Squirrel: A Scalable Secure Two-Party Computation Framework for Training Gradient Boosting Decision Tree

Wen-jie Lu and Zhicong Huang, *Alibaba Group;* Qizhi Zhang, *Ant Group;* Yuchen Wang, *Alibaba Group;* Cheng Hong, *Ant Group*

https://www.usenix.org/conference/usenixsecurity23/presentation/lu

This paper is included in the Proceedings of the 32nd USENIX Security Symposium.

August 9–11, 2023 • Anaheim, CA, USA

978-1-939133-37-3

Open access to the Proceedings of the 32nd USENIX Security Symposium is sponsored by USENIX.

Squirrel: A Scalable Secure Two-Party Computation Framework for Training Gradient Boosting Decision Tree

Wen-jie Lu
Alibaba Group

Zhicong Huang
Alibaba Group

Qizhi Zhang
Ant Group

Yuchen Wang
Alibaba Group

Cheng Hong*
Ant Group

Abstract

Gradient Boosting Decision Tree (GBDT) and its variants are widely used in industry, due to their strong interpretability. Secure multi-party computation allows multiple data owners to compute a function jointly while keeping their input private. In this work, we present *Squirrel*, a two-party GBDT training framework on a vertically split dataset, where two data owners each hold different features of the same data samples. *Squirrel* is private against semi-honest adversaries, and no sensitive intermediate information is revealed during the training process. *Squirrel* is also scalable to datasets with millions of samples even under a Wide Area Network (WAN).

Squirrel achieves its high performance via several novel co-designs of the GBDT algorithms and advanced cryptography. Especially, 1) we propose a new and efficient mechanism to hide the sample distribution on each node using oblivious transfer. 2) We propose a highly optimized method for gradient aggregation using lattice-based homomorphic encryption (HE). Our empirical results show that our method can be three orders of magnitude faster than the existing HE approaches. 3) We propose a novel protocol to evaluate the sigmoid function on secretly shared values, showing $19\times$-$200\times$-fold improvements over two existing methods. Combining all these improvements, *Squirrel* costs less than 6 seconds per tree on a dataset with 50 thousands samples which outperforms `Pivot` (VLDB 2020) by more than $28\times$. We also show that *Squirrel* can scale up to datasets with more than one million samples, e.g., about 90 seconds per tree over a WAN.

1 Introduction

Gradient Boosting Decision Tree (GBDT) [25] and its variants such as LightGBM [38] and XGBoost [15] are widely used tree-based machine learning algorithms. Due to their high performances as well as strong interpretability, the GBDT algorithms have been regarded as a standard recipe for many industrial tasks such as fraud detection [11, 51], financial risk management [1, 55] and online advertisement [30, 43]. In such tasks, a GBDT model owner could improve the prediction performance of his model by integrating more data of different features. For example, an insurance company might want to improve its risk assessment model by integrating more features of its customers from a hospital. However, with privacy concerns and regulations (e.g., HIPPA and GDPR) coming into force, it might be unsuitable to send the plain data for a cross-enterprise collaborative GBDT training.

Secure multi-party computation (MPC) [31, 60] is a powerful tool that allows multiple data owners to jointly compute a function without revealing anything beyond the function result. The recent works of `Pivot` [58] and `HEP-XGB` [22] have demonstrated the possibility to train GBDT models collaboratively using MPC techniques. However, there are still many obstacles to deploying their works in practice. For instance, `Pivot` requires a communication-intensive pre-processing phase, e.g., generating Beaver's triples [8], which can be hundreds of times more expensive than its online training phase. On the other hand, `HEP-XGB` heavily relies on a semi-honest third party (STP) [50], e.g., a Trusted Execution Environment, to improve their performance. In a word, the performance of `HEP-XGB` might drop significantly without a STP. Even under their sweet-spot setting, the running times of `Pivot` and `HEP-XGB` are still long due to the massive number of expensive cryptographic operations, such as encrypting millions of message using the Paillier cryptosystem [48].

The reason why it is difficult to design a scalable MPC solution for the GBDT training might be three-fold: 1) We require oblivious algorithms to prevent intermediate information from leaking. For example, when we are choosing a split point for a tree node, in the context of plaintext, it suffices to scan the samples belonging to that node only. However, in the context of MPC, we need a full scan of the whole dataset, because the sample distribution (i.e., which samples belong to which node) is sensitive, and thus should be kept private. 2) GBDT algorithms involve both complicated non-linear operations (e.g., sigmoid and division) and a vast number of linear operations (e.g., large matrix multiplications), requiring efforts

*This work was partially done when Hong was at Alibaba Group.

on both sides to improve the scalability. 3) MPC techniques usually introduce a high communication cost. However, a high-speed bandwidth (e.g., 10Gbps) is barely available for cross-enterprise collaborative learning in practice.

All the above bring up a rigorous challenge for designing a secure GBDT training protocol that is computationally fast and communication-friendly.

1.1 Related Work

The first MPC-based privacy-preserving decision tree training protocol is proposed by Lindell et al. [42], where they assume secret-shared data over two parties. Their protocol is based on oblivious transfer and Yao's garbled circuit. After that, Hoogh et al. [19] propose secret sharing-based protocols for any number of parties. Moreover Abspoel et al. [3] propose an oblivious sorting network for training decision trees privately. The recent MPC-based frameworks `Pivot` [58] and `HEP-XGB` [22] have made considerable improvements, and demonstrate the ability to privately train gradient boosting trees on vertically partitioned datasets. Particularly, both `Pivot` and `HEP-XGB` suggest using mixed cryptographic primitives to achieve a better performance. More specifically, `Pivot` is communication intensive and thus it prefers the parties to be interconnected with a high-speed bandwidth. `HEP-XGB` depends on a STP to generate correlated randomness efficiently in a pre-processing phase.

The most computationally expensive step in the existing privacy-preserving GBDT [16, 22, 27, 58] is gradient aggregation. These methods depend on an additive homomorphic encryption (HE) such as the Paillier cryptosystem. For a Paillier-like HE, however, both encryption and decryption involve multiple modular exponentiation operations with big integers, making them extremely expensive to compute. [52] improves the encryption performance of Paillier by about $3\times$ using a specialized hardware. Unfortunately, even with these optimizations, the operations of Paillier-like HEs are still too expensive for GBDT training. As a result, most of the existing approaches resort to a short key for efficiency at the cost of a legacy security level. For instance, `Pivot` uses a 1024-bit key while the recommendation from NIST [47] for long term security is to use a 3072-bit key.

Federated learning (FL) [16, 17, 27, 45] is paradigm that transfers the intermediate values (e.g., gradients) instead of the raw data itself. However, when the number of participants is small (e.g., only two parties), one can hardly protect his private information by "hiding a tree in the forest". Revealing intermediate results, e.g., statistics or model updates, produces a potential leakage about the training data. Take the sample distribution as an example. Suppose a sample x_0 is categorized to the left child due to its feature say *age* on one node, and the other sample x_1 is categorized to the right child on the same node. Then the adversary can infer that the age of x_0 is less than the age of x_1. Many works have shown that FL-based solutions that leak intermediate values are vulnerable to the attacks [26, 28, 37].

Differential privacy [21] can be used as an orthogonal primitive to enhance the security of the existing privacy-preserving GBDT. However, the GBDT prediction performance might drop significantly. [23] reports a accuracy drop of more than 20% when applying differential privacy techniques to GBDT.

We refer to the excellent survey by Chatel et al. [13] for a more comprehensive discussion on the privacy-preserving tree-based model learning. To the best of our knowledge, a scalable MPC framework for the GBDT training without a heavy dependency on a high-speed network and trusted hardware is still missing.

1.2 Our Contributions

In this manuscript, we present *Squirrel*, a two-party framework for privately training GBDT models on a distributed dataset among two parties. *Squirrel* is private and no sensitive intermediate information is revealed during the training. *Squirrel* is also scalable to datasets with millions of samples even under a Wide Area Network. *Squirrel* achieves its good performance via a careful co-design of GBDT, lattice-based cryptography, oblivious transfer, and secret sharing. Our contributions can be summarized as follows.

1. *New mechanism for securing the sample distribution.* Both [58] and [22] have their own mechanism to keep the sample distribution secret for each tree node. In *Squirrel*, we propose a new mechanism that is significantly faster than the HE-based mechanism in [58], and renders a lower communication overhead (more than 50% off) than the Beaver's triple -based mechanism used by [22].

2. *Orders of magnitude faster and less communication gradient aggregation.* We propose an efficient gradient aggregation protocol. We design special primitives to fully leverage the algebraic properties of the underlying lattice-based HEs. Our approach is up to 3 orders of magnitude faster than the existing methods that rely on Paillier-like HEs.

3. *Efficient and accurate sigmoid.* To train GBDT for a binary classification task, we propose an efficient and effective two-party protocol Seg3Sigmoid for the sigmoid function. Seg3Sigmoid is about $9\times$ faster (under WAN) than the existing OT-based protocol used by `Pivot`. With about $1.4\times$ more communication, Seg3Sigmoid is about $200\times$ faster than the recent approach [4] based on Function Secret Sharing. Seg3Sigmoid is also accurate, introducing less than 3.0% F1-score drop on 6 real-word datasets, comparing to a plain GBDT baseline.

4. *Extensive evaluations.* We implement the proposed protocols and optimizations. With all our optimizations,

$\mathsf{GBDT}(I_k, \mathbf{X}, \mathsf{state}, \mathsf{pp})$.

1. If $1 \leq k < 2^{D-1}$ is not reaching the maximum depth,
   (a) Find the feature $z_*^{(k)} \in [m]$ and the threshold $u_*^{(k)} \in \mathbb{R}$ that best split the samples in I_k.
   (b) Let $I_{2k} \subset I_k$ and $I_{2k+1} = I_k / I_{2k}$ be the partition of I_k according to the split $(z_*^{(k)}, u_*^{(k)})$. That is the $z_*^{(k)}$-th feature of the samples in I_{2k} (resp. I_{2k+1}) has a smaller (resp. larger) value than the threshold $u_*^{(k)}$.
   (c) Return a tree whose root is attached with $(z_*^{(k)}, u_*^{(k)})$ and has edges to trees $\mathsf{GBDT}(I_{2k}, \mathbf{X}, \mathsf{state}, \mathsf{pp})$ and $\mathsf{GBDT}(I_{2k+1}, \mathbf{X}, \mathsf{state}, \mathsf{pp})$.
2. Otherwise, compute a weight $w^{(k)} = -\dfrac{\sum_{i \in I_k} \mathbf{g}[i]}{\sum_{i \in I_k} \mathbf{h}[i] + \gamma}$ and update the prediction score of samples $\tilde{\mathbf{y}}[i] = \tilde{\mathbf{y}}[i] + w^{(k)}$ for all $i \in I_k$. Return a leaf node attached with the weight $w^{(k)}$.

Figure 1: The GBDT algorithm for training one full tree.

Squirrel outperforms the state of the art. The total running time of *Squirrel* is 28× faster than `Pivot`'s online time[1]. *Squirrel* is about 3× faster than the TEE-aided `HEP-XGB` over a WAN. We also show the scalability of *Squirrel* on a dataset with one million samples. The training time is about 90 seconds per tree under the WAN.

2 Preliminaries

2.1 Notations

We denote by $[n]$ the set $\{0, \cdots, n-1\}$ for $n \in \mathbb{N}$. For a set $\mathcal{D}$, $x \in_R \mathcal{D}$ means x is sampled from $\mathcal{D}$ uniformly at random. We use $\lceil \cdot \rceil$, $\lfloor \cdot \rfloor$ and $\lfloor \cdot \rceil$ to denote the ceiling, flooring, and rounding function, respectively. We denote $\mathbb{Z}_q = \mathbb{Z} \cap [0, q)$ for $q \geq 2$. The logical AND and XOR is $\wedge$ and $\oplus$, respectively. Let $\mathbf{1}\{\mathcal{P}\}$ denote the indicator function that is 1 when the predicate $\mathcal{P}$ is true and 0 when $\mathcal{P}$ is false. We use lower-case letters with a "hat" symbol such as $\hat{a}$ to represent a polynomial, and $\hat{a}[j]$ to denote the j-th coefficient of $\hat{a}$. We use the dot symbol $\cdot$ such as $\hat{a} \cdot \hat{b}$ to represent the multiplication of polynomials. For a 2-power number N, and $q > 0$, we write $\mathbb{A}_{N,q}$ to denote the set of integer polynomials $\mathbb{A}_{N,q} = \mathbb{Z}_q[X]/(X^N + 1)$. We use bold letters such as $\mathbf{a}, \mathbf{M}$ to represent vectors and matrix, and use $\mathbf{a}[j]$ to denote the j-th component of $\mathbf{a}$ and use $\mathbf{M}[j, i]$ to denote the (j, i) entry of $\mathbf{M}$. The Hadamard product of vectors is written as $\mathbf{a} \odot \mathbf{b}$.

Other notations related to GBDT are summarized as follows. n and m denote number of samples and features, respectively. D is the maximum depth of a tree. The node index in a full and balanced tree iterates in $k \in \{1, 2, \cdots, 2^D - 1\}$.

[1]The pre-processing costs of `Pivot` were not reported in their paper.

2.2 Gradient Boosting Decision Tree

GBDT trains a sequence of $T > 0$ decision trees $\mathcal{T}_t : \mathbb{R}^m \mapsto \mathbb{R}$ in an additive manner. On the input $\mathbf{x} \in \mathbb{R}^m$, each tree $\mathcal{T}_t$ will classify it to one leaf node and results at a weight. The final prediction is given as the sum of T weights $\tilde{y} = \sum_{t=1}^{T} \mathcal{T}_t(\mathbf{x})$.

A GBDT tree is constructed top-down in a recursive manner. At the root, each feature is tested to determine how well it alone classifies the current samples. The "best" feature and threshold (to be discussed below) are then chosen and we split the current samples into two partitions by them. We then recursively call GBDT on the two partitions. See Fig. 1 for a description of the GBDT training algorithm for one decision tree. The matrix $\mathbf{X} \in \mathbb{R}^{n \times m}$ indicates the training dataset. The state parameter state contains stateful values, including $\tilde{\mathbf{y}}$ the prediction of all the samples, $\mathbf{g}$ and $\mathbf{h}$, the first- and second-order gradient of all the samples. These values will be updated along the training process. The public parameter pp contains some fixed values such as the maximum depth D and a regularizer $\gamma > 0$.

