliminating unnecessary data transmission. This design strategy achieves efficient message compression, minimizing bandwidth usage and saving $\ell - \ell_f - 2$ bits of communication cost for each gradient element during aggregation.

4.2 Division-free Sigmoid Function

Guard-GBDT achieves better efficiency with minor compromises in accuracy. To make the sigmoid function division-free, our main idea is to segment the input space into several intervals, approximate the sigmoid value of inputs in the same intervals to the same value,

[2] The existing private GBDT for regression tasks usually uses MSE loss such that $\mathbf{g} = \mathbf{y}^{(\hat{t-1})} - \mathbf{y}$, $\mathbf{h}$=$\mathbf{1}$, avoiding the necessary complex division.

Algorithm 2 Secure approximated *sigmoid* protocol $\Pi_{\mathrm{LUT}^n_\delta}$

```
Input: The secret share $\langle x \rangle$.
Output: The secret share $\langle \delta^{'}(x) \rangle$
$\Pi_{\mathrm{LUT}^n_\delta}$.offline($n$,$\ell$)
1: for $i \in \{0, 1, \cdots n\}$ in parallel do
2:     Let $\mathrm{LUT}^n_\delta[i] = \delta(\omega_i)$ and $\omega_i = -5 + 10i/n$.
3:     $\alpha_i \xleftarrow{\$} \mathbb{Z}_\ell$
4:     $\mathcal{K}^i_b \leftarrow$ DCF.Gen$(\alpha_i)$
5:     $\langle \alpha_i \rangle \leftarrow$ Share$(\alpha_i)$
6:     Send $(\langle \alpha \rangle_b, \mathcal{K}^i_b)$ into the party $P_b$.
7: end for
$\Pi_{\mathrm{LUT}^n_\delta}$.online($\langle x \rangle$)
8: for $i \in \{1, \cdots n\}$ in parallel do
9:     $\delta'(\omega_i) \leftarrow \mathrm{LUT}^n_\delta[i]$
10:    $m_i = x + \alpha_i - \omega_i \leftarrow$ Open$(\langle x \rangle + \langle \alpha_i \rangle - \omega_i)$
11:    $\langle \beta_i \rangle =$ DCF.Eval$(\mathcal{K}^i_b, m_i)$ // check if $x < \omega_i$
12: end for
13: $\langle \delta^{'}(x) \rangle = \delta(\omega_0) + \sum_{i=1}^{n} \langle \beta_i \rangle (\delta'(\omega_i) - \delta'(\omega_{i-1}))$
14: return $\langle \delta'(x) \rangle$
```

i.e., the minimum, and store them in a table. During the training, the parties can get the sigmoid value directly from the table by checking which interval the input belongs to. This process just involves secure comparisons, which is significantly more efficient than other alternatives of division and exponentiation.

As shown in Fig.2, the *sigmoid* $\delta(x)$ function is a smooth, continuous, and monotonic function with outputs strictly bounded within the range $(0, 1)$. Particularly, $\delta(x)$ saturates for the inputs beyond the interval $(-5, 5)$, approaching 0 or 1. As a result, we can approximate it as follows:

$$\delta(x) \approx \delta^{'}(x) = \begin{cases} \delta(\omega_0), & \text{if } x < \omega_0 \\ \delta(\omega_0), & \text{if } x \in [\omega_0, \omega_1) \\ \cdots & \\ \delta(\omega_{n-1}), & \text{if } x \in [\omega_{n-1}, \omega_n) \\ \delta(\omega_n), & \text{if } x \geq \omega_n. \end{cases} \tag{8}$$

where $\omega_i = -5 + 10i/n$, $\delta(x) = sigmoid(x) = 1/(1 + e^{-x})$, and let $\mathbf{LUT}^n_\delta[i] = \delta(w_i)$ for $i \in \{0, 1, \cdots, n\}$.

Fig. 2 shows the absolute difference between the ground truth $\delta(x) = sigmoid(x)$ and our approximation $\delta'(x)$ using the lookup table. Although there is some deviation between our approximation and the original output of the sigmoid function, which might affect the accuracy of one single tree, it is tolerable for training the GBDT model. The quantized training for GBDT[35] has demonstrated that the model's performance can be enhanced by ensembling multiple GBDT trees with low accuracy. Our experiments in Section 5.2 prove that.

$\mathbf{LUT}^n_\delta$ is public for two parties P_0 and P_1 such that they can construct the public look-up table $\mathbf{LUT}^n_\delta$ locally in offline phase. In the online phase, they can merely perform secure table queries and output the corresponding results. Algorithm 2 shows the detail of our secure approximated *sigmoid* protocol denoted as $\Pi_{\mathbf{LUT}^n_\delta}$. The core of this protocol is to privately select an activated segment of Eq. 8. Thus, we use the DCF-based less-than protocol in Section 3.2 to check if $\beta_i = x < \omega_i$ in every segment (ref. Line 8-11 in Algorithm 2). Specifically, we assume that the i-th segment is activated and the j-th segment is not activated. After checking all segment, we have $\beta_j = 1$ for all $j < i$ and $\beta_j = 0$ for all $j > i$. To obtain $\delta(\omega_i)$ of the activated segment, we only need to compute $\delta'(x) = \delta(\omega_0) + \sum_{i=1}^{n} \beta_i(\mathrm{LUT}^n_\delta[i] - \mathrm{LUT}^n_\delta[i]) = \delta(\omega_0) + \sum_{i=1}^{n} \beta_i(\delta(\omega_i) - \delta(\omega_{i-1}))$ in MPC. During the online phase, only *Open* operation (ref. Line 10 in Algorithm 2) necessitates a single communication involving the transmission of ℓ bits, whereas other computations are performed locally.

4.3 Division-free Leaf Weight

The leaf weight $w_t = -G_X/H_X$ is fundamental to GBDT training but requires division. To avoid division, we also use a lookup table to approximate the leaf weight, which is similar to the approach applied to our sigmoid function. From the GBDT algorithm, we observe that the prediction $p = sigmoid(\sum_t^T w_t)$ offers a practical property: the prediction $p \approx 1$ when $\sum_t^T w_t >= 5$ and $p \approx 0$ when $\sum_t^T w_t =< -5$. The property allows each leaf weight w_t to be scaled within the range of approximately $[-5, 5]$. Thus, we employ a numerical approximation using multiple segments to approximate the leaf weight w_t, complemented by a lookup table $\mathbf{LUT}_w$ to store these approximations. Our approximated leaf weight is given as follows:

$$w \approx w^{'} = \begin{cases} -5, & \text{if } -\frac{G_X}{H_X} < \omega'_0 \\ \omega'_0, & \text{if } -\frac{G_X}{H_X} \in [\omega'_0, \omega'_1) \\ \cdots & \\ \omega'_{n-1}, & \text{if } -\frac{G_X}{H_X} \in [\omega'_{n-1}, \omega'_n) \\ \omega'_n, & \text{if } -\frac{G_X}{H_X} \geq \omega'_n \end{cases} \tag{9}$$

where $\omega'_i = -5 + 10i/n$ for $i \in \{0, 1, \cdots, n\}$ and let $\mathbf{LUT}^n_\delta[i] = \omega'_i$.

Algorithm 3 Secure approximated leaf weight protocol $\Pi_{\mathrm{LUT}^n_w}$

```
Input: The secret shares $\langle G_X \rangle$ and $\langle H_X \rangle$.
Output: The secret share $\langle w' \rangle$
$\Pi_{\mathrm{LUT}^n_w}$.offline($n$,$\ell$)
for $i \in \{0, 1, \cdots n\}$ in parallel do
    Let $\mathrm{LUT}^n_w[i] = \omega'_i = -5 + 10i/n$.
    $\alpha_i \xleftarrow{\$} \mathbb{Z}_{2^\ell}$
    $\mathcal{K}^i_b \leftarrow$ DCF.Gen$(\alpha_i)$
    $\langle \alpha_i \rangle \leftarrow$ Share$(\alpha_i)$
    Send $(\langle \alpha \rangle_b, \mathcal{K}^i_b)$ into the party $P_b$.
end for
$\Pi_{\mathrm{LUT}^n_w}$.online($\langle G_X \rangle$, $\langle H_X \rangle$)
for $i \in \{1, \cdots n\}$ in parallel do
    $\omega'_i \leftarrow \mathrm{LUT}^n_w[i]$
    $\langle x \rangle = -\langle G_I \rangle - \omega'_i \langle H_X \rangle$
    $x + \alpha_i \leftarrow$ Open$(\langle x \rangle + \langle \alpha_i \rangle)$
    $\langle \beta_i \rangle =$ DCF.Eval$(x + \alpha_i)$ // check if $-\frac{G_X}{H_X} < \omega'_i$.
end for
$\langle w' \rangle = \omega_0 + \sum_{i=1}^{n} \langle \beta_i \rangle (\omega_i - \omega_{i-1})$
return $\langle w' \rangle$
```

Algorithm 3 depicts the detail of the $\Pi_{\mathrm{LUT}^n_w}$ protocol for leaf weight. Its core consists of selecting an activated segment of Eq. (9) in a private manner. To avoid a complex division of $w = -G_X/H_X < \omega'_i$, we reformulate this computation as $-G_X - H_X\omega_i < 0$. Similar to our sigmoid protocol, n DCF-based comparison protocols are used to check whether $\beta_i = -G_X - H_X\omega_i < 0$ for every segment. After that, we only compute $w' = \omega'_0 + \sum_{i=1}^{n} \beta_i(\omega'_i - \omega'_{i-1}) = \mathrm{LUT}^n_w[0] + \sum_{i=1}^{n} \beta_i(\mathrm{LUT}^n_w[i] - \mathrm{LUT}^n_w[i-1])$ in the MPC setting.