What remains is to explain how to choose the best predicting feature and the threshold. However, this is intractable to test all possible thresholds in practice. Most GBDT frameworks [15, 38] thus accelerate the training process by discretizing the numeric features. For instance, the 'Age' feature can be discretized into 2 sorted bins like Age $<$ 18 and Age $\geq$ 18. To ease the presentation, we assume all features are discretized into B sorted bins in this work. Given a sample set I_k, we can move the subset to the left child $I_{2k}^{(z,u)} = \{i | \mathbf{X}[i, z] \leq \mathsf{bin}^{(z,u)} \wedge i \in I_k\}$ and $\mathsf{bin}^{(z,u)} \in \mathbb{R}$ is a threshold defined by the u-th bin of the z-th feature. The samples in $I_{2k+1} = I_k / I_{2k}$ are moved to the right child. The score under this partition is then given by

$$\mathcal{G}^{(k,z,u)} = \frac{(\sum_{i \in I_{2k}^{(z,u)}} \mathbf{g}[i])^2}{\gamma + \sum_{i \in I_{2k}^{(z,u)}} \mathbf{h}[i]} + \frac{(\sum_{i \in I_{2k+1}^{(z,u)}} \mathbf{g}[i])^2}{\gamma + \sum_{i \in I_{2k+1}^{(z,u)}} \mathbf{h}[i]} - \frac{(\sum_{i \in I_k} \mathbf{g}[i])^2}{\gamma + \sum_{i \in I_k} \mathbf{h}[i]}. \tag{1}$$

The idea is therefore to iterate all (z, u) pairs and find the one that gives the maximum score. We designate it as the 'split identifier' on the node k.

The prediction vector $\tilde{\mathbf{y}}$ is usually initialized randomly for the 1st tree. Once a GBDT tree is built, we update the gradient vectors before moving to the next tree. For instance, for a binary classification task using the cross-entropy loss, the gradients are computed as $\mathbf{g} = \mathbf{y} - \mathbf{p}$ and $\mathbf{h} = \mathbf{p} \odot (1 - \mathbf{p})$ where $\mathbf{y}$ is the sample label vector and $\mathbf{p} = 1/(1 + \exp(-\tilde{\mathbf{y}}))$.

2.3 Cryptographic Primitives

2.3.1 Additive Secret Sharing

Throughout this manuscript, we use 2-out-of-2 additive secret sharing schemes over the ring $\mathbb{Z}_{2^\ell}$. An ℓ-bit ($\ell \geq 2$) value x

is additively shared as $\langle x\rangle_0$ and $\langle x\rangle_1$ where $\langle x\rangle_l$ is a random share of x held by P_l. To reconstruct the value x, we compute the modulo addition, i.e., $x \equiv \langle x\rangle_0 + \langle x\rangle_1 \bmod 2^\ell$. For a real value $\tilde{x} \in \mathbb{R}$, we first encode it as a fixed-point value $x = \lfloor \tilde{x}2^f \rfloor \in [-2^{\ell-1}, 2^{\ell-1})$ under a specified precision $f > 0$ before secretly sharing it. For a boolean value $z \in \{0,1\}$, we use $\langle z\rangle_0^B$ and $\langle z\rangle_1^B$ to denote the shares of z such that $z = \langle z\rangle_0^B \oplus \langle z\rangle_1^B$. Also we omit the subscript and only write $\langle x\rangle$ or $\langle z\rangle^B$ when the ownership is irrelevant from the context.

2.3.2 Oblivious Transfer

We rely on oblivious transfer (OT) for the non-linear computation in GBDT. In a general 1-out-of-2 OT, a sender inputs two messages m_0 and m_1 of length ℓ bits and a receiver inputs a choice bit $c \in \{0,1\}$. At the end of the protocol, the receiver learns m_c, whereas the sender learns nothing. When sender messages are correlated, the Correlated OT (COT) is more efficient in communication [7]. In our additive COT, a sender inputs a function $f(x) = x + \Delta$ for some $\Delta \in \mathbb{Z}_{2^\ell}$, and a receiver inputs a choice bit c. At the end of the protocol, the sender learns $x \in \mathbb{Z}_{2^\ell}$ whereas the receiver learns $x + c \cdot \Delta \in \mathbb{Z}_{2^\ell}$. In this work, we use the `Ferret` [59] protocol for a lower communication COT. `Ferret` exchanges about $O(\ell)$ bits for each choice on ℓ-bit strings.

2.3.3 Lattice-based Additive Homomorphic Encryption

A homomorphic encryption (HE) of x enables computing the encryption of $F(x)$ without the knowledge of the decryption key. In this work, we use two lattice-based HEs, i.e., learning with errors (LWE) HE and its ring variant (RLWE). These two HEs share a set of public parameters $\mathsf{HE.pp} = \{N, q, p\}$. Particularly, we set the plaintext modulus $p = 2^\ell$ in this work. We leverage the following functions.

- **KeyGen.** Generate the RLWE key pair $(\mathsf{sk}, \mathsf{pk})$ where the secret key $\mathsf{sk} \in \mathbb{A}_{N,q}$ and the public key $\mathsf{pk} \in \mathbb{A}_{N,q}^2$. We identify the LWE secret key $\mathbf{s} \in \mathbb{Z}_q^N$ as the coefficient vector of sk, i.e., $\mathbf{s}[j] = \mathsf{sk}[j]$ for all $j \in [N]$.

- **Encryption.** We write $\mathsf{RLWE}_{\mathsf{pk}}^{N,q,p}(\hat{m})$ to denote the RLWE ciphertext of $\hat{m} \in \mathbb{A}_{N,p}$ under the key pk. An RLWE ciphertext is given as polynomials tuple $(\hat{b}, \hat{a}) \in \mathbb{A}_{N,q}^2$. We write $\mathsf{LWE}_{\mathbf{s}}^{N,q,p}(m)$ to denote the LWE ciphertext of of $m \in \mathbb{Z}_q$ under the key $\mathbf{s}$. An LWE ciphertext is given as a vector $(b, \mathbf{a}) \in \mathbb{Z}_p^{N+1}$.

- **Addition** ($\boxplus$)**.** Given two LWE ciphertexts $\mathsf{ct}_0 = (b_0, \mathbf{a}_0)$ and $\mathsf{ct}_1 = (b_1, \mathbf{a}_1)$ that encrypts $m_0 \in \mathbb{Z}_p$ and $m_1 \in \mathbb{Z}_p$ respectively, the operation $\mathsf{ct}_0 \boxplus \mathsf{ct}_1$ computes the LWE tuple $(b_0 + b_1 \bmod q, \mathbf{a}_0 + \mathbf{a}_1 \bmod q)$ which can be decrypted to $m_0 + m_1 \bmod p$. The RLWE homomorphic addition is computed similarly but over the ring $\mathbb{A}_{N,q}$.

- **PackLWEs.** Given LWE ciphertexts $\{\mathsf{LWE}_{\mathsf{sk}}^{N,q,p}(m_i)\}_i$ for $i \in [2^n]$ $(2^n \le N)$, we can homomorphically merge them into one RLWE ciphertext that decrypts to a polynomial $\hat{m} \in \mathbb{A}_{N,p}$ under the secret key sk satisfying $\hat{m}[(N/2^n) \cdot i] = m_i$ for all $i \in [2^n]$. We defer the details of $\mathsf{PackLWEs}$ to Chen et al.'s paper [14].

2.3.4 Mixed Primitives System

The oblivious GBDT algorithm involves both a large number of linear operations and complicated non-linear operations. We use the lattice-based HEs to utilize the players' local computation power as much as possible, and switch to secret sharing where HE is unsuitable in terms of functionality. Specifically, we use the following conversions to safely switch forth-and-back between secret sharing and HE.
Arithmetic Share to HE $\mathsf{A2H}$**.** We write $\langle \mathbf{a}\rangle_l^H$ to denote RLWE ciphertext(s) that held by P_l but encrypted under P_{1-l}'s key. Converting the arithmetic share of N-sized vector $\langle \mathbf{a}\rangle$ to homomorphic encryption form can be done by evaluating the modulo addition homomorphically. Specifically, P_l arranges his shares into a polynomial $\hat{a}_l = \sum_{i=0}^{N-1} \langle \mathbf{a}[i]\rangle_l X^i$, and then sends a ciphertext $\mathsf{RLWE}_{\mathsf{pk}_l}^{N,q,2^\ell}(\hat{a}_l)$ to P_{1-l} using his key pk_l. Then P_{1-l} computes $\langle \mathbf{a}\rangle_{1-l}^H = \mathsf{RLWE}_{\mathsf{pk}_l}^{N,q,2^\ell}(\hat{a}_l) \boxplus \hat{a}_{1-l}$.
HE to Arithmetic Share $\mathsf{H2A}$ **([34]).** A conversion from $\langle \mathbf{a}\rangle_l^H$ to the arithmetic share $\langle \mathbf{a}\rangle$ is done as follows: P_l samples $\hat{r} \in_R \mathbb{A}_{N,q}$, and sends the sum $\mathsf{ct} = \langle \mathbf{a}\rangle_l^H \boxplus \hat{r}$ to the opposite. Then P_{1-l} decrypts ct and outputs the coefficients vector as the share $\langle \mathbf{a}\rangle_{1-l}$. On the other hand, P_l sets $\langle \mathbf{a}[i]\rangle_l = -\lceil 2^\ell \cdot \hat{r}[i]/q \rfloor \bmod 2^\ell$ for all $i \in [N]$.

3 Problem Statement

3.1 Threat Model and Privacy

Similar to previous work [3, 42, 58], we target privacy against a *static and semi-honest* probabilistic polynomial time (PPT) adversary following the simulation paradigm [10, 32]. We recap the privacy definition from [32, 42]. Let $F : \{0,1\}^* \times \{0,1\}^* \mapsto \{0,1\}^* \times \{0,1\}^*$ be a deterministic functionality where $F_0(x_0, x_1)$ (resp. $F_1(x_0, x_1)$) denotes the 1st element (resp. the 2nd) of $F(x_0, x_1)$ and let Π be a two-party protocol for computing F. The view of P_0 (resp. P_1) during an execution of Π on (x_0, x_1) is denoted $\mathcal{V}_0^\Pi(x_0, x_1)$ (resp. $\mathcal{V}_1^\Pi(x_0, x_1)$).

Definition 1 (Privacy ([32, 42])) *For a function F, we say that Π privately computes F if there exist PPT algorithms, denoted $\mathcal{S}_0$ and $\mathcal{S}_1$, such that*

$$\{\mathcal{S}_0(x_0, F_0(x_0, x_1))\}_{x_0, x_1 \in \{0,1\}^*} \stackrel{c}{\equiv} \{\mathcal{V}_0^\Pi(x_0, x_1)\}_{x_0, x_1 \in \{0,1\}^*}$$
$$\{\mathcal{S}_1(x_1, F_1(x_0, x_1))\}_{x_0, x_1 \in \{0,1\}^*} \stackrel{c}{\equiv} \{\mathcal{V}_1^\Pi(x_0, x_1)\}_{x_0, x_1 \in \{0,1\}^*}$$

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

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

- $\mathsf{Output}_0^{\Pi} = \left\{\mathcal{C}_0, \left\{\left\langle w^{(k)}\right\rangle_0\right\}_{k\geq 2^{D-1}}\right\}$ where $\mathcal{C}_0[k] = (z_*^{(k)}, u_*^{(k)})$ if $u_*^{(k)} < m_0$ otherwise $\mathcal{C}_0[k] = \perp$.
- $\mathsf{Output}_1^{\Pi} = \left\{\mathcal{C}_1, \left\{\left\langle w^{(k)}\right\rangle_1\right\}_{k\geq 2^{D-1}}\right\}$ where $\mathcal{C}_1[k] = (z_*^{(k)}, u_*^{(k)})$ if $u_*^{(k)} \geq m_0$ otherwise $\mathcal{C}_1[k] = \perp$.

Figure 2: Target GBDT functionality $\mathcal{F}_{\mathrm{GBDT}}$. $\perp$ denotes a special error symbol such that $\perp \neq \{0,1\}^*$.

Definition 1 states that the views of the parties can be properly constructed by a polynomial time algorithm given the party's input and output solely. Also, the parties here are semi-honest and the view is therefore exactly according to the protocol specification.

Composition of Private Protocols. Our *Squirrel* framework is composed of many sub-protocols of smaller private computations. We describe *Squirrel* using the hybrid model to simplify our protocol description and the security proofs. A protocol invoking a functionality $\mathcal{F}$ is said to be in "$\mathcal{F}$-hybrid model". This is similar to a real protocol, except that some sub-protocols are replaced by the invocations of instances of corresponding functionalities. One can consider there is an oracle who computes the corresponding functionalities faithfully in the ideal world. This follows the composition theorem of the semi-honest model [10].

3.2 Private GBDT Training

We focus on a **vertical setting** where the two parties P_0 and P_1 share a different feature space for the same samples. They run a two-party protocol Π to privately implement the GBDT function described in Fig. 1 on the joint dataset. We assume the joint dataset $\mathbf{X}$ contains n samples and m features. We write $\mathbf{X}_l \in \mathbb{R}^{n\times m_l}$ to denote the data owned by the player P_l. The features are vertically distributed between the two parties. We assume P_0's features come before P_1's features, and we also assume the two datasets are already aligned via the common private set intersection technique [20, 24]. That is $\mathbf{X} = \mathbf{X}_0 \| \mathbf{X}_1$. Without loss of generality, we let P_1 hold the whole label vector $\mathbf{y}$.

We note that encrypted computation does not automatically make GBDT private. We now define our private GBDT functionality in Fig. 2. Specifically, the split identifier is only opened to the party who holds the corresponding feature. All the leaf weights are kept in the secret share form. The intermediate values (e.g,. I_k and state in Fig. 1) during the training process are not revealed.

Most of the existing approaches such as [22, 42, 58] also open the split identifiers as part of the output. We reveal no more information than these approaches. The split identifiers may be exploited by attackers. For instance, Zhu et al. [61] quantify the privacy risks associated with publishing decision trees. We think it is an orthogonal problem and could be studied in a separated work.

4 Proposed Squirrel Framework

4.1 Overview

The plain GBDT algorithm described in Fig 1 involves many data-dependent branches, which are not MPC-friendly. We need to convert the plain GBDT algorithm to an oblivious counterpart where the GBDT execution flow is independent with the input data. Specifically, we additionally maintain a (secret) indicator vector $\mathbf{b}^{(k)} \in \{0,1\}^n$ between the two parties P_0 and P_1, for each tree node. That is $\forall i \in [n]$, $\mathbf{b}^{(k)}[i] = 1$ indicates the i-th sample is available on the k-th node, and $\mathbf{b}^{(k)}[i] = 0$ otherwise. We will show how to obliviously update the indicator vector $\mathbf{b}^{(k)}$ between the parties later. The core idea for our private GBDT is to use the following invariant

$$\mathbf{g}^{(k)} = \mathbf{b}^{(k)} \odot \mathbf{g}, \quad \mathbf{h}^{(k)} = \mathbf{b}^{(k)} \odot \mathbf{h}. \tag{2}$$

Then we can aggregate the gradients statistics in (1) using a binary matrix–vector multiplications $\mathbf{M}^{(z)} \cdot \mathbf{g}^{(k)}$. The binary matrix $\mathbf{M}^{(z)} \in \{0,1\}^{B\times n}$ and $\mathbf{M}^{(z)}[u,i] = 1$ means that the i-th sample is categorized into the u-th bin of the z-th feature. The statistics in (1) now can be given as

$$\sum_{i\in I_{2k}^{(z,u)}} \mathbf{g}[i] = \sum_{j\leq u} (\mathbf{M}^{(z)} \cdot \mathbf{g}^{(k)})[j], \; \sum_{i\in I_{2k+1}^{(z,u)}} \mathbf{g}[i] = \sum_{j>u} (\mathbf{M}^{(z)} \cdot \mathbf{g}^{(k)})[j].$$

A similar computation translates to the 2nd order statistics using $\mathbf{M}^{(z)} \cdot \mathbf{h}^{(k)}$. We write $\mathbf{q}^{(k,z)} \in \mathbb{R}^B$ and $\mathbf{p}^{(k,z)} \in \mathbb{R}^B$ to denote product vector $\mathbf{M}^{(z)} \cdot \mathbf{g}^{(k)}$ and $\mathbf{M}^{(z)} \cdot \mathbf{h}^{(k)}$, respectively. Then the score of (1) can be rewritten as

$$\mathcal{G}^{(k,z,u)} = \frac{(\sum_{j\leq u} \mathbf{q}^{(k,z)}[j])^2}{\gamma + \sum_{j\leq u} \mathbf{p}^{(k,z)}[j]} + \frac{(\sum_{j>u} \mathbf{q}^{(k,z)}[j])^2}{\gamma + \sum_{j>u} \mathbf{p}^{(k,z)}[j]} - \frac{(\sum_{j} \mathbf{q}^{(k,z)}[j])^2}{\gamma + \sum_{j} \mathbf{p}^{(k,z)}[j]} \tag{3}$$

There are two kinds of sub-protocols needed to implement our private GBDT algorithm. The ones whose costs grow considerably with the sample size n, and the ones whose costs increase mildly with n. For the latter, we re-use existing protocols for the following functionalities.

- $\langle z\rangle \leftarrow \mathcal{F}_{\mathrm{mul}}(\langle x\rangle, \langle y\rangle)$ [8]. On receiving the shares $\langle x\rangle_l$ and $\langle y\rangle_l$ from each player, outputs $\langle z\rangle_l$ to P_l such that $z \equiv x\cdot y \bmod 2^{\ell}$.
- $\langle z\rangle \leftarrow \mathcal{F}_{\mathrm{recip}}(\langle \tilde{x}\cdot 2^f\rangle; f)$ [12]. On receiving the share $\langle \tilde{x}\cdot 2^f\rangle$ with f-bit fixed-point precision from both players, outputs $\langle z\rangle_l$ to P_l such that $z \equiv \lfloor 2^f/\tilde{x}\rfloor \bmod 2^{\ell}$.

- $\langle b \rangle^B \leftarrow \mathcal{F}_{\text{greater}}(\langle x \rangle, \langle y \rangle)$ [34]. On receiving the shares $\langle x \rangle_l$ and $\langle y \rangle_l$ from each player, outputs $\langle b \rangle_l^B$ to P_l such that $b = \mathbf{1}\{x > y\}$.
- $\langle z \rangle \leftarrow \mathcal{F}_{\text{argmax}}(\{\langle x_i \rangle\}_i)$ [41]. On receiving the shares $\{\langle x_i \rangle_l\}_i$ from each player, outputs $\langle z \rangle_l$ to P_l such that $x_z = \max(x_0, x_1, x_2, \cdots)$.

On the other hand, we design efficient protocols for the more expensive functions needed by GBDT. Specifically, the computation of the first order gradients (e.g., $\mathbf{g} = \mathbf{y} - \sigma(\tilde{\mathbf{y}})$ for the cross-entropy loss) and gradient statistics aggregation (e.g., $\mathbf{M}^z \cdot \mathbf{g}^{(k)}$) are two of the "hot spot" of private GBDT protocols. For example, the sigmoid function and gradient statistics aggregation can respectively take up more than 15% and 63% of the total training time in HEP-XGB. Also, we need an efficient method to maintain the invariant of (2) since we need to compute it for each tree node. Unfortunately, Pivot uses a Paillier-like HE for (2), rendering about $O(n \cdot 2^D)$ HE operations which is extremely expensive.

In the following sections, we present three private protocols for these "hot spot" functions, including a lightweight protocol (§4.2) for (2) using COT, an efficient Seg3Sigmoid protocol (§4.3) for the (approximated) sigmoid function, and a highly optimized protocol BinMatVec (§4.4) for the gradient statistics aggregation using lattice-based HEs.

4.2 To Maintain the Invariant (2) While Keeping the Sample Indicator Secret

In *Squirrel*, the sample indicator vector $\mathbf{b}^{(k)}$ is kept private from P_0 and P_1. To maintain the invariant (2), we let P_l to hold one more vector $\mathbf{b}_l^{(k)} \in \{0,1\}^n$ on each node k. Then we maintain the relation $\mathbf{b}_0^{(k)} \wedge \mathbf{b}_1^{(k)} = \mathbf{b}^{(k)}$ for each node. We observe three advantages of this AND-style relation.

1. P_l can *locally compute* the corresponding $\mathbf{b}_l^{(2k)}$ and $\mathbf{b}_l^{(2k+1)}$ for the two child nodes from $\mathbf{b}_l^{(k)}$.
2. Only one call of $\mathcal{F}_{\text{COT}}$ is needed to compute (2).
3. The gradient statistics aggregation can be accelerated using the AND property. We discuss this later in §4.5.

For the root node, we have $\mathbf{b}_0^{(1)} = \mathbf{b}_1^{(1)} = \mathbf{1}_n$ since all the samples are available on the root node. We now show how to update the indicator $\mathbf{b}^{(k)}$ for its child nodes. The key point for *Squirrel* here is that once the best split identifier $(z_*^{(k)}, u_*^{(k)})$ on the node k has been opened to a party (say P_c), then P_c can update $\mathbf{b}^{(k)}$ locally. To do so, P_c first constructs a vector $\mathbf{b}_*^{(k)} \in \{0,1\}^n$ where $\mathbf{b}_*^{(k)}[i] = 0$ means the $z_*^{(k)}$-th feature of the i-th sample has a larger value than a threshold defined by the $u_*^{(k)}$-th bin. Thus, from the view of P_c, this sample is definitely not in the left child. Other positions of $\mathbf{b}_*^{(k)}$ are set

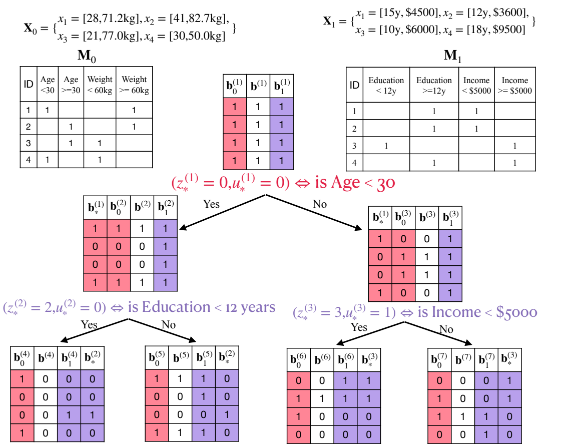


Figure 3: Toy example for indicator update on $n = 4, m_0 = m_1 = 2$ and $B = 2$. P_0 splits the node $k = 1$ on the $z_*^{(1)} = 0$ bin of his $u_*^{(1)} = 0$ feature. P_1 splits the nodes $k = 2$ and $k = 3$. Cells painted in red are known locally by P_0 and cells painted in purple are known locally by P_1.

to 1. The ground sample indicator is $\mathbf{b}^{(2k)} = \mathbf{b}^{(k)} \wedge \mathbf{b}_*^{(k)}$ for the left child, and $\mathbf{b}^{(2k+1)} = \mathbf{b}^{(k)} \oplus \mathbf{b}^{(2k)}$ for the right child.

P_c then locally updates the indicators for child nodes as $\mathbf{b}_c^{(2k)} = \mathbf{b}_c^{(k)} \wedge \mathbf{b}_*^{(k)}$ and $\mathbf{b}_c^{(2k+1)} = \mathbf{b}_c^{(k)} \oplus \mathbf{b}_c^{(2k)}$. On the opposite side, P_{1-c} has no information about the split and thus he keeps the indicators unchanged, i.e., $\mathbf{b}_{1-c}^{(2k)} = \mathbf{b}_{1-c}^{(2k+1)} = \mathbf{b}_{1-c}^{(k)}$. To see the correctness, we let $c = 0$ without loss of generality. The correctness for the two child indicators is given as

$$\begin{aligned}
\mathbf{b}_0^{(2k)} \wedge \mathbf{b}_1^{(2k)} &= (\mathbf{b}_0^{(k)} \wedge \mathbf{b}_*^{(k)}) \wedge \mathbf{b}_1^{(k)} = \mathbf{b}^{(k)} \wedge \mathbf{b}_*^{(k)} = \mathbf{b}^{(2k)}, \\
\mathbf{b}_0^{(2k+1)} \wedge \mathbf{b}_1^{(2k+1)} &= (\mathbf{b}_0^{(k)} \oplus \mathbf{b}_0^{(2k)}) \wedge \mathbf{b}_1^{(2k+1)} \\
&= (\mathbf{b}_0^{(k)} \wedge \mathbf{b}_1^{(2k+1)}) \oplus (\mathbf{b}_0^{(2k)} \wedge \mathbf{b}_1^{(2k+1)}) = \mathbf{b}^{(2k+1)}.
\end{aligned}$$

In Fig. 3, we present a toy example for the indicator update.

The invariant (2) now can be viewed as the product between a private choice vector $\mathbf{b}_*^{(k)}$ and secretly shared vectors as $\mathbf{g}^{(2k)} = \mathbf{b}^{(2k)} \odot \mathbf{g} = (\mathbf{b}^{(k)} \wedge \mathbf{b}_*^{(k)}) \odot \mathbf{g} = \mathbf{b}_*^{(k)} \odot \mathbf{g}^{(k)}$. This can be computed using one instance of $\mathcal{F}_{\text{COT}}$ (see Appendix B.1). A similar computation translates to $\mathbf{h}^{(2k)}$ as $\mathbf{h}^{(2k)} = \mathbf{b}_*^{(k)} \odot \mathbf{h}^{(k)}$.

Privacy. We note that the information $\mathbf{b}_*^{(k)}[i] = 0$ can be directly derived from the split identifier $(z_*^{(k)}, u_*^{(k)})$ which is a part of the protocol output of P_c. Also, the update of child indicators involves local computation only. As a result, the privacy for the indicator update follows trivially in the $\mathcal{F}_{\text{COT}}$ hybrid.

Complexity. The Ferret COT [59] sends about $O(\ell)$ bits per choice on ℓ-bit messages. Thus to maintain the invariant of (2), we need to send about $O(2n\ell)$ bits. To compare, HEP-XGB uses the Beaver's triples [8] to compute the multiplications in (2), and thus about $O(4n\ell)$ bits are exchanged in their method. On the other hand, Pivot uses Paillier HE for (2) by

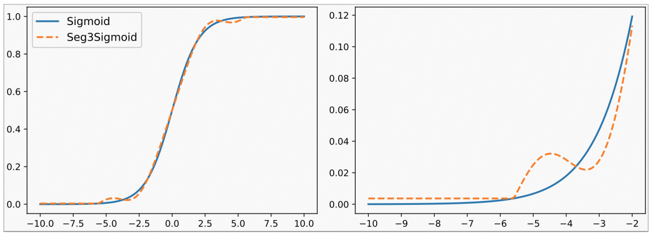


Figure 4: **Left:** We approximate $\sigma(\cdot)$ using three segments. **Right:** We use fixed activation on the margins.

keeping the indicator in the encryption form all the time. As a result, their method sends $O(2n)$ Paillier ciphertexts which is a significantly larger communication than ours.