4.4 Division-free Split Gain

The essence of the split gain is to select the best-split candidate with the maximum $\mathcal{G}$ among multiple different split candidates. Without loss of generality, we assume that the data samples $\mathbf{X}$ are partitioned into $\{L_1, R_1\}$ and $\{L_2, R_2\}$ by two different split candidates. A naive division-free solution[3] is to convert the comparison between $\mathcal{G}_1$ and $\mathcal{G}_2$ into a check that:

$$\begin{aligned}\mathcal{G}_1 > \mathcal{G}_2 \Leftrightarrow\ & H_{L_2}H_{R_2}H_{R_1}G_{L_1}^2 + H_{L_2}H_{R_2}H_{L_1}G_{R_1}^2 \\ & > H_{L_1}H_{R_1}H_{R_2}G_{L_2}^2 + H_{L_1}H_{R_1}H_{L_2}G_{R_2}^2.\end{aligned} \tag{11}$$

However, each element of the solution requires to involve 8 secure multiplication operations and the exchange of 17ℓ bits communication messages. To reduce overheads in both computation and communication, we propose a new division-free gain, which is given as follows:

$$\mathcal{G}' = \begin{cases}(H_R+\gamma)(G_L)^2 + (H_L+\gamma)(G_R)^2 & 2H_L < H_X \\ -((H_R+\gamma)(G_L)^2 + (H_L+\gamma)(G_R)^2) & 2H_L \geq H_X\end{cases} \tag{12}$$

where $\{L, R\}$ is one split schema generated by a possible split candidate on an available sample set. Compared to the original $\mathcal{G}$ of Eq. 6, our key insight involves replacing the non-linear function $1/H_{@}$ with a simpler linear function and removing the term $G_X^2/(H_X+\gamma)$, where $@ \in \{L, R\}$. This change is based on our observation that $\mathcal{G}'$ is consistent with $\mathcal{G}$ in Eq. 6, and the term $G_X^2/(H_X+\gamma)$ is found to be ineffective in comparing between split candidates in the same sample set $\mathbf{X}$, indicating these operations does not affect the choice of the best split. Compared to the conversion of Eq. 11, our split metric does not depend on multiple divisors from other gains, making it particularly well suited to train a private GBDT model on a large-scale dataset. This independence from other gains facilitates more efficient parallel processing, enhancing both the scalability and efficiency of the training process. The detailed correctness analysis of $\mathcal{G}'$ is follows as:

Correctness of $\mathcal{G}'$ in Guard-GBDT. In the general GBDT, the second-order approximation can be used to optimize the objective function $\mathcal{L}$ quickly in the training task:

$$\mathcal{L}^{(t)} \approx \quad = \sum_{j=1}^{K}\left[\left(\sum_{i\in X} g_i\right)w_j + \frac{1}{2}\left(\sum_{i\in X} h_i + \gamma\right)w_j^2\right] \tag{13}$$

where K denotes the total number of leaves in the tree, w_j is the leaf weight at the j-th leaf node, X is the set of samples in leaf node j, and γ is a constant. Assuming the tree structure is fixed, the objective function can be simplified into a quadratic function with respect to the leaf weights w_j. For each leaf node j, the optimal weight is:

$$w_j^* = -\frac{\sum_{i\in X} g_i}{\sum_{i\in X} h_i + \lambda} = -\frac{G_X}{H_X+\lambda} \tag{14}$$

Substituting the weights, the minimum of $\mathcal{L}^{(t)}$ at single one leaf node (i.e., $K = 1$) is:

$$\mathcal{L}^{(t)} == -\frac{1}{2}\cdot\frac{(\sum_{i\in X} g_i)^2}{\sum_{i\in X} h_i + \gamma} = -\frac{1}{2}\cdot\frac{(G_X)^2}{H_X+\gamma} \tag{15}$$

Assume that L and R are the instance sets of the left and right nodes after the split. Let $X = L \cup R$. Before splitting, the minimum of $\mathcal{L}^{(t)}$ is as follows:

$$\mathcal{L}_X^* = -\frac{(G_X)^2}{2(H_X+\gamma)}. \tag{16}$$

After splitting, the minimum of $\mathcal{L}^{(t)}$ is as follows:

$$\mathcal{L}_L^* = -\frac{(G_L)^2}{2(H_L+\gamma)}; \mathcal{L}_R^* = -\frac{(G_R)^2}{2(H_R+\gamma)}. \tag{17}$$

Based on Eq. 16 and 17, the gain score $\mathcal{G}$ can be computed as:

$$\begin{aligned}\mathcal{G} &= \mathcal{L}_X^* - (\mathcal{L}_L^* + \mathcal{L}_R^*) \\ &= \frac{1}{2}\left(\frac{(G_L)^2}{H_L+\gamma} + \frac{(G_R)^2}{H_R+\gamma} - \frac{(G_X)^2}{H_X+\gamma}\right).\end{aligned} \tag{18}$$

When the parent node is fixed, G_X and H_X are constants such that $-\frac{(G_X)^2}{H_X+\gamma}$ also a constant since the dataset determines them and does not change with the split. Thus, we have:

$$\begin{aligned}\mathcal{G} = \mathcal{L}_X^* - (\mathcal{L}_L^* + \mathcal{L}_R^*) &\equiv \frac{(G_L)^2}{H_L+\gamma} + \frac{(G_R)^2}{H_R+\gamma} \\ &= [(H_R+\lambda)(G_L)^2 + (H_L+\lambda)(G_L)^2]\cdot\frac{1}{(H_L+\gamma)(H_R+\gamma)}\end{aligned} \tag{19}$$

Given $H_X = H_L + H_R$ at the fixed parent node, the derivative of $f(H_L, H_R) = 1/((H_L+\gamma)(H_R+\gamma))$ is:

$$\frac{\partial f(H_L, H_R)}{\partial H_L} = \frac{H_X - 2H_L}{((H_L+\gamma)(H_R+\gamma))^2} \tag{20}$$

Following on Eq. 20, the function $f(H_L, H_R)$ is monotonically increasing in H_L when $2H_L < H_X$ and monotonically decreasing when $2H_L \geq H_X$. This implies that it is strictly monotonic in each segmented range. Based on this, we have:

$$\mathcal{G} \propto \mathcal{G}' = \begin{cases} G^* & 2H_L < H_X \\ -G^* & 2H_L \geq H_X\end{cases} \tag{21}$$

where $\mathcal{G}^* = (H_R+\lambda)(G_L)^2 + (H_L+\lambda)(G_L)^2$ and $\propto$ denotes that $\mathcal{G}$ is proportional to $\mathcal{G}'$, meaning that the ordering among candidates remains unchanged. Thus, $\mathcal{G}'$ is equivalent to $\mathcal{G}$.

4.5 Communication-efficient Aggregation Protocol

The gain for each pair is computed using the aggregated gradients of the data samples assigned to the node. In MPC-based solutions, to prevent data exposure, each party employs a secret indicator vector $\mathbf{s} \in \{0,1\}^N$ to denote sample assignments, where $\mathbf{s}[i] = 1$ if the i-th sample belongs to the current tree node and $\mathbf{s}[i] = 0$ otherwise. Thus, the aggregation of the gradient is the computation of $G = \mathbf{s}\cdot\mathbf{g}$, followed by a local addition of the vector entries of the results. However, this process incurs data inflation after encryption and increases communication overhead. To minimize communication overhead, we propose a secure aggregation protocol Π_{Agg}, where the protocol inputs boolean secret shares $\langle\mathbf{s}\rangle = \{\langle s_0\rangle, \ldots, \langle s_{N-1}\rangle\}$, ℓ-bit secret shares $\langle\mathbf{g}\rangle = \{\langle g_0\rangle, \ldots, \langle g_{N-1}\rangle\}$ and outputs a ℓ-bit

[3] Following the equation 6 and the principles of inequality transformation, we have:

$$\begin{aligned}\mathcal{G}_1 > \mathcal{G}_2 \Leftrightarrow\ & \frac{(G_{L_1})^2}{H_{L_1}} + \frac{(G_{R_1})^2}{H_{R_1}} > \frac{(G_{L_2})^2}{H_{L_2}} + \frac{(G_{R_2})^2}{H_{R_2}} \\ \Leftrightarrow\ & H_{L_2}H_{R_2}(H_{R_1}(G_{L_1})^2 + H_{L_1}(G_{R_1})^2) \\ & > H_{L_1}H_{R_1}(H_{R_2}G_{L_2})^2 + H_{L_2}(G_{R_2})^2).\end{aligned} \tag{10}$$

secret shares $\langle G\rangle = \sum_{i=0}^{N-1}\langle s_i \cdot g_i\rangle$. Algorithm 4 depicts the details of Π_{Agg} protocol.

Recall that in a 2PC multiplication protocol of $\langle s_i \cdot g_i\rangle$, three secret shared values $\langle r\rangle, \langle r_{g_i}\rangle, \langle r_{s_i}\rangle$ are pre-generated to achieve an efficient online computation in the offline phase, where $r = r_{s_i} \cdot r_{g_i}$. During the online phase, each party P_b ($b \in \{0, 1\}$) computes $\langle \tilde{s}_i\rangle_b = \langle s_i\rangle_b + \langle r_{s_i}\rangle_b$ and $\langle \tilde{g}_i\rangle_b = \langle g_i\rangle_b + \langle r_{g_i}\rangle_b$. In this way, s_i and g_i are securely masked by random values, allowing the masked $\tilde{s}_i = s_i + r_{s_i}$ and $\tilde{g}_i = g_i + r_{g_i}$ to be reconstructed without leakage by exchanging their ℓ-bit $\langle \tilde{s}_i\rangle$ and $\langle \tilde{g}_i\rangle$. After that, P_b finishes the computation by locally calculating $\langle g_i \cdot s_i\rangle_b = b \cdot \tilde{s}_i \cdot \tilde{g}_i - \tilde{s}_i \cdot \langle g_i\rangle_b - \tilde{g}_i\langle s_i\rangle \cdot \langle g_i\rangle_b + \langle r\rangle_b$. However, using ℓ-bit r_{g_i} to mask g_i and ℓ-bit r_{s_i} to mask s_i occur data inflation due to $g_i \in [-2^{\ell_f}, 2^{\ell_f} - 1]$ and $s_i \in \{0, 1\}$, which increases communication costs. To reduce communication overhead during the 2PC multiplication protocol, our main idea compresses intermediate $\langle \tilde{s}_i\rangle$ and $\langle \tilde{g}_i\rangle$ in a compact bit-width ring, while ensuring the output of $s_i \cdot g_i$ remains on the ℓ-bit ring $\mathbb{Z}_{2^\ell}$.

Algorithm 4 Secure efficient aggregation protocol Π_{Agg}

```
Input: The secret shares $\langle \mathbf{s}\rangle = \{\langle s_0\rangle, \ldots, \langle s_{N-1}\rangle\}$ and $\langle \mathbf{g}\rangle = \{\langle g_0\rangle \ldots \langle g_{N-1}\rangle\}$.
Output: The secret share $\langle G\rangle = \sum_{i=0}^{N-1}\langle s_i \cdot g_i\rangle$
$\Pi_{\text{Agg}}$.offline($\ell$, $\ell_f$)
 1: for $i \in [0, N-1]$ in parallel do
 2:     $r_{s_i} \xleftarrow{\$} \mathbb{Z}_2, r_{g_i} \xleftarrow{\$} \mathbb{Z}_{2^{\ell'}}$, where $\ell' = \ell_f + 2$.
 3:     $u_i = r_{s_i} \cdot r_{g_i}, v_i = r_{g_i} \gg (\ell' - 1)$, and $m_i = r_{s_i} \cdot v_i$
 4:     $\langle r_{s_i}\rangle, \langle r_{g_i}\rangle, \langle u_i\rangle, \langle v_i\rangle, \langle m_i\rangle \leftarrow \text{Share}(r_{s_i}, r_{g_i}, u_i, v_i, m_i)$
 5:     $\mathcal{K}_b = \langle r_{s_i}\rangle_b \| \langle r_{g_i}\rangle_b \| \langle u_i\rangle_b \| \langle v_i\rangle_b \| \langle m_i\rangle_b$.
 6:     Send $\mathcal{K}_b$ to the party $P_b$.
 7: end for
$\Pi_{\text{Agg}}$.online($\langle \mathbf{s}\rangle$,$\langle \mathbf{g}\rangle$)
 8: for $i \in [0, N-1]$ in parallel do
 9:     $\langle r_{s_i}\rangle_b, \langle r_{g_i}\rangle_b, \langle u_i\rangle_b, \langle v_i\rangle_b, \langle m_i\rangle_b \leftarrow \mathcal{K}_b$
10:     $[[s_i]] = \langle s_i\rangle \mod 2$
11:     $\langle \hat{g}'_i\rangle^{\ell'} = \langle g_i\rangle + \langle r_{g_i}\rangle \mod 2^{\ell'}$
12:     $[[\hat{s}_i]] = [[s_i]] + \langle r_{s_i}\rangle \mod 2$.
13:     $P_b$ sends $\ell'$-bit $\langle \hat{g}'_i\rangle_b^{\ell'}$ and 1-bit $[[\hat{s}_i]]_b$ to $P_{1-b}$.
14:     $P_b$ receives $\langle \hat{g}'_i\rangle_{1-b}^{\ell'}$ and $[[\hat{s}]]_{1-b}$ from $P_{1-b}$.
15:     $\hat{g}'_i = \langle \hat{g}'_i\rangle_b^{\ell'} + \langle \hat{g}'_i\rangle_{1-b}^{\ell'} + 2^{\ell'} \mod 2^{\ell'+1}$ and $\hat{s}_i = [[\hat{s}]]_b \oplus [[\hat{s}]]_{1-b}$.
16:     $v_{g_i} = \neg\hat{g}'_i \gg (\ell' - 1)$
17:     $\langle s_i \cdot g_i\rangle = \hat{s}_i\hat{g}'_i + \hat{s}v_{g_i}\langle v_i\rangle \cdot 2^{\ell'} + \hat{s}\langle r_{g_i}\rangle - \hat{s} \cdot 2^{\ell'} + (1 - 2\hat{s}_i)\hat{g}'_i\langle r_{s_i}\rangle + (1 - 2\hat{s}_i)v_{g_i}\langle m_i\rangle \cdot 2^{\ell'} + (1 - 2\hat{s}_i)(\langle u_i\rangle - \langle r_{s_i}\rangle \cdot 2^{\ell'})$
18: end for
19: return $\langle G\rangle = \sum_{i=0}^{N-1}\langle s_i \cdot g_i\rangle$.
```

First, we explain how to compress intermediate messages $\tilde{s}_i$ and $\tilde{g}_i$ (ref. Lines 9-14 in Algorithm 4). In our protocol, the arithmetic secret-shared $\langle s_i\rangle$ over the ring $\mathbb{Z}_{2^\ell}$ is scaled into the boolean secret-shared $[[s_i]]$ through a local modulo 2 operation, i.e., $[[s_i]] = \langle s_i\rangle \mod 2$. Then, each party computes $[[\hat{s}_i]] = [[s_i]] + \langle r_{s_i}\rangle \mod 2$ such that we can reconstruct a 1-bit compressed $\hat{s}_i = s_i \oplus r_{s_i} \mod 2$ by exchanging the boolen secret-shared $[[\hat{s}_i]]$ without leakage, where $r_{s_i} \in \{0, 1\}$ is generated in the offline and is secret-shared between P_0 and P_1. For the arithmetic secret-shares of $g_i \in [-2^{\ell_f}, 2^{\ell_f} - 1]$, only the sign and fractional parts of g_i need to be masked with an $(\ell_f + 1)$-bit random value $r_{g_i} \in [-2^{\ell_f}, 2^{\ell_f} - 1]$. This allows us to scale $\langle \hat{g}_i\rangle$ into the ℓ'-bit secret share $\langle \hat{g}_i\rangle^{\ell'}$ through a local modulo $2^{\ell'}$ operation, i.e., $\langle \hat{g}_i\rangle^{\ell'} = \langle \hat{g}_i\rangle + \langle r_{g_i}\rangle \mod 2^{\ell'}$, where ℓ' is set as $\ell_f + 2$ since $g_i + r_i \in [-2^{\ell_f+1}, 2^{\ell_f+1} - 2]$. Thus, we can reconstruct an ℓ'-bit compressed $g_i^{\ell'} = g_i + r_i \mod 2^{\ell'}$ by exchanging the secret-shared $\langle \hat{g}_i\rangle^{\ell'}$.

What remains is to explain how to ensure the result of $s_i \cdot g_i$ still is secret-shared on the ℓ-bit ring $\mathbb{Z}_{2^\ell}$ (ref. Lines 15-17 in Algorithm 4). Since the compressed $\hat{s}_i$ and $\hat{g}_i$ are encoded using different bit widths, they cannot be used directly for secure computation of $s_i \cdot g_i$ over the ring $\mathbb{Z}_{2^\ell}$. Thus, we need the upcast conversion from a different small bit-width fixed-point representation to a uniform ℓ-bit fixed-point representation. Fortunately, there is a useful relationship between $\hat{s}_i$ and s_i over the ring $\mathbb{Z}_{2^\ell}$, and similarly for g_i and $\hat{g}_i$. That is:

$$\begin{aligned} s_i \bmod 2^\ell &= (\hat{s}_i + r_{s_i} - 2r_{s_i}\hat{s}_i) \bmod 2^\ell \\ g_i \bmod 2^\ell &= (\hat{g}_i + w \cdot 2^{\ell'} - r_{g_i} - 2^{\ell'})) \bmod 2^\ell \end{aligned} \tag{22}$$

where $w = (g_i + r_{g_i}) \overset{?}{>} 2^{\ell'}$. However, we can not directly compute w since it involves non-linear comparisons that are more expensive than linear operations in MPC. To further facilitate the computation, we employ a positive heuristic trick used in Ditto[39]. That is, we add a large bias $2^{\ell'-1}$ to g_i to ensure that $g'_i = g_i + 2^{\ell_f} \in [0, 2^{\ell'-1}]$ is positive over the ring $\mathbb{Z}_{2^{\ell'}}$. The caculation $w = (g_i + r_{g_i}) \overset{?}{>} 2^{\ell'}$ then can be converted into $w = (r_{g_i} \gg (\ell' - 1)) \cdot \neg((g'_i + r_{g_i}) \gg (\ell' - 1))$, where $\gg$ denotes a right-shift operation, and $\neg$ denotes a negation operation. After the upcast conversion, the bias can be directly subtracted to eliminate its influence. As a result, the secure computation of $s_i \cdot g_i$ over the $\mathbb{Z}_{2^\ell}$ becomes:

$$\begin{aligned} & s_i \cdot g_i \bmod 2^\ell \\ & = ((\hat{s}_i - r_{s_i} + 2\hat{s}_i r_s) \cdot (\hat{g}'_i + w \cdot 2^{\ell'} - r_{g_i} - 2^{\ell'})) \bmod 2^\ell \\ & = \hat{s}_i\hat{g}'_i + \hat{s}_i w \cdot 2^{\ell'} - \hat{s}_i(r_{g_i} - 2^{\ell'}) + (1 - 2\hat{s}_i)\hat{g}'_i r_{s_i} \\ & + (1 - 2\hat{s}_i)r_{s_i} w \cdot 2^{\ell'} + (1 - 2\hat{s}_i)(r_{s_i}r_{g_i} - 2^{\ell'} r_{s_i}) \\ & = \textcolor{red}{\hat{s}_i\hat{g}'_i} + \textcolor{red}{\hat{s}_i v_{g_i}}\textcolor{blue}{v_i} \cdot 2^{\ell'} + \textcolor{red}{\hat{s}_i}\textcolor{blue}{r_{g_i}} - \textcolor{red}{\hat{s}_i \cdot 2^{\ell'}} + \textcolor{red}{(1 - 2\hat{s}_i)\hat{g}'_i}\textcolor{blue}{r_{s_i}} \\ & + \textcolor{red}{(1 - 2\hat{s}_i)v_{g_i} \cdot 2^{\ell'}}\textcolor{blue}{m_i} + \textcolor{red}{(1 - 2\hat{s}_i)}\textcolor{blue}{(u_i - r_{s_i} \cdot 2^{\ell'})} \end{aligned} \tag{23}$$

where $u_i = r_{s_i} \cdot r_{g_i}$, $v_{r_{g_i}} = r_{g_i} \gg (\ell' - 1)$, $m_i = r_{s_i} \cdot (r_{g_i} \gg (\ell' - 1))$ and $v_{g_i} = \neg\hat{g}'_i \gg (\ell' - 1)$. In the equation, we use different colors to denote the nature of variables: **red** for input-dependent variables computed locally online and **blue** for input-independent random variables computed offline. These blue random variables are used to generate a pair of keys, $\mathcal{K}_0$ and $\mathcal{K}_1$ (ref. Lines 2-5 in the Algorithm 4) on offline phase. Each key $\mathcal{K}_b$ is then securely sent to the corresponding party. This pre-computation step ensures that when the online phase begins, the protocol can efficiently compute the necessary values with minimal communication overhead.