4.3 Private Two-party Sigmoid Protocol

The exact (or highly accurate) evaluation of the sigmoid function in MPC can be significantly expensive. For example, the OT-based sigmoid protocol used by `Pivot` requires to exchange of about 15.3KB messages *per input*. For the sake of a lower communication overhead, many approximations for the sigmoid are considered [22, 39, 44]. In *Squirrel*, we approximate the sigmoid function by delicately combining numeric approximation with multiple segments. In detail, we numerically approximate the sigmoid function over a limited range, and use two linear pieces to cover the margins. Our approximated sigmoid function is given by three segments:

$$\sigma(x) \approx \mathsf{Seg3Sigmoid}(x) = \begin{cases} \sigma(-\tau) & \text{if } x < -\tau \\ F(x) & \text{if } x \in [-\tau, \tau] \\ \sigma(\tau) & \text{if } x > \tau \end{cases}. \quad (4)$$

The middle segment is a degree-J Fourier series

$$F(x) = \omega_0 + \sum_{j=1}^{J} \omega_j \sin(\frac{2\pi \cdot j \cdot x}{2^{L+1}}), \quad (5)$$

where $L = \lceil \log_2 \tau \rceil$ and $\omega_k \in [-1, 1]$ are some fixed Fourier coefficients. By choosing the proper τ and J parameters, we can approximate the sigmoid within a relatively small error.

In Fig 4, we plot out the absolute difference between the ground truth $\sigma(x)$ and $F(x)$ which is bounded by 2.2×10^{-2} when $\tau = 5.6$ and $J = 9$. One might wonder that the private evaluation of the sine function should be more expensive than the numeric methods [22, 39]. Fortunately, we observe a nice property of the sine function that allows us to evaluate the scaled sine $\sin(2\pi x/2^\delta)$ on the share of $x \in [0, 2^\delta)$ *using the* $\mathcal{F}_{\text{mul}}$ *functionality only*. We first introduce a fraction function $\mathsf{Fr}_\delta(a) = (a \bmod 2^\delta)/2^\delta \in [0, 1)$ to ease the presentation. By the definition of arithmetic share, we know $x = \langle x \rangle_0 + \langle x \rangle_1 + \varepsilon 2^\ell$ (without modulo) for some $\varepsilon \in \{0, 1\}$. Then we have the

Require: $\langle x \rangle$ with f-bit fixed-point precision that is $x = \lfloor \tilde{x} 2^f \rfloor$. Fixed Fourier coefficients $\omega_0, \omega_1, \cdots, \omega_{J-1} \in \mathbb{R}$ and interval parameters $\tau \in \mathbb{R}^+$ and $L \in \mathbb{Z}^+$.
Ensure: The approximated sigmoid $\langle g(\tilde{x}) \rangle$.

1: P_l computes the fraction $\tilde{u}_l = \mathsf{Fr}_{2^{L+1+f}}(\langle x \rangle_l) \in \mathbb{R}$.
2: P_l computes fixed-point values $s_{l,j} = \lfloor \omega_k \cdot \sin(2\pi \cdot j \cdot \tilde{u}_l) \cdot 2^f \rfloor \in \mathbb{Z}_{2^\ell}$ and $c_{l,j} = \lfloor \cos(2\pi \cdot j \cdot \tilde{u}_l) \cdot 2^f \rfloor \in \mathbb{Z}_{2^\ell}$ for $j = 1, \cdots, J-1$.
3: For $j = 1, \cdots, J-1$, jointly compute

$$\langle s_j \rangle \leftarrow \mathcal{F}_{\text{mul}}(s_{0,j}, c_{1,j}) \quad \triangleright s_j \equiv s_{0,j} \cdot c_{1,j} \bmod 2^\ell$$
$$\langle c_j \rangle \leftarrow \mathcal{F}_{\text{mul}}(c_{0,j}, s_{1,j}) \quad \triangleright c_j \equiv c_{0,j} \cdot s_{1,j} \bmod 2^\ell$$

4: P_0 computes $\langle F(\tilde{x}) \rangle_0 = \lfloor \omega_0 \cdot 2^{2f} \rfloor + \sum_{j=1}^{J-1} (\langle s_j \rangle_0 + \langle c_j \rangle_0)$.
5: P_1 computes $\langle F(\tilde{x}) \rangle_1 = \sum_{j=1}^{J-1} (\langle s_j \rangle_1 + \langle c_j \rangle_1)$.
6: Jointly compute two greater-than bits

$$\langle b_{\text{left}} \rangle^B \leftarrow \mathcal{F}_{\text{greater}}(-\tau \cdot 2^f, \langle x \rangle) \quad \triangleright b_{\text{left}} = \mathbf{1}\{-\tau > \tilde{x}\}$$
$$\langle b_{\text{right}} \rangle^B \leftarrow \mathcal{F}_{\text{greater}}(\langle x \rangle, \tau \cdot 2^f) \quad \triangleright b_{\text{right}} = \mathbf{1}\{\tilde{x} > \tau\}$$

P_0 then locally sets $\langle b_{\text{mid}} \rangle_0^B = \langle b_{\text{left}} \rangle_0^B \oplus \langle b_{\text{right}} \rangle_0^B \oplus 1$, and P_1 locally sets $\langle b_{\text{mid}} \rangle_1^B = \langle b_{\text{left}} \rangle_1^B \oplus \langle b_{\text{right}} \rangle_1^B$.
7: Jointly compute the multiplexers using $\mathcal{F}_{\text{COT}}$.

$$\langle b_{\text{left}} \cdot \sigma(-\tau) \rangle \leftarrow \langle b_{\text{left}} \rangle^B, \sigma(-\tau)$$
$$\langle b_{\text{mid}} \cdot F(\tilde{x}) \rangle \leftarrow \langle b_{\text{mid}} \rangle^B, \langle F(\tilde{x}) \rangle$$
$$\langle b_{\text{right}} \cdot \sigma(\tau) \rangle \leftarrow \langle b_{\text{right}} \rangle^B, \sigma(\tau)$$

P_l then aggregates them and outputs as the share of $\langle g(\tilde{x}) \rangle_l$.

Figure 5: Seg3Sigmoid Private sigmoid protocol under the $\mathcal{F}_{\text{mul}}$-, $\mathcal{F}_{\text{greater}}$- and $\mathcal{F}_{\text{COT}}$ hybrid.

following equations.

$$\begin{aligned} \sin(2\pi x/2^\delta) &= \sin(2\pi(\langle x \rangle_0 + \langle x \rangle_1 + \varepsilon 2^\ell)/2^\delta) \\ &= \sin(2\pi \mathsf{Fr}_\delta(\langle x \rangle_0) + 2\pi \mathsf{Fr}_\delta(\langle x \rangle_1)) \\ &= \sin(2\pi \mathsf{Fr}_\delta(\langle x \rangle_0)) \cos(2\pi \mathsf{Fr}_\delta(\langle x \rangle_1)) \\ &\quad + \cos(2\pi \mathsf{Fr}_\delta(\langle x \rangle_0)) \sin(2\pi \mathsf{Fr}_\delta(\langle x \rangle_1)). \end{aligned} \quad (6)$$

The last line comes from the double-angle formula $\sin(x + y) = \sin(x)\cos(y) + \sin(y)\cos(x)$. The insight in (6) is that the values $\sin(2\pi \mathsf{Fr}_\delta(\langle x \rangle_l))$ and $\cos(2\pi \mathsf{Fr}_\delta(\langle x \rangle_l))$ can be computed locally by each party. Also, we can merge the multiplication with the Fourier coefficients $\{\omega_j\}_j$ into (6) since they are known by both parties. In other words, for each shared input $\langle x \rangle$, we can privately evaluate (5) by invoking $2(J-1)$ concurrent calls to $\mathcal{F}_{\text{mul}}$.

We now present our private sigmoid protocol Seg3Sigmoid in Fig. 5. To privately select the activate segment in (4), we access to $\mathcal{F}_{\text{greater}}$. Particularly, given the shares of two greater-than bits $b_{\text{left}} = \mathbf{1}\{-\tau > \tilde{x}\}$ and $b_{\text{right}} = \mathbf{1}\{\tilde{x} > \tau\}$, we can obtain the boolean share of $b_{\text{mid}} = \mathbf{1}\{\tilde{x} \in [-\tau, \tau]\}$ locally via

$b_{\text{mid}} = 1 \oplus b_{\text{left}} \oplus b_{\text{right}}$. Finally, we complete the evaluation as $b_{\text{left}} \cdot \sigma(-\tau) + b_{\text{mid}} \cdot F(x) + b_{\text{right}} \cdot \sigma(\tau)$ using two COTs (see Appendix B.2). Finally, we need one truncation to bring down the fixed-point precision. In practices, when $2f \ll \ell$, we prefer to use the local truncation [46] to avoid extra interactions between the parties. Otherwise, when $2f \approx \ell$, we need to use a faithful truncation protocols such as [34].

Theorem 1 Seg3Sigmoid *in Fig. 5 is a private protocol that implements (4) following the Definition 1 under the* $\mathcal{F}_{\text{mul}}$-, $\mathcal{F}_{\text{greater}}$- *and* $\mathcal{F}_{\text{COT}}$ *hybrid.*

We defer the proof to Appendix C due to space limit.

Complexity. The evaluation of Seg3Sigmoid on **one input** will exchange about $O(8J \log_2 q)$ bits for $\mathcal{F}_{\text{mul}}$, $O(22\ell)$ bits for two $\mathcal{F}_{\text{greater}}$ and $O(6\ell)$ bits for $\mathcal{F}_{\text{COT}}$.

Compared with the existing private sigmoid protocols. To compute the sigmoid, MP-SPDZ [39] uses a Taylor series polynomials to approximate the exponential $\exp(x)$ followed by a division. HEP-XGB [22] uses a numeric approximation $\sigma(x) \approx 0.5 + 0.5 \cdot x/(1+|x|)$. These numeric methods can communicate more messages than Seg3Sigmoid (as shown later in Table 2).

Other private protocols [33, 40] use least-square polynomials to approximate the sigmoid in a limited range. However, these methods are tend to be numerically unstable in the context of MPC computation. For instance, the degree-7 least-square polynomials in [40] contains a coefficient value $1.196 \times 10^{-6} \approx 2^{-20}$ which is expensive to represent using fixed-point values. A lower degree approximation polynomial (e.g., degree-3) can alleviate this issue but will introduce larger errors.

Besides the numeric approximation, piece-wise approximations are also commonly considered in literature. Mohassel et al. [46] use 3 linear segments to approximate the sigmoid at a cost of accuracy loss due to the rough granularity. To avoid a such accuracy loss, Liu et al. [44] suggest using more than 12 linear segments for the sigmoid function which thus needs more calls to $\mathcal{F}_{\text{greater}}$ than ours.

4.4 Efficient Gradient Aggregation Protocol

The gradient aggregation in GBDT involves matrix-vector multiplication where the (data) matrices are binary and locally held in plaintext by the parties in our vertical setting. The multiplication $\mathbf{M} \cdot \mathbf{g}$ given a binary matrix can be achieved via a "pick-then-sum" manner, i.e., $(\mathbf{M} \cdot \mathbf{g})[j] = \sum_{i|\mathbf{M}[j,i]=1} \mathbf{g}[i]$ for each row j. In most of the existing approaches, each element of the gradient vector $\mathbf{g}$ is encrypted separately using HE scheme so that the multiplication $\mathbf{M} \cdot \mathbf{g}$ can be done via homomorphic additions performed by the holder of $\mathbf{M}$. However, the encryption of a long vector using a Paillier-like HE can be extremely expensive.

On the other hand, the RLWE encryption can provide a significantly faster encryption throughput than a Paillier-like HE. In *Squirrel*, the gradient vector $\mathbf{g}$ is first encoded as the coefficients of polynomials before the encryption, e.g., $\mathsf{RLWE}_{\mathsf{pk}}^{N,q,2^\ell}(\mathbf{g}[0] + \mathbf{g}[1]X + \cdots + \mathbf{g}[N-1]X^{N-1})$. When $n > N$, we can use multiple RLWE ciphertext to hold the n coefficients. Then the problem becomes to select a specific encrypted coefficient from RLWE ciphertexts. Indeed, we can extract (or "pick") a specific encrypted coefficient from RLWE *while the extracted ciphertext is a valid LWE ciphertext under the same secret key.* More specifically, given $(\hat{b}, \hat{a}) = \mathsf{RLWE}_{\mathsf{pk}}^{N,q,p}(\hat{m})$ of $\hat{m}$, we use the function $\mathsf{Extract} : \mathbb{A}_{N,q}^2 \times [N] \mapsto \mathbb{Z}_q^{N+1}$ to obtain

$$(b, \mathbf{a}) \in \mathbb{Z}_q^{N+1} \leftarrow \mathsf{Extract}((\hat{b}, \hat{a}), k)$$

such that $b = \hat{b}[k]$ and $\mathbf{a} = [\hat{a}[k], \hat{a}[k-1], \cdots, \hat{a}[0], -\hat{a}[N-1], \cdots, -\hat{a}[k+1]]$. We argue that $(b, \mathbf{a})$ is a valid LWE ciphertext of the k-th coefficient of $\hat{m}$ that is $\hat{m}[k]$ under the key sk. Extract is cheap in terms of computation, e.g., simply reordering the coefficients of $\hat{a}$. We can accelerate Extract using the AVX256 instruction at the cost of $4\times$ RAM consumption.

Fig. 6 depicts our protocol BinMatVec for matrix-vector multiplication protocol on binary matrix. In Step 1, we first convert the secret shares of the vector $\mathbf{g}$ to RLWE ciphertexts using the A2H function (see §2.3.4). Recall that $\langle \mathbf{g} \rangle_l^H$ is RLWE ciphertext(s) of $\mathbf{g}$ held by the party P_l but under P_{1-l}'s key. Then the "pick step" is achieved by extracting the corresponding entry $\mathbf{g}[i]$ from RLWE as an LWE ciphertext

Require: P_{1-l} : $\langle \mathbf{g} \rangle_{1-l} \in \mathbb{Z}_{2^\ell}^n$, key pair $(\mathsf{sk}_{1-l}, \mathsf{pk}_{1-l})$. P_l : $\langle \mathbf{g} \rangle_l \in \mathbb{Z}_{2^\ell}^n$ and binary matrix $\mathbf{M} \in \{0,1\}^{B \cdot m \times n}$. $\boxed{\text{A lifting key LK for LWE dimension lifting.}}$

Ensure: $\langle \mathbf{M} \cdot \mathbf{g} \rangle \in \mathbb{Z}_{2^\ell}^{B \cdot m}$.

1: Jointly run A2H where P_{1-l} acts as sender with input $\langle \mathbf{g} \rangle_{1-l}$ and P_l act as receiver with input $\langle \mathbf{g} \rangle_l$. After the execution, P_l obtains $\langle \mathbf{g} \rangle_l^H$ while P_{1-l} obtains nothing.
2: P_l initializes a $(B \cdot m)$-sized array of LWE encryption of 0, denoted as ct_j for $j \in [B \cdot m]$.
3: **for** all position (j, i) such that $\mathbf{M}[j, i] = 1$ **do**
4:     **[Pick.]** P_l extracts an LWE $\tilde{\mathsf{ct}}_{j,i}$ from $\langle \mathbf{g} \rangle_1^H$ that decrypts to $\mathbf{g}[i]$ under P_{1-l}'s secret key.
5:     **[Sum.]** P_l updates the j-th entry using LWE homomorphic addition: $\mathsf{ct}_j = \mathsf{ct}_j \boxplus \tilde{\mathsf{ct}}_{j,i}$.
6: **end for**
7: $\boxed{\mathsf{ct}_j \leftarrow \mathsf{LWEDimLift}(\mathsf{ct}_j, \mathsf{LK}) \forall j \in [m] \triangleright \text{opt. from §4.5}}$
8: P_l locally merges the LWE ciphertexts as RLWE ciphertexts by $\langle \mathbf{M} \cdot \mathbf{g} \rangle_l^H \leftarrow \mathsf{PackLWEs}(\mathsf{ct}_0, \mathsf{ct}_1, \cdots, \mathsf{ct}_{B \cdot m-1})$.
9: Run H2A jointly where P_l acts as sender with input $\langle \mathbf{M} \cdot \mathbf{g} \rangle_l^H$ and P_{1-l} act as receiver with input sk_{1-l}. Both party P_l outputs what he recevied from H2A as his arithmetic share $\langle \mathbf{M} \cdot \mathbf{g} \rangle_l$.

Figure 6: BinMatVec Private binary matrix-vector multiplication using the H2A and A2H conversions. **NOTE**: The framed parts are run in the optimized version.

using Extract. For instance, the i-th entry $\mathbf{g}[i]$ is encoded as the $(i \bmod N)$-th coefficient of the $(\lfloor i/N \rfloor)$-th RLWE ciphertext. Then the "sum step" is done using LWE homomorphic additions over dimension N. The PackLWEs in Step 8 aims to reduce the ciphertext volume by merging multiple LWEs into a single RLWE, at the cost of some local RLWE computation.

Theorem 2 *If the* RLWE *scheme provides semantic security, then* BinMatVec *in Fig. 6 is a private binary matrix–vector multiplication protocol following the Definition 1.*

We defer the proof to Appendix C due to space limit.

Complexity. For the gradient aggregation step in GBDT, the binary matrix $\mathbf{M}$ consists of *at most* $n \cdot m$ non-zero entries. Thus, we need about $O(nm)$ LWE additions in BinMatVec which translates to $O(nmN)$ machine-word operations. The PackLWEs on $B \cdot m$ LWEs needs about $O(BmN \log_2 N)$ machine-word operations. In terms of communication costs, P_{1-l} sends about $O(n \log_2 q)$ bits in A2H, and P_l sends about $O(2Bm \log_2 q)$ bits in H2A.

Compared with SIMD- and COT- based methods. Besides the coefficient encoding used in *Squirrel*, the homomorphic Single-Instruction-Multiple-Data (SIMD) technique [54] is another common encoding to transform vectors as the elements of the ring $\mathbb{A}_{N,p}$. However, for SIMD, the "pick" step requires a homomorphic multiplication with a binary vector (i.e., only one non-zero in the selected position), followed by an expensive homomorphic rotation (e.g., 500× more expensive than the LWE addition) to align the position of the selected slot before doing the "sum" step.

We can also use COT to compute the matrix multiplication on binary matrix via [5]. However, the communication complexity of the COT-based method is quadratic on the matrix size, i.e., $O(Bmn\ell)$ bits of communication. This is a significantly larger overhead than that of BinMatVec protocol, particularly when millions of samples n are considered.

4.5 Further Optimizations

We now propose two orthogonal optimizations to further accelerate BinMatVec. One can reduce the number of LWE additions needed in BinMatVec. The other can lower down the concrete cost of each LWE addition. Indeed, the end-to-end *Squirrel* training time can be reduced by about 60% – 70% when using the optimized BinMatVec.

4.5.1 To Use the Indicator Sparsity

The computation cost of BinMatVec in Fig. 6 majorly depends on the number of non-zero entries of the matrix $\mathbf{M}$. We can leverage the sparsity of the indicator $\mathbf{b}^{(k)}$ to reduce the number LWE additions. In stead of $\mathbf{M} \cdot \mathbf{g}^{(k)}$, we prefer to compute the identical $(\mathbf{M} \cdot \mathbf{b}_l^{(k)}) \cdot \mathbf{g}^{(k)}$. That is because we have $\mathbf{b}_l^{(k)}[i] = 0 \Rightarrow \mathbf{g}^{(k)}[i] = 0\ \forall i \in [n]$ according to our definition of the sample indicator $\mathbf{b}^{(k)} = \mathbf{b}_0^{(k)} \wedge \mathbf{b}_1^{(k)}$.

4.5.2 To Use a Smaller Lattice Dimension

For the case that the number of samples $n \gg B \cdot m$, the 'pick-then-sum' step will dominate the running time of Fig. 6. This motivates us to have a cheaper 'pick-then-sum' by instantiating A2H with a smaller lattice dimension N. However, the following PackLWEs step (usually) requires a moderate dimension, e.g., $N \geq 8192$. One might consider to skip the PackLWEs step, and use LWE ciphertexts for the H2A conversion. However, the PackLWEs is crucial for a small communication overhead. For example, to convert 10^4 shares using LWE, P_l might need to send about 532MB LWE ciphertexts to P_{1-l} for our RLWE parameters. On the other hand, P_l sends only 436KB of RLWE ciphertexts if he apply PackLWEs to merge the LWEs into the RLWE form first.

To take advantage of the low communication overhead from PackLWEs while keeping a cheap LWE addition, we propose a specific LWE-to-LWE conversion that inspired by [14]. Indeed, our LWE-to-LWE conversion can help to reduce about half of the running time of Fig. 6 when $n \gg B \cdot m$. . Intuitively, we instantiate A2H using a small lattice dimension $\underline{N}$ (e.g., $\underline{N} \leq 4096$) and then perform the "pick-then-sum" under this dimension. Before doing PackLWEs, we first apply an LWEDimLift procedure to convert a given LWE ciphertext of dimension $\underline{N}$ to a larger lattice dimension (e.g., $N \geq 8192$) without changing the encrypted message (i.e., Step 7 of Fig. 6). Then the following PackLWEs and H2A are still performed over the larger dimension N.

We first define two helper functions for the following descriptions. The first one is a lifting function $\mathsf{lift} : \mathbb{A}_{\underline{N},\underline{q}} \mapsto \mathbb{A}_{N,q}$, $\mathsf{lift}(\hat{s}) = q' \cdot (\hat{s}[0] - \sum_{i=1}^{N-1} \hat{s}[i] X^{N-i}) \bmod q$, where $q' = q/\underline{q} \in \mathbb{Z}$. The other one is a gadget decomposition $\mathbf{g}_{\mathrm{gdt}}^{-1} : \mathbb{Z}_q \mapsto \mathbb{Z}^W$, parameterized by a vector $\mathbf{g}_{\mathrm{gdt}} \in \mathbb{Z}^W$, satisfying $\left\langle \mathbf{g}_{\mathrm{gdt}}^{-1}(a), \mathbf{g}_{\mathrm{gdt}} \right\rangle \equiv a \bmod q$ for all $a \in \mathbb{Z}_q$. To achieve the LWE dimension lifting, we need a lifting key $\mathsf{LK}_{\underline{\mathsf{sk}} \to \mathsf{sk}}$ which is given as an array of W RLWE ciphertexts. Specially, the d-th entry is given as $(\mathbf{g}_{\mathrm{gdt}}[d] \cdot \mathsf{lift}(\underline{\mathsf{sk}}) - \hat{\alpha}_d \cdot \mathsf{sk} + \hat{e}_d, \hat{\alpha}_d) \in \mathbb{A}_{N,q}^2$, where the coefficients of $\hat{\alpha}_d, \hat{e}_d \in \mathbb{A}_{N,q}$ are sampled using the same distributions in the public key generation.

We now present our LWE dimension lifting procedure LWEDimLift in Fig. 8. The computation in Fig. 8 can be seen as a Key-Switching in many RLWE schemes [14, 18, 53] with an extra factor q'. Most of the computation lies on the inner product between the vector of polynomials $\mathbf{g}_{\mathrm{gdt}}^{-1}(\hat{a})$ and the lifting key, requiring about $O(WN \log_2 N)$ machine-word operations. The correctness of Fig. 8 is deferred to Appendix A.

4.6 Putting Everything Together

We describe how the actual *Squirrel* work for building one GBDT tree in Fig. 7. Here we prefer a full and balanced tree for the sake of simplicity. That is a tree node with an index $k \geq 2^{D-1}$ is a leaf. We assume the 1st and 2nd order gradients

Require: $\mathsf{Input}_0 = \{\mathbf{X}_0 \in \mathbb{R}^{n\times m_0}\}$ and $\mathsf{Input}_1 = \{\mathbf{X}_1 \in \mathbb{R}^{n\times m_1}, \mathbf{y}\}$. Secretly shared stateful values $\mathsf{state} = \{\langle \mathbf{g}\rangle_l, \langle \mathbf{h}\rangle_l, \langle \tilde{\mathbf{y}}\rangle_l\}$. Publicly known values $\mathsf{pp} = \{D > 0, B > 0, \gamma > 0, (\underline{\mathsf{sk}}_l, \underline{\mathsf{pk}}_l)$ for A2H, $(\mathsf{sk}_l, \mathsf{pk}_l)$ for H2A, and the LWE lifting key $\mathsf{LK}_{\underline{\mathsf{sk}}_l \to \mathsf{sk}_l}\}$.

Ensure: Output_0^{Π} and Output_1^{Π} (See the definition in § 3.2)

1: P_l locally partitions his data $\mathbf{X}_l$ into bins, written as $\mathbf{M}_l \in \{0,1\}^{B\cdot m_l \times n}$ where $\mathbf{M}_l[B\cdot z + u, i] = 1$ indicates that the i-th sample is categorized into the u-th bin according to its z-th feature for $i \in [n]$, $z \in [m_l]$ and $u \in [B]$.

2: P_l sets the indicator $\mathbf{b}_l^{(1)} = \mathbf{1}_n$ with all 1s, and set $\left\langle \mathbf{g}^{(1)} \right\rangle_l = \langle \mathbf{g}\rangle_l$ and $\left\langle \mathbf{h}^{(1)} \right\rangle_l = \langle \mathbf{h}\rangle_l$. ▷ All samples are on the root node.

3: **for** internal nodes $k = 1, 2, \cdots, 2^{D-1} - 1$ **do**

4: [**Computing Partition Scores.**] If $k = 1, 2, 4, \cdots, 2^{D-2}$ is a left node, then jointly run two optimized $\mathsf{BinMatVec}$ where P_1 acts as the matrix holder. $\left\langle \mathbf{p}^{(k,1)} \right\rangle, \left\langle \mathbf{q}^{(k,1)} \right\rangle \leftarrow \mathsf{BinMatVec}\left(\left\{\left\langle \mathbf{g}^{(k)} \right\rangle_0, \left\langle \mathbf{h}^{(k)} \right\rangle_0, \mathsf{sk}_0\right\}, \left\{\left\langle \mathbf{g}^{(k)} \right\rangle_1, \left\langle \mathbf{h}^{(k)} \right\rangle_1, \mathbf{M}_1 \cdot \mathrm{diag}(\mathbf{b}_1^{(k)})\right\}\right)$. Flip the role of matrix holder, and run two more $\mathsf{BinMatVec}$ simultaneously to obtain $\left\langle \mathbf{p}^{(k,0)} \right\rangle$ and $\left\langle \mathbf{q}^{(k,0)} \right\rangle$.

5: Locally concatenate the shares $\left\langle \mathbf{p}^{(k)} \right\rangle = \left\langle \mathbf{p}^{(k,0)} \right\rangle \| \left\langle \mathbf{p}^{(k,1)} \right\rangle$ and $\left\langle \mathbf{q}^{(k)} \right\rangle = \left\langle \mathbf{q}^{(k,0)} \right\rangle \| \left\langle \mathbf{q}^{(k,1)} \right\rangle$.

6: Otherwise, $k = 3, 5, \cdots, 2^{D-1} - 1$ is a right node, locally set $\left\langle \mathbf{p}^{(k)} \right\rangle = \left\langle \mathbf{p}^{(k/2)} \right\rangle - \left\langle \mathbf{p}^{(k-1)} \right\rangle$ and $\left\langle \mathbf{q}^{(k)} \right\rangle = \left\langle \mathbf{q}^{(k/2)} \right\rangle - \left\langle \mathbf{q}^{(k-1)} \right\rangle$.

7: Jointly compute the all the partition scores $\left\langle \mathcal{G}^{(k,z,u)} \right\rangle$ for $z \in [m_0 + m_1]$ and $u \in [B]$, according to (3) using $\mathcal{F}_{\mathrm{recip}}$ and $\mathcal{F}_{\mathrm{mul}}$.

8: [**Find the Best Split with Maximum Score.**] Jointly compute $\left\langle z_*^{(k)} \right\rangle, \left\langle u_*^{(k)} \right\rangle \leftarrow \mathcal{F}_{\mathrm{argmax}}^{z,u}(\{\left\langle \mathcal{G}^{(k,z,u)} \right\rangle\}_{z,u})$.

9: Open a bit $\langle c\rangle^B \leftarrow \mathcal{F}_{\mathrm{greater}}(\left\langle z_*^{(k)} \right\rangle, m_0 - 1)$ to both players.