4.6 Putting Everything Together

Here we introduce how Guard-GBDT combines the above components to train a tree in the GBDT model over the vertical datasets $\mathbf{X}_0$ and $\mathbf{X}_1$. Guard-GBDT first invokes the offline program of our components to preprocess all randomness independent of the online inputs, and then trains a GBDT model, which is given in Algorithm 5. Each party P_b inputs its dataset $\mathbf{X}_b \in \mathbb{R}^{N \times F_0}$, secret-shared sample space $\langle \mathbf{s}\rangle$, the secret-shared first-order gradients $\langle \mathbf{g}\rangle$, the secret-shared second first-order gradients $\langle \mathbf{h}\rangle$ and outputs a distributed GBDT tree $\mathcal{T}_{t,b}$. We assume that the first-order and

Algorithm 5 Training a GDBT tree in Guard-GBDT

```
Input: The vertical dataset $\mathbf{X}_b \in \mathbb{R}^{N\times F_0}$; The sample space $\langle \mathbf{s}\rangle$; The first-order gradients $\langle \mathbf{g}\rangle$; The second-order gradients $\langle \mathbf{h}\rangle$; The current node $node$ of $\mathcal{T}_{t,b}$; The tree depth $d = 0$ of tree.
Output: A distributed GBDT tree $\mathcal{T}_{t,b}$ for $P_b$
Public hyperparameters: the bucket number $B$, the maximum tree depth $D$
Function: SecureBuildTree($\mathbf{X}_b$, $[[\mathbf{s}]]$, $\langle \mathbf{g}\rangle$, $\langle \mathbf{h}\rangle$, $\mathcal{T}_{t,b}.node, d$):
1: if $d < D$ then
2:     $(\langle z_*\rangle, \langle u_*\rangle) \leftarrow$ SecureBestSplit($\mathbf{X}$, $\langle \mathbf{s}\rangle$, $\langle \mathbf{g}\rangle$, $\langle \mathbf{h}\rangle$)
3:     [Open best-split identifier:] $c = \text{Open}(\langle z_*\rangle - F_0 < 0)$; $P_c$ receives $(\langle z_*\rangle_{1-c}$, $\langle u_*\rangle_{1-c})$ from $P_{1-c}$; $P_c$ opens $z_* = \langle z_*\rangle_{1-c} + \langle z_*\rangle_c$ and $u_* = \langle u_*\rangle_c + \langle u_*\rangle_{1-c}$.
4:     [Update left and right sample space:]$P_c$ set $\langle \mathbf{s}_{test}\rangle_c = \{\mathbf{X}_c[1, z_*] < \mathbf{Bin}_c[z_*, u_*], \ldots, \mathbf{X}_c[N, z_*] < \mathbf{Bin}_c[z_*, u_*]\}$ and $\langle \mathbf{s}_{test}\rangle_{1-c} = \mathbf{0}$.
5:     $P_0$ and $P_1$ compute $\langle \mathbf{s}_{L^*}\rangle = \langle \mathbf{s}_{test}\rangle \cdot \langle \mathbf{s}\rangle$, $\langle \mathbf{s}_{R^*}\rangle = (\mathbf{1} - \langle \mathbf{s}_{test}\rangle) \cdot \langle \mathbf{s}\rangle$
6:     [Recods $z_*$ and $u_*$ into tree $\mathcal{T}_{t,b}$:] $P_c$ records $\mathcal{T}_{t,b}$.node.$value$ = $(z_*^{(k)}, u_*^{(k)})$; $P_{1-c}$ records $\mathcal{T}_{t,b}$.node.$value = (-1, -1)$
7:     [Build subtrees:] SecureBuildTree($\mathbf{X}$, $\langle \mathbf{s}_{L^*}\rangle$, $\langle \mathbf{g}\rangle$, $\langle \mathbf{h}\rangle$, $\mathcal{T}_{t,b}$.left, $d + 1$); SecureBuildTree($\mathbf{X}$, $\langle \mathbf{s}_{R^*}]\rangle$, $\langle \mathbf{g}\rangle$, $\langle \mathbf{h}\rangle$, $\mathcal{T}_{t,b}$.right, $d + 1$)
8: else
9:     [Build leaf node:] $\langle G\rangle = \Pi_{\text{Agg}}.\text{online}(\langle \mathbf{s}\rangle, \langle \mathbf{g}\rangle)$;
       $\langle H\rangle = \Pi_{\text{Agg}}.\text{online}(\langle \mathbf{s}\rangle, \langle \mathbf{h}\rangle)$; $\langle w\rangle = \Pi_{\text{LUT}_w}.\text{online}(\langle G\rangle, \langle H\rangle)$;
       $\mathcal{T}_{t,b}.node.value = (\langle w\rangle_b, -1)$;
10: end if
11: return $\mathcal{T}_{t,b}$
Function: SecureBestSplit($\mathbf{X}$, $[[\mathbf{s}]]$, $\langle \mathbf{g}\rangle$, $\langle \mathbf{h}\rangle$):
12: for $z \in \{1, \ldots, F\}$ do
13:     for $u \in \{1, B - 1\}$ in parallel do
14:         [Compute sample space:] $P_b$ computes $\langle \mathbf{s}_{test}\rangle_b = \{\mathbf{X}_b[1, z] < \mathbf{Bucket}_b[z, u], \ldots, \mathbf{X}_b[N, z] < \mathbf{Bucket}_b[z, u]\}$ and $P_{1-b}$ sets $\langle \mathbf{s}_{test}\rangle_{1-b} = \mathbf{0}$;$\langle \mathbf{s}_L\rangle = \langle \mathbf{s}\rangle \cdot \langle \mathbf{s}_{test}\rangle$ and $\langle \mathbf{s}_R\rangle = \langle \mathbf{s}_{test}\rangle \cdot (\mathbf{1} - \langle \mathbf{s}\rangle)$.
15:         [Aggregate gradients:] $\langle G_L\rangle = \Pi_{\text{Agg}}.\text{online}(\langle \mathbf{s}_L\rangle, \langle \mathbf{g}\rangle)$; $\langle G_R\rangle = \Pi_{\text{Agg}}.\text{online}(\langle \mathbf{s}_R\rangle, \langle \mathbf{g}\rangle)$; $\langle H_L\rangle = \Pi_{\text{Agg}}.\text{online}(\langle \mathbf{s}_L\rangle, \langle \mathbf{h}\rangle)$; $\langle H_R\rangle = \Pi_{\text{Agg}}.\text{online}(\langle \mathbf{s}_R\rangle, \langle \mathbf{h}\rangle)$.
16:         [Compute gain:] $\langle \mathbf{Sign}\rangle = \langle 2H_L - H_X\rangle < 0$; $\langle \mathcal{G}^{(z,u)}\rangle = \langle H_R + \gamma\rangle \cdot \langle (G_L)^2\rangle + \langle H_L + \gamma\rangle \cdot \langle (G_R)^2\rangle$; $\langle \mathcal{G}^{(z,u)}\rangle = \langle \mathbf{Sign}\rangle \cdot \langle \mathcal{G}^{(z,u)}\rangle + \langle 1 - \mathbf{Sign}\rangle \cdot \langle \mathcal{G}^{(z,u)}\rangle$;
17:     end for
18: end for
19: [The best split:] $\langle z_*\rangle, \langle u_*\rangle = \Pi_{\text{Argmax}}(\{\langle \mathcal{G}^{(1,1)}\rangle, \ldots, \langle \mathcal{G}^{(F,B-1)}\rangle\})$
20: return $(\langle z_*\rangle, \langle u_*\rangle)$
```

second-order gradients have been computed privately and given as inputs in the algorithm. This allows us to directly reuse the algorithm to build the next tree. For a binary classification task using the cross-entropy loss, our secure $\Pi_{\text{LUT}^n_\delta}$ protocol in Algorithm 2 can be used to compute $\langle \mathbf{g}\rangle = \Pi_{\text{LUT}^n_\delta}.\text{online}(\langle \hat{\mathbf{y}}^{(t-1)}\rangle) - \mathbf{y}$ and $\langle \mathbf{h}\rangle = \Pi_{\text{LUT}^n_\delta}.\text{online}(\langle \hat{\mathbf{y}}^{(t-1)}\rangle) \cdot \Pi_{\text{LUT}^n_\delta}.\text{online}(\langle \hat{\mathbf{y}}^{(t-1)}\rangle)$, where the initial prediction is $\hat{\mathbf{y}} = \langle \mathbf{0}\rangle$ for the first tree.

With the help of the gradients, the algorithm first uses a secure best-split function (Line 2 in Algorithm 5) to determine the best-split identifier (z_*, u_*). In the function, each party P_b first picks all possible thresholds $\mathbf{Bucket}_b$ of each feature locally from its own dataset $\mathbf{X}_b$. For a possible split candidate $\mathbf{Bucket}_b[z, u]$, party P_b generates a local test of sample space $\langle \mathbf{s}_{\text{test}}\rangle_b = \{\mathbf{X}[0, z] < \mathbf{Bucket}_b[z, u], \ldots, \mathbf{X}[N, z] < \mathbf{Bucket}_b[z, u]\}$, while the party P_{1-b} sets $\langle \mathbf{s}_{test}\rangle_{1-b} = \mathbf{0}$. Next, the two parties compute the corresponding possible left sample space $\langle \mathbf{s}_L\rangle = \langle \mathbf{s}_{test}\rangle \cdot \langle \mathbf{s}\rangle$ and the right sample space $\langle \mathbf{s}_R\rangle = (1 - \langle \mathbf{s}_{test}\rangle) \cdot \langle \mathbf{s}\rangle$. We use our secure Π_{Agg} protocol to aggregate the gradients on the left and right sample space like $\langle G_L\rangle$, $\langle G_R\rangle$, $\langle H_L\rangle$ and $\langle H_R\rangle$. After that, the two parties can compute the splitting scores $\{\langle \mathcal{G}^{z,u}\rangle\}$ jointly from $\{\langle G_L\rangle, \langle G_R\rangle, \langle H_L\rangle, \langle H_R\rangle\}$ according to our split metric, secure multiplication and secure DCF-based comparison. Finally, the best-split identifier $(\langle z_*\rangle, \langle u_*\rangle)$ is determined using the Π_{Argmax} protocol of SiGBDT.

Once the best-split identifier $(\langle z_*\rangle, \langle u_*\rangle)$ is computed, we can only reveal it to its holder ((Line 3 in Algorithm 5) according to the secure definition and the existing private GDBT training paradigm. To this end, we first compute a 1-bit indicator $c = \text{Open}\langle z_*\rangle < F_0$ to indicate the chosen (z_*, u_*) belongs to the P_c. We let P_{1-c} send ist share of the best-split identifier into P_c. Thus, P_c records (z_*, u_*) into the current node of $\mathcal{T}_{t,b}$ and P_{1-c} records (-1,-1) into the current node of $\mathcal{T}_{t,b}$. Next, P_c can locally test $ts = \{\mathbf{X}_c[1, z_*] < \mathbf{Bin}_c[z_*, u_*], \ldots, \mathbf{X}_c[N, z_*] < \mathbf{Bin}_c[z_*, u_*]\}$ and set $\langle \mathbf{s}_{test}\rangle_c = ts$ and $\langle \mathbf{s}_{test}\rangle_{1-c} = \mathbf{0}$. Then, the two parties can update the left sample space $\langle \mathbf{s}_{L^*}\rangle = \langle \mathbf{s}\rangle \cdot \langle \mathbf{s}_{test}\rangle$ and the right sample space $\langle \mathbf{s}_{R^*}\rangle = \langle \mathbf{s}\rangle \cdot (1 - \langle \mathbf{s}_{test}\rangle$ (Line 4 in Algorithm 5). After that, P_0 and P_1 loop through the above process to train the left and right child of the current node. When reaching the maximum depth, P_0 and P_1 aggregates gradients in sample space of the current leaf to compute leaf weight $\langle w\rangle$ using secure $\Pi_{\text{LUT}^n_w}$ protocol. Finally, the secret-shared leaf weight $\langle w\rangle$ is recorded into the current node of $\mathcal{T}_{t,b}$.