10: Open the split identifier $(z_*^{(k)}, u_*^{(k)})$ to P_c who then writes them into $\mathcal{C}_c[k]$ while P_{1-c} writes $\perp$ to $\mathcal{C}_{1-c}[k]$.

11: [**Locally Update Sample Indicator.**] P_{1-c} keeps the indicators unchanged $\mathbf{b}_{1-c}^{(2k)} = \mathbf{b}_{1-c}^{(2k+1)} = \mathbf{b}_{1-c}^{(k)}$.

12: P_c locally updates $\mathbf{b}_c^{(2k)} = \mathbf{b}_c^{(k)} \wedge \mathbf{b}_*^{(k)}$ and $\mathbf{b}_c^{(2k+1)} = \mathbf{b}_c^{(k)} \oplus \mathbf{b}_c^{(2k)}$ where $\mathbf{b}_*^{(k)}[i] = \mathbf{1}\left\{\sum_{u \le u_*^{(k)}} \mathbf{M}_l[z_*^{(k)} \cdot B + u, i] > 0\right\}$ for $i \in [n]$.

13: [**Maintain the Invariant of (2).**] Jointly compute $\left\langle \mathbf{g}^{(2k)} \right\rangle = \left\langle \mathbf{b}_*^{(k)} \odot \mathbf{g}^{(k)} \right\rangle$ and $\left\langle \mathbf{h}^{(2k)} \right\rangle = \left\langle \mathbf{b}_*^{(k)} \odot \mathbf{h}^{(k)} \right\rangle$ using two $\mathcal{F}_{\mathrm{COT}}$ instances where P_c acts as the receiver with choice bits $\mathbf{b}^*$, and P_{1-c} provides the correlations $\mathbf{x} + \left\langle \mathbf{g}^{(k)} \right\rangle_{1-c}$ and $\mathbf{x} + \left\langle \mathbf{h}^{(k)} \right\rangle_{1-c}$.

14: Locally sets $\left\langle \mathbf{g}^{(2k+1)} \right\rangle = \left\langle \mathbf{g}^{(k)} \right\rangle - \left\langle \mathbf{g}^{(2k)} \right\rangle$ and $\left\langle \mathbf{h}^{(2k+1)} \right\rangle = \left\langle \mathbf{h}^{(k)} \right\rangle - \left\langle \mathbf{h}^{(2k)} \right\rangle$.

15: **end for**

16: **for** leaf nodes $k = 2^{D-1}, \cdots, 2^D - 1$ **do**

17: Jointly compute the weight $\left\langle w^{(k)} = -\sum_i \mathbf{g}^{(k)}[i] / (\lambda + \sum_i \mathbf{h}^{(k)}[i]) \right\rangle$ using $\mathcal{F}_{\mathrm{recip}}$ and $\mathcal{F}_{\mathrm{mul}}$.

18: Update the prediction scores in the state list $\langle \tilde{\mathbf{y}}\rangle \leftarrow \left\langle \sum_{k=2^{D-1}}^{2^D-1} (\mathbf{b}_0^{(k)} \wedge \mathbf{b}_1^{(k)}) \cdot w^{(k)} \right\rangle + \langle \tilde{\mathbf{y}}\rangle$ using two concurrent $\mathcal{F}_{\mathrm{COT}}$ instances.

19: P_l writes his share $\left\langle w^{(k)} \right\rangle_l$ into Output_l^{Π}.

20: **end for**

Figure 7: *Squirrel* Private GBDT Training Framework under $\mathcal{F}_{\mathrm{mul}}$-, $\mathcal{F}_{\mathrm{recip}}$-, $\mathcal{F}_{\mathrm{argmax}}$-, $\mathcal{F}_{\mathrm{greater}}$-, and $\mathcal{F}_{\mathrm{COT}}$ hybrid.

have been computed privately and given as an input in Fig. 7. This allows us to directly reuse the specification of Fig. 7 for both classification and regression tasks. For instance, for a binary classification task using the cross-entropy loss, we can use our $\mathsf{Seg3Sigmoid}$ protocol to privately compute the 1st order gradient like $\langle \mathbf{g}\rangle \leftarrow \mathsf{Seg3Sigmoid}(\langle \tilde{\mathbf{y}}\rangle) - \mathbf{y}$, and then compute the 2nd order gradient $\langle \mathbf{h}\rangle$ using $\mathcal{F}_{\mathrm{mul}}$. For a linear regression task using the least squares loss, the gradients are $\mathbf{g} = \tilde{\mathbf{y}} - \mathbf{y}$.

To use the optimizations of §4.5, we require each player to generate two key pairs $(\mathsf{sk}_l, \mathsf{pk}_l)$ and $(\underline{\mathsf{sk}}_l, \underline{\mathsf{pk}}_l)$ for two sets of HE parameters (N, q) and $(\underline{N}, \underline{q})$, respectively. Also, each player needs to provide the corresponding lifting key. All of these HE materials are already stored into the public list pp.

In Step 1, each party P_l first locally converts his data $\mathbf{X}_l$ to a binary matrix $\mathbf{M}_l \in \{0,1\}^{B\cdot m_l \times n}$. The gradient statistics $\mathbf{p}^{(k)} = \mathbf{M}_0 \cdot \mathbf{g}^{(k)} \| \mathbf{M}_1 \cdot \mathbf{g}^{(k)}$ now are computed by two concurrent executions of $\mathsf{BinMatVec}$ in Step 4. Another two $\mathsf{BinMatVec}$ for the 2nd order gradient statistics $\mathbf{q}^{(k)}$ can be run concurrently. Also, we apply the histogram subtraction trick in Step 6, e.g., $\mathbf{p}^{(k)} = \mathbf{p}^{(2k)} + \mathbf{p}^{(2k+1)}$, which is commonly used in the plain GBDT training. Then the two parties jointly compute the all the partition scores $\{G^{(k,z,u)}\}_{z\in[m],u\in[B]}$ from the shared statistics $\left\langle \mathbf{p}^{(k)} \right\rangle$ and $\left\langle \mathbf{q}^{(k)} \right\rangle$ according to (3) using $\mathcal{F}_{\mathrm{recip}}$ and $\mathcal{F}_{\mathrm{mul}}$. Finally the split identifier is determined using $\mathcal{F}_{\mathrm{argmax}}$.

We do not open the split identifier $(z_*^{(k)}, u_*^{(k)})$ to both parties which will invalidate our definition in Fig. 2. We first invoke $\mathcal{F}_{\mathrm{greater}}$ to compute the bit $c = \mathbf{1}(z_*^{(k)} \ge m_0)$ where it indicates that the chosen feature $z_*^{(k)}$ belongs to the party P_c. We then let P_{1-c} send his share of the split identifier to P_c for opening. Step 11 to Step 13 follow the descriptions in §4.2. Briefly, P_c updates the gradient vectors for the left child according to

Require: LWE ciphertext $(\underline{b}, \underline{\mathbf{a}}) \in \mathbb{Z}_{\underline{q}}^{N+1}$ which decrypts to $m \in \mathbb{Z}_p$ under a secret key $\underline{\mathsf{sk}}$. A lifting key $\mathsf{LK}_{\underline{\mathsf{sk}} \to \mathsf{sk}}$ consists of W-many $\mathsf{RLWE}^{N,q}$ ciphertexts under under another key sk. Also q is divisible by $\underline{q}$ and let $q' = q/\underline{q}$.

Ensure: LWE ciphertext $(b, \mathbf{a}) \in \mathbb{Z}_q^{N+1}$ that decrypts to the identical message m under the key sk.

Convert $\underline{\mathbf{a}}$ as a polynomial $\hat{a} = \sum_{i=0}^{N-1} \underline{\mathbf{a}}[i] X^i \in \mathbb{A}_{N,q}$.

Compute $\mathsf{CT} = (q' \cdot \underline{b}, 0) + \left\langle \mathbf{g}_{\mathrm{gdt}}^{-1}(\hat{a}), \mathsf{LK}_{\underline{\mathsf{sk}} \to \mathsf{sk}} \right\rangle \in \mathbb{A}_{N,q}^2$ where $\mathbf{g}_{\mathrm{gdt}}^{-1}$ is carried out to each coefficient of $\hat{a}$.

Output an LWE via $\mathsf{Extract}(\mathsf{CT}, 0)$.

Figure 8: LWE Dimension Lifting Procedure $\mathsf{LWEDimLift}$

the revealed split identifier $(z_*^{(k)}, u_*^{(k)})$ and his private data $\mathbf{M}_c$ using $\mathcal{F}_{\mathrm{COT}}$. Finally, leaf weight is privately computing and the prediction vector $\tilde{\mathbf{y}}$ is privately updated using the ideal functionalities in Step 17 and Step 18.

Theorem 3 *The protocol of Fig 7 is a private* GBDT *of Fig. 1 under* $\mathcal{F}_{\mathrm{mul}}$- $\mathcal{F}_{\mathrm{recip}}$- $\mathcal{F}_{\mathrm{argmax}}$-, $\mathcal{F}_{\mathrm{greater}}$-, *and* $\mathcal{F}_{\mathrm{COT}}$- *hybrid, and assuming the* RLWE *scheme providing semantic security.*

We defer the proof to Appendix C due to space limit.

Secure Inference. Once the GBDT model is trained, the secure inference can also be done using COTs. On a new input sample, each player can *locally prepare* a binary vector β_l for each tree. Specially, $\beta_l[k] = 0$ means the input **will not** be classified to the k-th leaf according to the split identifier(s) held by P_l. That is, from the view of P_l, the new sample invalidates at least one split identifier along the prediction path to the k-th leaf. Also $\beta_l[k] = 1$ means the new sample **might be** classified to the leaf k. At the end, there is only one non-zero entry in $\beta_0 \wedge \beta_1$. Then the inference on the t-th tree is to compute $\sum_k (\beta_0[k] \wedge \beta_1[k]) \cdot w^{(k)}$ given the shares of the leaf weights. This can be evaluated using two COTs (see Appendix B.3).

5 Evaluations

5.1 Evaluations Setup

Concrete Parameters. We use $\ell = 64$ bits arithmetic sharing and $f = 16$ for fixed-point values. For our GBDT training, we fix $\gamma = 0.001$. We use the `Ferret` implementation from the EMP toolkit [57]. Our RLWE/LWE implementation is built on top of the SEAL library [53] with the AVX acceleration [36]. We instantiate two SEAL parameters $\mathsf{HE.pp} = \{p = 2^\ell, (\underline{N} = 4096, \underline{q} \approx 2^{109}), (N = 8192, q = \underline{q})\}$. The security levels under these parameters are at least 128-bit security according to [6]. Under these parameters, we can run *Squirrel* for more than 5×10^{16} samples without a decryption failure w.h.p. (see the noise analysis in Appendix). We implement [49] for $\mathcal{F}_{\mathrm{mul}}$ using RLWE, and [12] for $\mathcal{F}_{\mathrm{recip}}$ using OT. We also reuse the OT-based implementation from [35] for $\mathcal{F}_{\mathrm{argmax}}$ and $\mathcal{F}_{\mathrm{greater}}$. Our lifting key $\mathsf{LK}_{\underline{\mathsf{sk}_l} \to \mathsf{sk}_l}$ is about 0.37 MB. The public key pk_l, including the materials for $\mathsf{PackLWEs}$ is about 5.06 MB.

Table 1: Microbenchmarks of *Squirrel* (single thread).

Interactive	End-to-End Time LAN	WAN	Commu. (MB)
A2H (10^6 inputs)	0.5s	0.9s	14.5
H2A (2^{13} inputs)	16.9ms	87.2ms	0.22
Seg3Sigmoid (10^4 inputs)	0.42s	0.9s	15.9
$\mathcal{F}_{\mathrm{recip}}$ (10^5 inputs)	3.8s	9.2s	178.9
$\mathcal{F}_{\mathrm{COT}}$ (10^6 inputs)	85.3ms	370.6ms	7.7
$\mathcal{F}_{\mathrm{mul}}$ (10^6 inputs)	1.4s	5.1s	106.1
$\mathcal{F}_{\mathrm{greater}}$ (10^6 inputs)	2.2s	4.1s	86.4
Local	**Throughput (#op. per second)**		
pick-then-sum ($\underline{N} = 4096$)	6.17×10^5		
pick-then-sum ($N = 8192$)	3.14×10^5		
LWEDimLift	1176.5		
PackLWEs (128 inputs)	43.1 (8 threads)		
PackLWEs (512 inputs)	14.3 (8 threads)		

Testbed Environment. Our programs are implemented in C++ and compiled by gcc-8.5.0. All the following experiments are performed on commercial cloud instances with a 2.7GHz processor and 32GB of RAM. The bandwidth between the cloud instances is manipulated with the traffic control command of Linux. We run the benchmarks mainly in two network settings including a LAN (1Gbps with 2ms ping time) and a WAN (200Mbps with 20ms ping time).

Metrics. We measure the end-to-end running time including the time of transferring HE/OT ciphertexts through the network. We measure the total communication including all the messages sent by the two parties.

5.2 Microbenchmarks

In Table 1, we present the performance of the underlying primitives used by *Squirrel*. Particularly, we categorize the primitives into *interactive primitives* that require to exchange of messages through the network, and *local primitives* that are performed locally by one party.

Our RLWE-based share conversions are very efficient in terms of computation time and communication size, handling millions of shared values in seconds. To compare, we also benchmark the A2H conversions in `Pivot` and `HEP-XGB` in Table 2 using the public libraries [9, 56] for the Paillier and OU scheme. In brief, our share conversions can be about 3 orders of magnitude faster than their approaches.

For the sigmoid function, the numeric approximated sigmoid in `HEP-XGB` computes $\sigma(x) \approx 0.5 + 0.5x/(1+|x|)$, requiring to invoke $\mathcal{F}_{\mathrm{recip}}$ on each input point. Thus, their nu-

Table 2: Compare ours A2H to the existing methods used in `Pivot` and `HEP-XGB`. Paillier and OU are instantiated using 1024-bit keys. We also compare to a recent FSS-based secure sigmoid [4]. Single thread is used for the comparisons.