Secure prediction. Once the GBDT model is trained, secure prediction can be performed using an oblivious algorithm. Previous works, such as Pivot [40] and SecureBoost[12], have applied HE for secure GBDT prediction in the MPC model. In contrast, we adopt a similar approach to SiGBDT[27] and Squirrel [32] by employing the secret-sharing-based prediction algorithm proposed by HEP-XGB [19], which avoids the time-consuming operations associated with HE. Briefly, for a new input sample, each party can locally prepare a binary vector $\mathbf{path}_b = \mathbf{0}$ for each tree. Recall that P_b holds the tree part $\mathcal{T}_{t,b}$ which includes his features. The elements of $\mathbf{path}_b$ are updated by comparing his features and corresponding node thresholds. That is, $\mathbf{path}_b[k] = 1$ indicates that the sample might be classified to the k-th leaf (the leaves are ordered from left to right), while $\mathbf{path}_b[k] = 0$ means the input will not be classified to the k-th leaf based on the split identifiers held by P_b. At the end, there is only one entry of '1' in $\mathbf{path}_0 \cdot \mathbf{path}_1$. We define $\langle \mathbf{pth}\rangle_0 = \mathbf{path}_0$, $\langle \mathbf{pth'}\rangle_0 = \mathbf{0}$ held by P_0 and $\langle \mathbf{pth}\rangle_1 = \mathbf{0}$, $\langle \mathbf{pth'}\rangle_1 = \mathbf{path}_1$ held by P_1. The final prediction on the tree is computed as $\sum_k \langle \mathbf{pth}\rangle[k] \cdot \langle \mathbf{pth'}\rangle[k] \cdot \langle \mathbf{w}\rangle[k]$ given the shares of the leaf weights.

4.7 Security Analysis

Due to the page limit, we provide a concise security analysis. A long formal security analysis will be provided in our full version. Guard-GBDT framework consists of multiple sub-protocols for smaller private computations and is secure against semi-honest PPT adversaries as defined in Definition 3. During the training, all the data communicated between the two parties are just secret shares. Our linear operations including addition and multiplication are all performed securely in ASS and their security has been proved in the existing two-party computation framework, ABY [16]. The nonlinear comparison protocol is implemented with DCF, which does not leak either the input from the other party or the output and has been proved in the FSS framework [7]. Notably, our Π_{agg} protocol achieves the same security and accuracy as existing

MPC-based protocols. When the attacker knows the gradient lies in $[-1, 1]$, the probability of correctly guessing the private data is $\frac{1}{2^\ell} \Big/ \frac{1}{2^{\ell-\ell_f-1}} = \frac{1}{2^{\ell_f+1}}$ in existing MPC-based works. Since our protocol protects only the sign bit and fraction, the attacker's success probability is also $\frac{1}{2^{\ell_f}} \times \frac{1}{2} = \frac{1}{2^{\ell_f+1}}$. As a result, the adversary who controls one party cannot learn any data about the other party following the composition theorem [9] of the semi-honest model.

4.8 Extension of Guard-GBDT

The Guard-GBDT framework can be flexibly applied to collaborative learning scenarios involving three or more parties, such as cross-enterprise collaborative learning and federated learning [22]. For the proposed approximation protocol and division-free split gain metric, the linear operations only need to replace the secret-sharing primitives that support three or more parties. The nonlinear operations can use the existing works of Boyle [5] to replace the two-party DCF. For the proposed aggregation protocol, we use a 1-out-of-m secret sharing protocol to construct a multi-party party and use the idea in Section 4.5 to compass intermediate mask value for lower communication overhead.

5 EXPRIMENTS EVALUATION

5.1 Expriment Setup

Testbed Environment. We implement a prototype of Guard-GBDT using PyTorch framework [34] and evaluate it on a computer equipped with an AMD Ryzen9 5900X CPU @ 3.20GHz, 512GB RAM, operating Ubuntu 20. Following prior works [2, 27], we utilize a Linux network tool, "tc", to simulate the local-are network (LAN, RTT: 0.2 ms, 1 Gbps) and the wide-area network (WAN, RTT: 40ms, 100 Mbps) between the two parties on the same workstation. Similarly to SiGBDT[27] and HEP-XGB [19], all secure protocols are on the $\ell = 64$ bit length ring. We set the fraction precision $\ell_f = 16$ and the security parameter of FSS $\lambda = 128$.

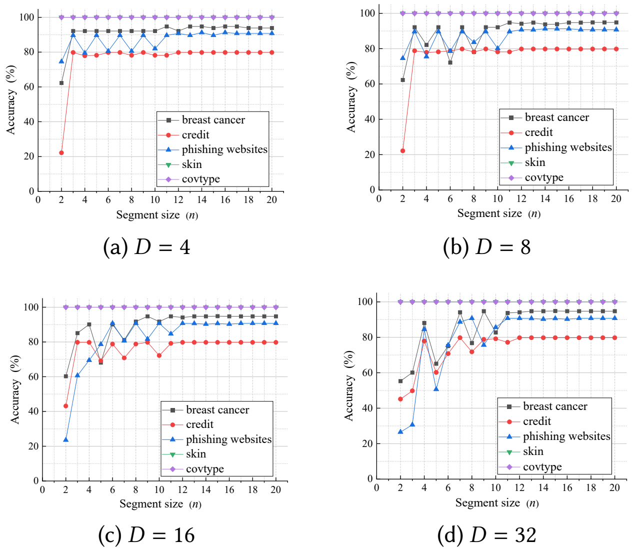
Baseline. We compare Guard-GBDT with XGboost (plaintext) model to verify the accuracy of our work. We also compare Guard-GBDT with the state-of-the-art works: SiGBDT[27] and HEP-XGB[19], and implement their works following the details provided in their paper. We do not compare with other works (such as Squirrel[32], Sgboost[42], NodeGuard[14], Privet[43], and SecureBoost[12]), as SiGBDT and HEP-XGB have already outperformed them.

Table 2: Information of training datasets

Dataset	Instances	Features
Breast Cancer	699	9
Credit	45211	16
phishing website	11055	67
Skin	245057	3
Covertype	581012	54

Dataset. Following previous works [19, 23, 27, 32], we use 5 real-world datasets from UCI Machine Learning Repository[4] to evaluate the accuracy and efficiency of Guard-GBDT. The datasets are summarized in Table 2. We follow a ratio of 8:2 for training and testing datasets for every dataset. Each training dataset is split vertically and evenly for P_0 and P_1. We assume that the samples in each party's database have been properly aligned beforehand. Similar to SiGBDT and HEP-XGB, each dataset is preprocessed to discretize features into different bins via the existing equal-width binning method in the offline phase, where the max bin size is $B = 8$.

5.2 Accuracy Evaluation



(a) $D = 4$

(b) $D = 8$

(c) $D = 16$

(d) $D = 32$

Figure 3: Accuracy with different segment approximations over different tree depths.

Guard-GBDT employs lookup tables with multiple segment approximations to replace the sigmoid function and leaf weight computation. How the data are segmented could affect the model's accuracy. We first test model accuracy with different segment sizes on five real datasets to find the best segment size of approximations, where the total number of trees in GBDT is $T = 5$. As shown in Fig. 3, the experimental results for large datasets, such as covertype and skin, demonstrate stable performance with accuracy close to 100% across various segment sizes (n) and tree depths (D), indicating that these datasets are insensitive to changes in segment size. In contrast, results on smaller datasets exhibit fluctuations and are more sensitive to segment size changes. These fluctuations became more pronounced as the tree depth increased. However, as the segment size grows, the accuracy at different depths tends to stabilize, suggesting that once the segment size exceeds a certain threshold (i.e., $n \geq 10$ for $D = 4$ or 8 and $n \geq 12$ for $D = 4$ or 8), the model's accuracy becomes consistent across different datasets.

Next, we use the plaintext XGBoost to train a model over 5 real-world datasets as a baseline. Then, we compare our Guard-GBDT with SiGBDT and HEP-XGB under varying tree depths ($D = 4, 8, 16, 32$), where the segment size $n = 12$ and tree number $T = 5$. The hyperparameters and initialization are consistent for Guard-GBDT, SiGBDT, and HEP-XGB. Results of Table 3 show that the performance of Guard-GBDT and SiGBDT is very similar under varying tree depths across multiple datasets, both are

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

Table 3: Model accuracy over different tree depth

Depth	Dataset	Guard-GBDT	SiGBDT	HEP-XGB	XGBoost (Plain)
D=4	breast-cancer	94.74	93.86	87.72	97.37
	credit	79.90	81.12	76.34	82.42
	phishing	89.42	89.19	78.11	89.91
	skin	100.00	100.00	91,11	100.00
	covertype	100.00	100.00	90.81	100.00
D=8	breast-cancer	94.78	95.86	90.72	97.37
	credit	80.70	81.12	77.14	82.37
	phishing	90.00	89.91	81.12	91.91
	skin	100.00	100.00	91,11	100.00
	covertype	100.00	100.00	90.81	100.00
D=16	breast-cancer	93.14	95.86	89.72	97.37
	credit	79.75	81.12	77.14	82.15
	phishing	90.00	89.91	84.12	92.11
	skin	100.00	100.00	91,11	100.00
	covertype	100.00	100.00	90.81	100.00
D=32	breast-cancer	92.74	92.86	89.12	97.37
	credit	80.12	81.12	77.90	82.17
	phishing	90.00	89.91	84.12	91.91
	skin	100.00	100.00	91,11	100.00
	covertype	100.00	100.00	90.81	100.00

Table 4: Model accuracy over different numbers of trees

Number of trees	Dataset	Guard-GBDT	SiGBDT	HEP-XGB	XGBoost (Plain)
T=5	breast-cancer	94.74	93.86	87.72	97.37
	credit	79.90	81.12	76.34	82.42
	phishing	89.42	89.19	78.11	89.91
	skin	100.00	100.00	91,11	100.00
	covertype	100.00	100.00	90.81	100.00
T=10	breast-cancer	94.74	96.86	90.72	97.37
	credit	78.85	82.12	77.14	82.37
	phishing	90.28	89.91	81.12	91.91
	skin	100.00	100.00	98,11	100.00
	covertype	100.00	100.00	99.81	100.00
T=15	breast-cancer	94.78	95.86	89.72	97.37
	credit	80.75	82.12	78.14	82.15
	phishing	90.73	89.91	89.12	92.11
	skin	100.00	100.00	100.00	100.00
	covertype	100.00	100.00	100.00	100.00
T=20	breast-cancer	94.90	92.86	89.12	97.37
	credit	80.80	81.12	79.90	82.17
	phishing	90.73	89.91	89.12	91.91
	skin	100.00	100.00	100.00	100.00
	covertype	100.00	100.00	100.00	100.00

comparable to the accuracy of plaintext XGBoost, indicating strong model performance. However, HEP-XGB suffers significant performance degradation, underperforming Guard-GBDT by $6 - 12\%$ across datasets (e.g., 87.72% vs. 94.74% on breast cancer at $D = 4$). The degradation stems from a numerically approximated sigmoid $sigmoid(x) \approx 0.5 + 0.5 \cdot x/(1 + |x|)$ in HEP-XGB such that it has a larger approximation error than Guard-GBDT and SiGBDT.

Finally, we evaluate the accuracy of Guard-GBDT, SiGBDT, and HEP-GBDT under varying ensemble sizes of trees ($T = 5, 10, 15, 20$) and compare them with the non-private XGBoost, as shown in Table 4. HEP-XGB exhibits a strong dependence on T, with accuracy improvements of up to 12.89% on the skin and 9.2% on phishing as T increased from 5 to 20. However, Guard-GBDT and SiGBDT have minor fluctuations ($\pm 1\% - 2\%$) and are comparable with the non-private XGBoost. They achieve perfect accuracy across total numbers of trees. This indicates that Guard-GBDT and SiGBDT are stable and insensitive to the ensemble sizes of trees.

5.3 Microbenchmarks

We evaluate component performance and compare it with SiGBDT and HEP-XGB in terms of runtime and communication costs in LAN and WAN, as shown in Table 5. SiGBDT approximates $\exp(x)$ using a Taylor series, followed by division for the sigmoid function, while HEP-XGB employs a numerical approximation $sigmoid(x) \approx 0.5 + 0.5 \cdot x/(1 + |x|)$, both using Goldschmidt's series for division. Their methods are less efficient than our lookup table-based approximation.

Table 5 also shows that our approach significantly reduces communication overhead. Similarly, our lookup table-based leaf weight computation outperforms theirs in runtime and communication, as it relies only on scalar multiplication and lightweight FSS comparisons, avoiding SiGBDT and HEP-XGB's costly division protocols. For splitting gain, we improve efficiency with a division-free method, outperforming SiGBDT and HEP-XGB. Guard-GBDT is 788× and 1000× faster in LAN and 814× and 905× faster in WAN than SiGBDT and HEP-XGB, respectively. For gradient aggregation, we optimize performance by compressing intermediate communication. In LAN, Guard-GBDT achieves 1.5× and 80.5× speedup over SiGBDT and HEP-XGB, while in WAN, it provides 2.03× and 16.48× speedup, respectively.

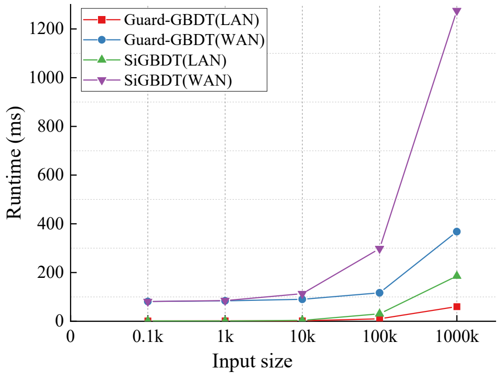


Figure 4: Gradient aggregation runtime

Table 5 also shows that the runtime for sigmoid, leaf weight, and split gain operations are no significant different between LAN and WAN settings, indicating that computational cost, rather than communication overhead, is the primary bottleneck for these non-linear MPC operations. In contrast, gradient aggregation runtime varies significantly between LAN and WAN settings. As shown in Fig. 4, its runtime increases much more in WAN than in LAN, highlighting its sensitivity to communication delays. This confirms that communication compression in aggregation is both effective and necessary.

5.4 Efficiency Evaluation

To evaluate the efficiency, we test the training performance of Guard-GBDT over a single tree using 6 real-world datasets and compare it with SiGBDT and HEP-XGB. As shown in table 6, Guard-GBDT is better than SiGBDT and HEP-XGB in terms of runtime and communication cost. The reason is that, we leverage more streamlined approximations to replace inefficient and complicated sigmoid function and division function and use a communication-efficient protocol to reduce data transmission by compressing intermediate messages during gradient aggregation.

Table 5: Microbenchmarks with input size of 10^5

Secure component	Runtime on LAN (s)			Runtime on WAN (s)			Communaction cost (MB)		
	Guard-GBDT	**SiGBDT**	**HEP-XGB**	**Guard-GBDT**	**SiGBDT**	**HEP-XGB**	**Guard-GBDT**	**SiGBDT**	**HEP-XGB**
Sigmoid	34.51	88.07	92.63	36.74	94.38	99.95	9.44	27.46	25.33
Leaf weight	34.52	84.61	84.62	36.71	91.51	91.51	7.62	35.82	35.82
Split gain	0.18	152.09	181.34	0.21	171.01	190.12	9.15	61.79	71.79
Gradient aggregation	0.02	0.03	1.61	0.31	0.63	5.11	0.21	0.77	1.06

Table 6: Runtime for training one tree

Dataset	Runtime on LAN (s)			Runtime on WAN (s)			Communaction cost (MB)		
	Guard-GBDT	**SiGBDT**	**HEP-XGB**	**Guard-GBDT**	**SiGBDT**	**HEP-XGB**	**Guard-GBDT**	**SiGBDT**	**HEP-XGB**
breast-cancer	29.2	79.27	356.71	216.29	516.12	901.23	29.71	35.37	197.84
phishing	31.14	66.08	376.44	480.12	1158.12	2012.57	162.57	200.06	220.84
credit	55.15	105.89	612.08	716.29	1312.12	$\geq$ 1312.12	780.38	904.67	1167.36
skin	97.95	222.17	319.29	1216.29	$\geq$ 1216.29	$\geq$ 1216.29	1059.93	1556.48	1991.56
covertype	865.90	1206.07	1868.84	9012.12	$\geq$ 9012.12	$\geq$ 9012.12	23568.38	33781.76	34,810.88

Table 7: Runtime for training one tree with synthetic datasets under different settings.

N	F	B	D	Runtime on LAN (s)			Runtime on WAN (s)			Communaction cost (MB)		
				Guard-GBDT	**SiGBDT**	**HEP-XGB**	**Guard-GBDT**	**SiGBDT**	**HEP-XGB**	**Guard-GBDT**	**SiGBDT**	**HEP-XGB**
10k	10	8	4	20.41	38.46	286.18	288.93	573.08	964.52	21.51	95.42	172.94
50k	10	8	4	34.97	77.68	326.83	392.55	699.72	1107.70	105.93	475.64	768.99
10k	20	8	4	30.21	64.56	530.07	433.08	932.75	1794.44	42.26	189.22	319.78
10k	10	16	4	25.10	51.16	369.22	315.94	551.36	1160.52	42.35	189.56	310.64
10k	10	8	5	32.49	68.12	598.46	580.19	1159.55	1,929.04	46.48	204.29	346.07

On the breast cancer dataset in the LAN network, Guard-GBDT is 2.71× faster than SiGBDT and 12.21× faster than HEP-XGB. For the phishing dataset, Guard-GBDT's runtime is 2.12× faster than SiGBDT and 12.09× faster than HEP-XGB. For larger datasets like covertype, Guard-GBDT is 1.39× and 2.16× than SiGBDT and HEP-XGB, respectively. Also in WAN settings, Guard-GBDT is more efficient than the other two approaches. Guard-GBDT runs in 216.29 s on the breast cancer dataset, which is 2.39× and 4.17× faster than HEP-XGB and SiGBDT, respectively. For larger datasets like covertype, Guard-GBDT also renders a faster running time. In terms of communication, Guard-GBDT requires 29.71 MB on the breast-cancer dataset, which improves 1.19× than SiGBDT and 6.66× than SiGBDT and HEP-XGB, respectively. For larger datasets like covertype, Guard-GBDT saves 1.43× and 1.48× communication overhead than SiGBDT and HEP-XGB, respectively.

5.5 Scalability Evaluation

To evaluate scalability, we generate random synthetic datasets, as real-world data often fail to simultaneously satisfy specific constraints on dataset size (N) and feature size (F), such as $N = 50K$ and $F = 10$. The default experimental parameters are configured as follows: a dataset size of $N = 10,000$, a tree depth of $D = 4$, a feature size of $F = 10$, and a bucket size of $B = 8$. In scalability evaluation, we vary training parameters based on the default settings. For clarity, the table 7 presents a comprehensive comparison of scalability evaluation over one tree for Guard-GBDT, SiGBDT, and HEP-XGB. The results show that our Guard-GBDT outperforms both SiGBDT and HEP-XGB in terms of runtime and communication cost.

For the baseline, Guard-GBDT achieves 1.88× and 14.02× improvement than SiGBDT and HEP-XGB, respectively. With sample size N, feature size F, bucket size B, and tree depth D increasing, the performance of Guard-GBDT is more significant. When the dataset size is $N = 50k$ with the same configuration, Guard-GBDT is 2.22× faster than SiGBDT and 9.34× faster than HEP-XGB. When training a tree of depth $D = 5$ Guard-GBDT is 2.3× and 18.7× faster than SiGBDT and HEP-XGB, respectively. In terms of communication, the baseline test of Guard-GBDT's communication cost improves 4.43× and 8.04× than SiGBDT and HEP-XGB, respectively. As different configurations increase, Guard-GBDT's communication cost is lower. When training a tree of depth $D = 5$ on $10k$ samples with $F = 10$ features and $B = 8$ buckets, Guard-GBDT is 4.48× and 7.57× faster than SiGBDT and HEP-XGB, respectively.