A2H (10^6 inputs)	[58] (Paillier)	[22] (OU)	Ours
Time (LAN)	$\approx$ 2000s	$\approx$ 800s	0.5s
Commu.	244MB	122MB	14.5MB
Sigmoid (10^4 inputs)	**[39] (OT)**	**[4] (FSS)**	**Ours**
Time (WAN)	8.9s	176.8s	0.9s
Commu.	150MB	11.5MB	15.9MB

meric method will be less efficient than Seg3Sigmoid when we implement their method using 2PC without the aid of TEE. Also from Table 2, we can see that the communication overhead of Seg3Sigmoid is significantly smaller than the method used in `Pivot`, i.e., the MP-SPDZ library [39]. The very recent work from Agarwal et al. [4] present a secure sigmoid protocol using Function Secret Sharing (FSS) as the building block. Their method transfers about 27% less messages than Seg3Sigmoid but takes about 200$\times$ more computation time.

We can also see that the COT is an efficient choice for (2), in terms of communication costs. Recall that `HEP-XGB` maintains (2) using an TEE-based $\mathcal{F}_{\mathrm{mul}}$ which will double the communication than our COT-based solution.

Prediction Efficiency. We can derive the prediction efficiency of *Squirrel* from Table 1. Assuming there are 1600 leaves (e.g., 100 trees with $D = 5$ depth) in the trained GBDT model, we need two COTs on 1600 inputs for the inferences. Thus we estimate the inferences throughput of *Squirrel* over LAN is about $1000\text{ms}/(2 \cdot 85.3\text{ms}) \cdot 10^6/1600 \approx 3.6 \times 10^3$ inferences per second. To compare, `Pivot` reports about 100 inferences per second [58, Figure 4g].

To demonstrate the effectiveness of our LWE dimension lifting optimization, we also benchmark the throughput of "pick-then-sum" under $N = 8192$. For any (n, m) pair, the time for the gradient aggregation under the parameter $N = 8192$ is about $\text{time}_N = 2nm/3.14 \times 10^5$ seconds without LWE lifting. When applying LWE lifting, the time becomes about $\text{time}_{\underline{N}} = 2nm/6.17 \times 10^5 + 2m/1176.5$ seconds. With some simple calculations, we know that the LWE dimension lifting optimization can reduce the aggregation time, i.e., $\text{time}_{\underline{N}} < \text{time}_N$, when we have $n \geq 544$ samples.

5.3 Compare with the Existing Frameworks

5.3.1 Efficiency Comparison

In Table 3, we compare the performance of *Squirrel* with two existing MPC frameworks for GBDT training. The timing numbers in the table are taken or derived from the cited papers. For example, `Pivot` reports 11.2 minutes for training 4 boosting trees between two parties. Then we compare with the average time 168 seconds for `Pivot` in Table 3. The performance of these existing frameworks are measured on different parameters and network settings. We run *Squirrel* on a similar setting with best efforts for the comparison. For example, [22] use more than 32 cores for the computation while our testbed can provide only 8 cores. The authors of `Pivot` claim to use a LAN but the specific bandwidth and latency were missing in their paper. It is worthy to note that all these existing approaches use short HE keys, resulting a lower security level than the 128-bit security level of *Squirrel*.

Table 3: Performance comparison of *Squirrel* with the existing privacy-preserving GBDT approaches. For each run, we measure the end-to-end running time *per tree*.

Approach	Parameters	Settings	Time
Ours	$n = 5 \times 10^4, D = 4$	LAN	6.0s
[58]†	$m_0 = 8, m_1 = 7, B = 8$	6 threads	168s (28$\times$)
Ours	$\mathbf{n = 2 \times 10^5}, D = 4$	LAN	11.1s
[58]	$m_0 = 8, m_1 = 7, B = 8$	6 threads	448s (40$\times$)
Ours	$n = 1.4 \times 10^5, D = 5$	LAN	11.4s
[22]‡	$m_0 = 7, m_1 = 16, B = 10$	32 threads	47.6s (4$\times$)
Ours	$n = 1.4 \times 10^5, D = 5$	**100 Mbps**	40.0s
[22]	$m_0 = 7, m_1 = 16, B = 10$	32 threads	151s (3.7$\times$)

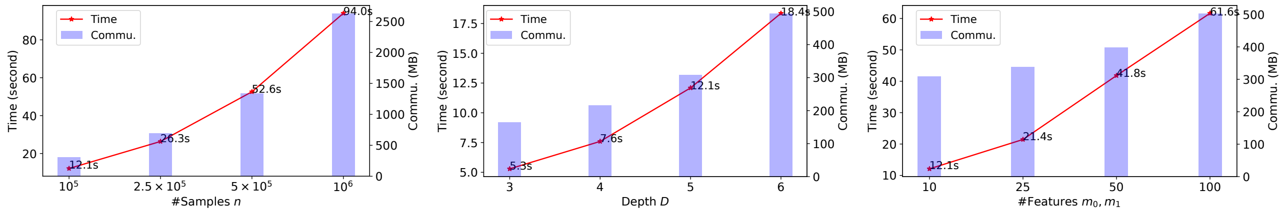
†`Pivot` [58] did not report the pre-processing time.
‡`HEP-XGB` [22] have used TEE to accelerate their computation.

Table 4: F1-score comparison with (simulated) `Pivot` and `HEP-XGB` on 6 datasets, using $T = 10$ trees of depth $D = 5$.

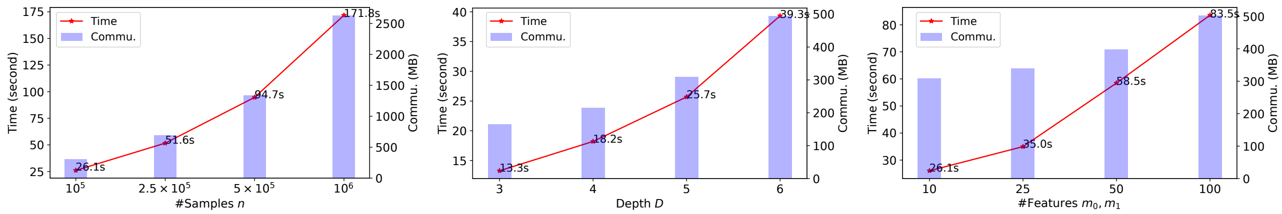
Dataset	(n/m)	*Squirrel*	`Pivot`	`HEP-XGB`
breast-cancer	683/9	0.917	0.918	0.889
phishing	11055/ 67	0.957	0.957	0.951
a9a	32561/122	0.651	0.653	0.643
cod-rna	59535/7	0.402	0.408	0.403
skin_nonskin	245057/2	0.742	0.743	0.741
covtype	581012/53	0.556	0.572	0.552

Squirrel is still 28$\times$ – 40$\times$ faster than `Pivot` even we have omitted the heavy pre-processing time in `Pivot`. Under the LAN setting, most of the running time of `Pivot` due to the expensive Paillier operations. It is no doubt that the performance advantages of *Squirrel* over `Pivot` can be even larger if we align the security level of `Pivot`'s HE parameters to a 3072-bit public key.

`HEP-XGB` [22] depends on a trusted hardware to accelerate some parts of their building blocks. For instance, the $\mathcal{F}_{\mathrm{mul}}$, $\mathcal{F}_{\mathrm{argmax}}$ and $\mathcal{F}_{\mathrm{recip}}$ operations in `HEP-XGB` are "almost free" in the LAN setting while *Squirrel* evaluates 2PC protocols for these operations. The computation overheads of their HE are



(a) Time (LAN) vs. Sample size n (b) Time (LAN) vs. Depth D (c) Time (LAN) vs. Feature size m_0, m_1



(d) Time (WAN) vs. Sample size n (e) Time (WAN) vs. Depth D (f) Time (WAN) vs. Feature size m_0, m_1

Figure 9: Effect of parameters in *Squirrel*. By default, we set $n = 10^5$, $m_0 = m_1 = 10$, $D = 5$, $B = 16$ and use 8 threads.

still too large, even they replace the Paillier scheme with a faster Okamoto-Uchiyama scheme as the alternative.

5.3.2 Effectiveness Comparison

We empirically show the effectiveness of *Squirrel* on 6 real-world datasets taken from [2]. All of them have two classes to classify, and contain more samples n than features m, without missing value. For each dataset, we apply 5-fold cross validation and report the average F1-scores on the validation sets. Specifically, we train $T = 10$ trees of depth $D = 5$. The results are given in Table 4.

To compare, we simulate the training using `Pivot` and `HEP-XGB`. Particularly, we use the exact sigmoid in our simulation for `Pivot`, and use the approximation $0.5 + 0.5x/(1+|x|)$ for the sigmoid function in the simulation of `HEP-XGB`. The hyper-parameters and initialization (e.g., $\tilde{\mathbf{y}}^{(0)}$) are kept identical for *Squirrel* and the simulated `Pivot` and `HEP-XGB`. We can see that the GBDT model trained by *Squirrel* is effective, giving a comparable prediction accuracy to `Pivot` which is identical to the GBDT training on the plain fixed-point values. We also observe that `HEP-XGB` converges slower than `Pivot` and *Squirrel*. The GBDT model trained by `HEP-XGB` can finally achieve a similar accuracy of `Pivot` when using more trees.

5.4 Scalability Test on Synthetic Data

To demonstrate the scalability of *Squirrel*, we train *one boosting tree* on various sets of parameters. Specifically, we conduct experiments by varying the number of samples (n), the maximum tree depth (D) and the number of features of held by each player (m_0, m_1). We apply all the proposed optimizations, and use multi-threading as much as possible.

The efficiency evaluation is given in Figure 9. The running time and communication of *Squirrel* increases linearly with n and D. We observe, by doubling n (or increasing D by 1), the running time of *Squirrel* will not be doubled. The reason is that, we leverage the sparsity of the AND-style sharing for the indicator vector to reduce the number of LWE additions. On the other hand, the total communication increases more gently with the number of features by the virtue of the low communication overheads from the `Ferret` OT and PackLWEs. For example, by increasing the number of features from 10 to 100, the communication only increases by 67% from about 300MB to 500MB.

Conclusion

We have proposed *Squirrel*, a scalable secure two-party computation framework for training Gradient Boosting Decision Tree. *Squirrel* guarantees that no intermediate information is disclosed during the training without any dependency on trusted hardware. *Squirrel* is accurate, achieving accuracy comparable to the non-private baseline. Also, our empirical results demonstrate that *Squirrel* is scalable to large-scale datasets with millions of samples even under WAN.

Acknowledgment. The authors would like to thank the anonymous reviewers for their insightful comments and suggestions. The authors would also like to thank Wenjing Fang from Ant Group for helpful discussions on `HEP-XGB` and Yuncheng Wu from National University of Singapore for the experimental details of `Pivot`.

References

[1] Default of credit card clients dataset. https://www.kaggle.com/datasets/uciml/default-of-credit-card-clients-dataset.

[2] Libsvm data: Classification (binary class). https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html, June 2022.

[3] Mark Abspoel, Daniel Escudero, and Nikolaj Volgushev. Secure training of decision trees with continuous attributes. *Proc. Priv. Enhancing Technol.*, 2021(1):167–187.

[4] Amit Agarwal, Stanislav Peceny, Mariana Raykova, Phillipp Schoppmann, and Karn Seth. Communication Efficient Secure Logistic Regression, 2022. https://eprint.iacr.org/2022/866.

[5] Nitin Agrawal, Ali Shahin Shamsabadi, Matt J. Kusner, and Adrià Gascón. QUOTIENT: two-party secure neural network training and prediction. In *CCS*, pages 1231–1247, 2019.

[6] Martin R. Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of Learning with Errors. *J. Math. Cryptol. 2015 (commit c50ab18)*, pages 169–203, 2015.

[7] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More Efficient Oblivious Transfer and Extensions for Faster Secure Computation. In *CCS*, pages 535–548, New York, NY, USA, 2013.

[8] Donald Beaver. Efficient multiparty protocols using circuit randomization. In *CRYPTO*, volume 576, pages 420–432, 1991.

[9] John Bethencourt. libpaillier. https://acsc.cs.utexas.edu/libpaillier, 2022.

[10] Ran Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, 2000.

[11] Shaosheng Cao, Xinxing Yang, Cen Chen, Jun Zhou, Xiaolong Li, and Yuan Qi. TitAnt: Online Real-time Transaction Fraud Detection in Ant Financial. *Proc. VLDB Endow (2019).*, 12(12):2082–2093.

[12] Octavian Catrina and Amitabh Saxena. Secure Computation with Fixed-Point Numbers. In *FC*, volume 6052, pages 35–50, 2010.

[13] Sylvain Chatel, Apostolos Pyrgelis, Juan Ramón Troncoso-Pastoriza, and Jean-Pierre Hubaux. SoK: Privacy-Preserving Collaborative Tree-based Model Learning. *Proc. Priv. Enhancing Technol.*, 2021(3):182–203.

[14] Hao Chen, Wei Dai, Miran Kim, and Yongsoo Song. Efficient Homomorphic Conversion Between (Ring) LWE Ciphertexts. In *ACNS*, pages 460–479, 2021.

[15] Tianqi Chen and Carlos Guestrin. XGBoost: A Scalable Tree Boosting System. In *SIGKDD*, pages 785–794, 2016.

[16] Weijing Chen, Guoqiang Ma, Tao Fan, Yan Kang, Qian Xu, and Qiang Yang. SecureBoost+ : A High Performance Gradient Boosting Tree Framework for Large Scale Vertical Federated Learning. *CoRR*, abs/2110.10927, 2021.

[17] Kewei Cheng, Tao Fan, Yilun Jin, Yang Liu, Tianjian Chen, Dimitrios Papadopoulos, and Qiang Yang. SecureBoost: A Lossless Federated Learning Framework. *IEEE Intell. Syst.*, 36(6):87–98, 2021.

[18] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. A full RNS variant of approximate homomorphic encryption. In *SAC 2018, Calgary, Canada, August 15-17, 2018, Revised Selected Papers*, pages 347–368.