6 CONCLUSION

This paper presents Gaurd-GBDT, a high-performance two-party framework for GBDT training. To improve efficiency, we design a new MPC-friendly approach that removes inefficient divisions and sigmoid functions in MPC. Besides, we present a communication-friendly aggregation protocol that can compress intermediate communication messages during gradient aggregation. Extensive experiments demonstrate that Guard-GBDT outperforms the state-of-the-art HEP-MPC and SiGBDT on the LAN and WAN networks.

REFERENCES

[1] Mark Abspoel, Daniel Escudero, and Nikolaj Volgushev. 2021. Secure training of decision trees with continuous attributes. Proceedings on Privacy Enhancing Technologies (2021).

[2] Jianli Bai, Xiangfu Song, Xiaowu Zhang, Qifan Wang, Shujie Cui, Ee-Chien Chang, and Giovanni Russello. 2023. Mostree: Malicious Secure Private Decision Tree Evaluation with Sublinear Communication. In Proceedings of the 39th Annual Computer Security Applications Conference. 799–813.

[3] Donald Beaver. 1995. Precomputing oblivious transfer. In Annual International Cryptology Conference. Springer, 97–109.

[4] Dayal Kumar Behera, Madhabananda Das, Subhra Swetanisha, and Janmenjoy Nayak. 2022. XGBoost regression model-based electricity tariff plan recommendation in smart grid environment. International Journal of Innovative Computing and Applications 13, 2 (2022), 79–87.

[5] Elette Boyle, Nishanth Chandran, Niv Gilboa, Divya Gupta, Yuval Ishai, Nishant Kumar, and Mayank Rathee. 2021. Function secret sharing for mixed-mode and fixed-point secure computation. In Annual International Conference on the Theory and Applications of Cryptographic Techniques. Springer, 871–900.

[6] Elette Boyle, Niv Gilboa, and Yuval Ishai. 2015. Function secret sharing. In Annual international conference on the theory and applications of cryptographic techniques. Springer, 337–367.

[7] Elette Boyle, Niv Gilboa, and Yuval Ishai. 2016. Function secret sharing: Improvements and extensions. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. 1292–1303.

[8] Elette Boyle, Niv Gilboa, and Yuval Ishai. 2019. Secure computation with preprocessing via function secret sharing. In Theory of Cryptography: 17th International Conference, TCC 2019, Nuremberg, Germany, December 1–5, 2019, Proceedings, Part I 17. Springer, 341–371.

[9] Ran Canetti. 2000. Security and composition of multiparty cryptographic protocols. Journal of CRYPTOLOGY 13 (2000), 143–202.

[10] Octavian Catrina and Amitabh Saxena. 2010. Secure computation with fixed-point numbers. In Financial Cryptography and Data Security: 14th International Conference, FC 2010, Tenerife, Canary Islands, January 25-28, 2010, Revised Selected Papers 14. Springer, 35–50.

[11] Tianqi Chen and Carlos Guestrin. 2016. Xgboost: A scalable tree boosting system. In Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining. 785–794.

[12] Kewei Cheng, Tao Fan, Yilun Jin, Yang Liu, Tianjian Chen, Dimitrios Papadopoulos, and Qiang Yang. 2021. Secureboost: A lossless federated learning framework. IEEE intelligent systems 36, 6 (2021), 87–98.

[13] Ke Cheng, Liangmin Wang, Yulong Shen, Yangyang Liu, Yongzhi Wang, and Lele Zheng. 2020. A Lightweight Auction Framework for Spectrum Allocation with Strong Security Guarantees. In IEEE INFOCOM 2020 - IEEE Conference on Computer Communications. 1708–1717. https://doi.org/10.1109/INFOCOM41043.2020.9155279

[14] Tianxiang Dai, Yufan Jiang, Yong Li, and Fei Mei. 2024. NodeGuard: A Highly Efficient Two-Party Computation Framework for Training Large-Scale Gradient Boosting Decision Tree. Cryptology ePrint Archive (2024).

[15] Sebastiaan De Hoogh, Berry Schoenmakers, Ping Chen, and Harm op den Akker. 2014. Practical secure decision tree learning in a teletreatment application. In Financial Cryptography and Data Security: 18th International Conference, FC 2014, Christ Church, Barbados, March 3-7, 2014, Revised Selected Papers 18. Springer, 179–194.

[16] Daniel Demmler, Thomas Schneider, and Michael Zohner. 2015. ABY-A framework for efficient mixed-protocol secure two-party computation.. In NDSS.

[17] Yamane El Zein, Mathieu Lemay, and Kévin Huguenin. 2023. PrivaTree: Collaborative Privacy-Preserving Training of Decision Trees on Biomedical Data. IEEE/ACM Transactions on Computational Biology and Bioinformatics 21, 1 (2023), 1–13.

[18] Milos D Ercegovac, Laurent Imbert, David W Matula, J-M Muller, and Guoheng Wei. 2000. Improving Goldschmidt division, square root, and square root reciprocal. IEEE Trans. Comput. 49, 7 (2000), 759–763.

[19] Wenjing Fang, Derun Zhao, Jin Tan, Chaochao Chen, Chaofan Yu, Li Wang, Lei Wang, Jun Zhou, and Benyu Zhang. 2021. Large-scale secure XGB for vertical federated learning. In Proceedings of the 30th ACM International Conference on Information & Knowledge Management. 443–452.

[20] Jun Feng, Laurence T Yang, Qing Zhu, and Kim-Kwang Raymond Choo. 2018. Privacy-preserving tensor decomposition over encrypted data in a federated cloud environment. IEEE Transactions on Dependable and Secure Computing 17, 4 (2018), 857–868.

[21] Yi Feng, Dujuan Wang, Yunqiang Yin, Zhiwu Li, and Zhineng Hu. 2020. An XGBoost-based casualty prediction method for terrorist attacks. Complex & Intelligent Systems 6 (2020), 721–740.

[22] Hossein Fereidooni, Samuel Marchal, Markus Miettinen, Azalia Mirhoseini, Helen Möllering, Thien Duc Nguyen, Phillip Rieger, Ahmad-Reza Sadeghi, Thomas Schneider, Hossein Yalame, et al. 2021. SAFELearn: Secure aggregation for private federated learning. In 2021 IEEE Security and Privacy Workshops (SPW). IEEE, 56–62.

[23] Fangcheng Fu, Jiawei Jiang, Yingxia Shao, and Bin Cui. 2019. An experimental evaluation of large scale GBDT systems. Proceedings of the VLDB Endowment 12, 11 (2019), 1357–1370.

[24] Oded Goldreich, Silvio Micali, and Avi Wigderson. 2019. How to play any mental game, or a completeness theorem for protocols with honest majority. In Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali. 307–328.

[25] Kanav Gupta, Neha Jawalkar, Ananta Mukherjee, Nishanth Chandran, Divya Gupta, Ashish Panwar, and Rahul Sharma. 2023. Sigma: Secure gpt inference with function secret sharing. Cryptology ePrint Archive (2023).

[26] Neha Jawalkar, Kanav Gupta, Arkaprava Basu, Nishanth Chandran, Divya Gupta, and Rahul Sharma. 2024. Orca: Fss-based secure training and inference with gpus. In 2024 IEEE Symposium on Security and Privacy (SP). IEEE, 597–616.

[27] Yufan Jiang, Fei Mei, Tianxiang Dai, and Yong Li. 2024. 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. 274–288.

[28] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. Lightgbm: A highly efficient gradient boosting decision tree. Advances in neural information processing systems 30 (2017).

[29] Yehuda Lindell. 2017. How to simulate it–a tutorial on the simulation proof technique. Tutorials on the Foundations of Cryptography: Dedicated to Oded Goldreich (2017), 277–346.

[30] Yehuda Lindell and Benny Pinkas. 2000. Privacy preserving data mining. In Annual international cryptology conference. Springer, 36–54.

[31] Yang Liu, Zhuo Ma, Ximeng Liu, Siqi Ma, Surya Nepal, Robert H Deng, and Kui Ren. 2020. Boosting privately: Federated extreme gradient boosting for mobile crowdsensing. In 2020 IEEE 40th international conference on distributed computing systems (ICDCS). IEEE, 1–11.

[32] Wen-jie Lu, Zhicong Huang, Qizhi Zhang, Yuchen Wang, and Cheng Hong. 2023. Squirrel: A Scalable Secure {Two-Party} Computation Framework for Training Gradient Boosting Decision Tree. In 32nd USENIX Security Symposium (USENIX Security 23). 6435–6451.

[33] Fucai Luo, Saif Al-Kuwari, and Yong Ding. 2022. SVFL: Efficient secure aggregation and verification for cross-silo federated learning. IEEE Transactions on Mobile Computing 23, 1 (2022), 850–864.

[34] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. Advances in neural information processing systems 32 (2019).

[35] Yu Shi, Guolin Ke, Zhuoming Chen, Shuxin Zheng, and Tie-Yan Liu. 2022. Quantized training of gradient boosting decision trees. Advances in neural information processing systems 35 (2022), 18822–18833.

[36] Zhihua Tian, Rui Zhang, Xiaoyang Hou, Lingjuan Lyu, Tianyi Zhang, Jian Liu, and Kui Ren. 2023. FederBoost: Private Federated Learning for GBDT. IEEE Transactions on Dependable and Secure Computing (2023).

[37] Di-ni Wang, Lang Li, and Da Zhao. 2022. Corporate finance risk prediction based on LightGBM. Information Sciences 602 (2022), 259–268.

[38] Fu-Yun Wang, Da-Wei Zhou, Han-Jia Ye, and De-Chuan Zhan. 2022. Foster: Feature boosting and compression for class-incremental learning. In European conference on computer vision. Springer, 398–414.

[39] Haoqi Wu, Wenjing Fang, Yancheng Zheng, Junming Ma, Jin Tan, Yinggui Wang, and Lei Wang. 2024. Ditto: Quantization-aware Secure Inference of Transformers upon MPC. arXiv preprint arXiv:2405.05525 (2024).

[40] Yuncheng Wu, Shaofeng Cai, Xiaokui Xiao, Gang Chen, and Beng Chin Ooi. 2020. Privacy preserving vertical federated learning for tree-based models. Proceedings of the VLDB Endowment 13, 12 (2020), 2090–2103.

[41] Wei Xu, Hui Zhu, Yandong Zheng, Fengwei Wang, Jiaqi Zhao, Zhe Liu, and Hui Li. 2024. ELXGB: An Efficient and Privacy-Preserving XGBoost for Vertical Federated Learning. IEEE Transactions on Services Computing (2024).

[42] Jiaqi Zhao, Hui Zhu, Wei Xu, Fengwei Wang, Rongxing Lu, and Hui Li. 2022. SGBoost: An efficient and privacy-preserving vertical federated tree boosting framework. IEEE Transactions on Information Forensics and Security 18 (2022), 1022–1036.

[43] Yifeng Zheng, Shuangqing Xu, Songlei Wang, Yansong Gao, and Zhongyun Hua. 2023. Privet: A privacy-preserving vertical federated learning service for gradient boosted decision tables. IEEE Transactions on Services Computing 16, 5 (2023), 3604–3620.

A CORRECTNESS AND SECURITY OF FSS

Definition 4 (Correctness and Security of FSS[7]). *Let $f(x)$ be one description in the function family $\mathcal{F}$, i.e, $f(x) \in \mathcal{F}$. We say that (**FSS.Gen, FSS.Eval**) as in Definition 1 is an FSS scheme for $\mathcal{F}$ with respect to leakage (Leak) if it satisfies the following.*

- ***Correctness**: For all $\hat{f}(x)$ describing $f(x)\colon \mathbb{G}^{in} \rightarrow \mathbb{G}^{out}$, and every $x \in \mathbb{G}^{in}$, if $(\mathcal{K}_0, \mathcal{K}_1) \leftarrow FSS.Gen(1^{\lambda}, \hat{f}(x))$, then*

$Pr[FSS.Eval(0, \mathcal{K}_0, x) + FSS.Eval(1, \mathcal{K}_1, x) = f(x)] = 1$, where $\mathbb{G}^{in}$ and $\mathbb{G}^{out}$ represent the input and output domains, respectively, of the function $\hat{f}(x)$.

- ***Security:*** *There exists a probabilistic polynomial-time (PPT) algorithm simulator Sim_b for $b \in \{0, 1\}$. This simulator ensures that for any sequence $\{f_\phi(x)\}_{\phi \in \mathbb{N}}$ of polynomial-size function descriptions from $\mathcal{F}$ and corresponding polynomial-size input sequences x_ϕ for $f_\phi(x)$, the outputs from the following **Real** and **Ideal** experiments are computationally indistinguishable:*
  *–**Real:**$(\mathcal{K}_0, \mathcal{K}_1) \leftarrow Gen(1^\lambda, \hat{f}_\phi(x))$ and output $\mathcal{K}_b$.*
  *–**Ideal:** Output $Sim_b(1^\lambda, Leak(\hat{f}_\lambda(x)))$.*

B SECURITY ANALYSIS

Guard-GBDT follows the standard definition of security against semi-honest adversaries (Definition 3). We can prove the security of our protocols using the real-world/idea-word simulation paradigm[24] in a bottom-up fashion:

1. First, we instantiate the two cryptographic primitives of ASS and FSS that we used, including secure multiplication, secure addition, and DCF. For these primitives, we can refer to ABY[16] to derive the security of ASS and Orca[26] to derive the security of FSS under the semi-honest model
2. Next, we prove the security of our proposed protocols in Theorem 1-3.
3. Finally, we put everything together and prove the security of the secure training protocol of Guard-GBDT in Theorem 4.

To prove the security of all proposed protocols under the semi-honest adversary model, we employ the sequential composition theory [] which is a tool that enables us to analyze the security of cryptographic protocols in a modular approach. Let a protocol $\prod_{\mathcal{F}}$ be securely computed by invoking sub-protocols $\prod_{f_1}, \ldots, \prod_{f_m}$ for computing $f_i, \ldots, f_m$. The theorem states that it suffices to consider the execution of $\prod_{\mathcal{F}}$ in a $(f_1, \ldots, f_m)$-hybrid model. If all invoked sub-protocols $\prod_{f_1}, \ldots, \prod_{f_m}$ is secure against the semi-honest adversaries in the hybrid model, then $\prod_{\mathcal{F}}$ protocol also secure against the semi-honest adversaries. Next, we provide the security proofs for our cryptographic protocols.

Theorem 1. *If the arithmetic operations of ASS and DCF of FSS are secure against semi-honest adversaries, then our $\Pi_{LUT^n_\delta}$ protocol and $\Pi_{LUT^n_w}$ protocol are secure under the semi-honest adversaries model.*

Proof. We first prove the security of $\Pi_{\mathrm{LUT}^n_\delta}$ protocol. In preprocessing on the offline phase, the secure third party (STP) provides only the randomness used to generate keys for the online computation and does not receive any private information from the computing parties P_0 and P_1. Hence, the offline phase of this protocol is trivially secure against the semi-honest corruption of STP. To prove the security of this protocol in the online phase, we construct simulators in two distinct cases depending on which party is corrupted. For all PPT adversaries, the corrupted party's view from the interaction between P_0 and P_1 is indistinguishable from its view when interacting with a simulator instead. Deriving from $\Pi_{\mathrm{LUT}^n_\delta}$ protocol, we see that the interactive messages between P_0 and P_1 include $m_i = x + \alpha_i - \omega_i$ (Line 10 in Algorithm 2) for $i \in \{0, n-1\}$ and data transferred in executing the arithmetic operations and DCF. Since the existing works have proved that the arithmetic operations and DCF are secure against semi-honest adversaries, any PPT adversary cannot distinguish the simulator's views from the data transferred in these operations. As for the interactive messages $\{m_i\}$, we can construct a simulator $\mathcal{S}_0$ to simulate P_0's view. S_0 chooses n random integers $\{r'_i \in \mathbb{Z}_{2^\ell}\}$ to simulate n interactive messages $\{m_i = x + \alpha_i - \omega_i\}$. Recall that the matrices $\{\alpha_i\}$ are generated randomly in the offline phase. Thus $\{r'_i\}$ and $\{m_i = x + \alpha_i - \omega_i\}$ are indistinguishable such that any PPT adversary cannot distinguish from its view when interacting with the simulator $\mathcal{S}_0$ instead. Clearly, a simulator $\mathcal{S}_1$ can do the same work as above to simulate P_1's view. According to the composition theory, we can claim that $\Pi_{\mathrm{LUT}^n_\delta}$ protocol is secure under the semi-honest adversaries model.

Since the implementation and execution steps of both protocols are similar, the security proof for the $\Pi_{\mathrm{LUT}^n_w}$protocol is analogous to that of the $\Pi_{\mathrm{LUT}^n_\delta}$ protocol. We can therefore claim that the $\Pi_{\mathrm{LUT}^n_w}$ protocol is also secure under the semi-honest adversaries model. □

Theorem 2. *If the arithmetic operations of ASS are secure against semi-honest adversaries, then our Π_{3Mul} protocol is secure under the semi-honest adversaries model.*

Proof. The proof of Theorem 2 is similar to that of Theorem 1, and we will prove the security of Π_{3Mul} protocol briefly. In preprocessing on the offline phase, STP provides only the randomness needed to generate correlated keys of Π_{3Mul} protocol and does not receive any private information from the P_0 and P_1. Hence, the offline phase of this protocol is trivially secure against the semi-honest corruption of STP. In the online phase, we see that the interactive messages between P_0 and P_1 include data transferred in executing the arithmetic operations and $m_i = x_i - r_i$ (Line **??** in Algorithm **??**) for $i \in \{0, 1, 2\}$. Since the existing works have proved that the arithmetic operations of ASS are secure against semi-honest adversaries, any PPT adversary cannot distinguish the simulator's views from the data transferred in these operations. For m_i, we can construct a simulator $\mathcal{S}_b$ to simulate each P_b's view, where $b \in \{0, 1\}$. The simulator S_b chooses 3 random integers $\{r'_i \in \mathbb{Z}_{2^\ell}\}$ to simulate 3 interactive messages $\{m_i = x_i + r_i\}$, where r_i are randomness generated form offline phase. Since the randomness $\{r'_i\}$ and $\{r_i\}$ are indistinguishable such that any PPT adversary cannot distinguish from its view when interacting with the simulator $\mathcal{S}_b$ instead. Thus, we can claim that Π_{3Mul} protocol is secure under the semi-honest adversaries model according to the composition theory. □

Theorem 3. *If the arithmetic operations of ASS are secure against semi-honest adversaries, then our Π_{Agg} protocol is secure under the semi-honest adversaries model.*

Proof. Similar to Theorem 2, we will prove the security of Π_{Agg} protocol briefly. In preprocessing on the offline phase, STP provides only randomness used to generate keys and does not receive any private information from the P_0 and P_1. Hence, the offline phase of this protocol is trivially secure against the semi-honest corruption of STP. In the online phase, the interactive messages between P_0 and P_1 include data transferred in executing the arithmetic operations and $\{\hat{s}_i = s_i \oplus r_{s_i}, \hat{g}'_i = g_i + r_i + 2^{\ell'} \mod 2^{\ell'}\}$ (Line 13 and 14 in Algorithm 4) for $i \in [0, N]$. Since the arithmetic operations of ASS are secure against semi-honest adversaries, any PPT adversary cannot distinguish the simulator's views from the data transferred in these operations. As for the interactive messages $\{\hat{s}_i = s_i \oplus r_{s_i}, \hat{g}'_i = g_i + r_i + 2^{\ell'} \mod 2^{\ell'}\}$ in which r'_{s_i} and r_i are randomness form offline phase, we can construct a simulator $\mathcal{S}_b$ to simulate each P_b's view, where $b \in \{0, 1\}$. The simulator S_b chooses N pairs of randomness $r'_{s_i} \in \{0, 1\}$ and $r'_{g_i} \mod 2^{\ell'}$ to simulate $\{\hat{s}_i, \hat{g}'_i\}$. Since the randomness r_i and r_{s_i} are random such that r'_{g_i} and $\hat{g}'_i$ are indistinguishable and r'_{s_i} and $\hat{s}_i$ are also indistinguishable. Thus, any PPT adversary cannot distinguish from its view when interacting with the simulator $\mathcal{S}_b$ instead. According to the composition theory, we can claim that Π_{Agg} protocol is secure under the semi-honest adversaries model □

Theorem 4. *As long as our proposed sub-protocols are secure against semi-honest adversaries, Guard-GBDT is secure under the semi-honest adversaries model.*

Proof. Guard-GBDT are stitched together from our proposed sub-protocols. Beyond the input and output of the protocols, the interactive messages of Guard-GBDT consist only of intermediate values and outputs for subprotocols. Since the security of these sub-protocols is proved in Theorem 1-3, their intermediate messages are individual and uniformly random. In the case of corrupted P_0 or P_1, the simulator $\mathcal{S}_b$ can choose uniformly random values to simulate these intermediate messages such that any PPT adversary cannot distinguish from its view when interacting with the simulator $\mathcal{S}_b$ instead. According to the composition theory, we can claim that the Guard-GBDT is secure under semi-honest adversary model. □