[19] Sebastiaan de Hoogh, Berry Schoenmakers, Ping Chen, and Harm op den Akker. Practical Secure Decision Tree Learning in a Teletreatment Application. In *FC*, volume 8437, pages 179–194, 2014.

[20] Changyu Dong, Liqun Chen, and Zikai Wen. When private set intersection meets big data: an efficient and scalable protocol. In *CCS 2013*, pages 789–800.

[21] Cynthia Dwork, Guy N. Rothblum, and Salil P. Vadhan. Boosting and differential privacy. In *FOCS 2010*.

[22] Wenjing Fang, Derun Zhao, Jin Tan, Chaochao Chen, Chaofan Yu, Li Wang, Lei Wang, Jun Zhou, and Benyu Zhang. Large-scale Secure XGB for Vertical Federated Learning. In *CIKM*, pages 443–452, 2021.

[23] Sam Fletcher and Md Zahidul Islam. Differentially private random decision forests using smooth sensitivity. *Expert Syst. Appl.*, 78:16–31, 2017.

[24] Michael J. Freedman, Kobbi Nissim, and Benny Pinkas. Efficient Private Matching and Set Intersection. In *EUROCRYPT*, volume 3027, pages 1–19, 2004.

[25] Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, 29(5):1189 – 1232, 2001.

[26] Chong Fu, Xuhong Zhang, Shouling Ji, Jinyin Chen, Jingzheng Wu, Shanqing Guo, Jun Zhou, Alex X. Liu, and Ting Wang. Label Inference Attacks Against Vertical Federated Learning. In *USENIX Security*, 2022.

[27] Fangcheng Fu, Yingxia Shao, Lele Yu, Jiawei Jiang, Huanran Xue, Yangyu Tao, and Bin Cui. VF^{2}Boost: Very Fast Vertical Federated Gradient Boosting for Cross-Enterprise Learning. In *SIGMOD*, pages 563–576, 2021.

[28] Jonas Geiping, Hartmut Bauermeister, Hannah Dröge, and Michael Moeller. Inverting Gradients - How easy is it to break privacy in federated learning? In *NeurIPS*, 2020.

[29] Craig Gentry, Shai Halevi, and Nigel P. Smart. Homomorphic evaluation of the AES circuit. In *CRYPTO*, pages 850–867, 2012.

[30] Nayanaba Pravinsinh Gohil and Arvind D. Meniya. Click Ad Fraud Detection Using XGBoost Gradient Boosting Algorithm. In *Computing Science, Communication and Security*, pages 67–81, 2021.

[31] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *TOC*, page 218–229, 1987.

[32] Oded Goldreich. *The Foundations of Cryptography - Volume 2: Basic Applications*. Cambridge University Press, 2004.

[33] Kyoohyung Han, Jinhyuck Jeong, Jung Hoon Sohn, and Yongha Son. Efficient privacy preserving logistic regression inference and training. *IACR Cryptol. ePrint Arch.*, 2020:1396, 2020.

[34] Zhicong Huang, Wen-jie Lu, Cheng Hong, and Jiansheng Ding. Cheetah: Lean and Fast Secure Two-Party Deep Neural Network Inference. *USENIX Security*, 2022.

[35] Zhicong Huang, Wen-jie Lu, Cheng Hong, and Jiansheng Ding. OpenCheetah. `https://github.com/Alibaba-Gemini-Lab/OpenCheetah`, 2022.

[36] Intel HEXL (release 1.2.2). `https://arxiv.org/abs/2103.16400`, October 2021.

[37] Xiao Jin, Pin-Yu Chen, Chia-Yi Hsu, Chia-Mu Yu, and Tianyi Chen. CAFE: Catastrophic Data Leakage in Vertical Federated Learning. In *NeurPIS*, 2021.

[38] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. LightGBM: A Highly Efficient Gradient Boosting Decision Tree. In *NeurIPS*, pages 3146–3154, 2017.

[39] Marcel Keller. MP-SPDZ: A versatile framework for multi-party computation. In *CCS 2020*, pages 1575–1590.

[40] Miran Kim, Yongsoo Song, Shuang Wang, Yuhou Xia, and Xiaoqian Jiang. Secure Logistic Regression based on Homomorphic Encryption. *IACR Cryptol. ePrint Arch.*, page 74, 2018.

[41] Vladimir Kolesnikov, Ahmad-Reza Sadeghi, and Thomas Schneider. Improved Garbled Circuit Building Blocks and Applications to Auctions and Computing Minima. In *CANS 2009*, volume 5888, pages 1–20. Springer, 2009.

[42] Yehuda Lindell and Benny Pinkas. Privacy Preserving Data Mining. In *CRYPTO*, volume 1880, pages 36–54, 2000.

[43] Xiaoliang Ling, Weiwei Deng, Chen Gu, Hucheng Zhou, Cui Li, and Feng Sun. Model Ensemble for Click Prediction in Bing Search Ads. In *World Wide Web Companion*, pages 689–698, 2017.

[44] Jian Liu, Mika Juuti, Yao Lu, and N. Asokan. Oblivious neural network predictions via minionn transformations. In *CCS*, pages 619–631, 2017.

[45] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *AISTATS*, pages 1273–1282, 2017.

[46] Payman Mohassel and Yupeng Zhang. SecureML: A system for scalable privacy-preserving machine learning. In *SP*, pages 19–38. IEEE Computer Society, 2017.

[47] National Institute of Standards and Technology (NIST). Nist, "nist special publication 800-57, recommendation for key management part 1: General (rev. 3),". 2012.

[48] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT '99*, volume 1592, pages 223–238.

[49] Deevashwer Rathee, Thomas Schneider, and K. K. Shukla. Improved Multiplication Triple Generation over Rings via RLWE-Based AHE. In *CANS 2019*, pages 347–359, 2019.

[50] M. Sadegh Riazi, Christian Weinert, Oleksandr Tkachenko, Ebrahim M. Songhori, Thomas Schneider, and Farinaz Koushanfar. Chameleon: A hybrid secure computation framework for machine learning applications. In *Asia CCS*, pages 707–721, 2018.

[51] Rushin, Stancil, Sun, MY, Adams, and Beling. Horse race analysis in credit card fraud-deep learning, logistic regression, and gradient boosted tree. *SIEDS*, pages 117–121, 2017.

[52] Ismail San, Nuray At, Ibrahim Yakut, and Huseyin Polat. Efficient paillier cryptoprocessor for privacy-preserving data mining. *Secur. Commun. Networks*, 9(11):1535–1546, 2016.

[53] Microsoft SEAL (release 3.7). https://github.com/Microsoft/SEAL, September 2021. Microsoft Research, Redmond, WA.

[54] Nigel P. Smart and Frederik Vercauteren. Fully homomorphic SIMD operations. *Des. Codes Cryptogr.*, 71(1):57–81, 2014.

[55] Zhenya Tian, Jialiang Xiao, Haonan Feng, and Yutian Wei. Credit Risk Assessment based on Gradient Boosting Decision Tree. *Procedia Computer Science*, 174:150–160, 2020.

[56] Mihai Todor. SeComLib. https://github.com/mihaitodor/SeComLib, 2022.

[57] Xiao Wang, Alex J. Malozemoff, and Jonathan Katz. EMP-toolkit: Efficient MultiParty computation toolkit. https://github.com/emp-toolkit, 2022.

[58] Yuncheng Wu, Shaofeng Cai, Xiaokui Xiao, Gang Chen, and Beng Chin Ooi. Privacy Preserving Vertical Federated Learning for Tree-based Models. *Proc. VLDB Endow. 2020*, pages 2090–2103.

[59] Kang Yang, Chenkai Weng, Xiao Lan, Jiang Zhang, and Xiao Wang. Ferret: Fast Extension for Correlated OT with Small Communication. In *CCS*, pages 1607–1626, 2020.

[60] Andrew Chi-Chih Yao. How to Generate and Exchange Secrets. In *FOCS*, pages 162–167, 1986.

[61] Zutao Zhu and Wenliang Du. Understanding Privacy Risk of Publishing Decision Trees. In *IFIP DBSec*, volume 6166 of *Lecture Notes in Computer Science*, pages 33–48. Springer, 2010.

A Correctness of Fig. 8

Proof 1 *For the correctness, we argue that the RLWE ciphertext* CT *in Step 2 encrypts a polynomial* $q' \cdot b + \hat{a} \cdot \mathsf{lift}(\underline{\mathsf{sk}}) \in \mathbb{A}_{N,q}$ *from the multiplicative property of RLWE and the definition the gadget decomposition. Writing out* $q' \cdot b + \hat{a} \cdot \mathsf{lift}(\underline{\mathsf{sk}})$ *gives:*

$$
\begin{aligned}
& q' \cdot b + \hat{a} \cdot \mathsf{lift}(\underline{\mathsf{sk}}) \\
& \equiv q' \cdot \left(b + (\sum_{i=0}^{N-1} \mathbf{a}[i] X^i) \cdot (\underline{\mathsf{sk}}[0] - \sum_{i=1}^{N-1} \underline{\mathsf{sk}}[i] X^{N-i}) \right) \\
& \equiv q' \cdot (b + \langle \mathbf{a}, \mathsf{sk} \rangle + r_0 \cdot \underline{q}) + \sum_{i>0} r_i X^i \text{ for some } \{r_i\} \\
& \equiv q' \cdot \lceil \underline{q}/p \cdot m \rfloor + r_0 \cdot q + \sum_{i>0} r_i X^i \\
& \equiv q' \cdot (\underline{q}/p \cdot m + e_{\text{round}}) + \sum_{i>0} r_i X^i \mod (X^N + 1, q).
\end{aligned}
$$

In other words, if the term $q' \cdot e_{\text{round}}$ *is bounded within* $q/(2p)$ *then* $\mathsf{Extract}(\mathsf{CT}, 0)$ *can give a valid LWE ciphertext that decrypts to* m *under the longer key* sk.

B Computations call $\mathcal{F}_{\mathrm{COT}}$

B.1 Private choice and shared message

P_0 inputs a private choice $c \in \{0,1\}$ and his shared message $\langle z \rangle_0 \in \mathbb{Z}_{2^\ell}$. P_1 inputs his share $\langle z \rangle_1 \in \mathbb{Z}_{2^\ell}$. At the end, they obtain their corresponding share of $\langle c \cdot z \rangle$. P_1 sends $(\mathsf{Send}, f(x) = x + \langle z \rangle_1)$ to $\mathcal{F}_{\mathrm{COT}}$ and then receives $x_1 \in \mathbb{Z}_{2^\ell}$. P_0 sends (Recv, c) to $\mathcal{F}_{\mathrm{COT}}$ and then receives $x_0 \in \mathbb{Z}_{2^\ell}$. P_0 then sets $\langle c \cdot z \rangle_0 \equiv c \cdot \langle z \rangle_0 + x_0 \bmod 2^\ell$, and P_1 sets $\langle c \cdot z \rangle_1 \equiv -x_1 \bmod 2^\ell$.

B.2 Shared choice and shared message

Each player P_l inptus a shared choice $\langle c \rangle_l^B \in \{0,1\}$ and a shared message $\langle w \rangle_l \in \mathbb{Z}_{2^\ell}$. At the end, they obtain their corresponding share of $\langle c \cdot w \rangle$. We need two $\mathcal{F}_{\mathrm{COT}}$ instances for this computation. For one instance, P_0 sends $(\mathsf{Send}, f(x) = x + (1 - 2 \cdot \langle c \rangle_0^B) \cdot \langle w \rangle_0)$ to $\mathcal{F}_{\mathrm{COT}}$ and then receives x_0. P_1 sends $(\mathsf{Recv}, \langle c \rangle_1^B)$ to $\mathcal{F}_{\mathrm{COT}}$ and receives y_1. For the other, P_1 sends $(\mathsf{Send}, f(x) = x + (1 - 2 \cdot \langle c \rangle_1^B) \cdot \langle w \rangle_1)$ to $\mathcal{F}_{\mathrm{COT}}$ and then receives x_1. P_0 sends $(\mathsf{Recv}, \langle c \rangle_0^B)$ to $\mathcal{F}_{\mathrm{COT}}$ and receives y_0. According to the COT property, we have $y_1 \equiv x_0 + \langle c \rangle_1^B \cdot (1 - 2 \cdot \langle c \rangle_0^B) \cdot \langle w \rangle_0$ and $y_0 \equiv x_1 + \langle c \rangle_0^B \cdot (1 - 2 \cdot \langle c \rangle_1^B) \cdot \langle w \rangle_1$. Finally, P_l sets $\langle c \cdot w \rangle_l \equiv \langle c \rangle_l^B \cdot \langle w \rangle_l + y_l - x_l \mod 2^\ell$.

B.3 AND-style choice and shared message

Each player P_l inptus a choice $\beta_l \in \{0,1\}$ and a shared message $\langle w \rangle_l \in \mathbb{Z}_{2^\ell}$. At the end, they obtain their corresponding share of $\langle (\beta_0 \wedge \beta_1) \cdot w \rangle$. We also need two $\mathcal{F}_{\mathrm{COT}}$ instances for this computation. For one $\mathcal{F}_{\mathrm{COT}}$ instance, P_0 sends $(\mathsf{Send}, f(x) = x + \beta_0 \cdot \langle w \rangle_0)$ to $\mathcal{F}_{\mathrm{COT}}$ and then receives x_0. P_1 sends (Recv, β_1) to $\mathcal{F}_{\mathrm{COT}}$ and receives y_1. For the other instance, P_1 sends $(\mathsf{Send}, f(x) = x + \beta_1 \cdot \langle w \rangle_1)$ to $\mathcal{F}_{\mathrm{COT}}$ and then receives x_1. P_0 sends (Recv, β_0) to $\mathcal{F}_{\mathrm{COT}}$ and receives y_0. According to the COT property, we have

$$y_1 \equiv x_0 + \beta_1 \cdot (\beta_0 \cdot \langle w \rangle_0), \quad y_0 \equiv x_1 + \beta_0 \cdot (\beta_1 \cdot \langle w \rangle_1).$$

Finally, P_l sets $\langle(\beta_0 \wedge \beta_1) \cdot w\rangle_l \equiv y_l - x_l \mod 2^\ell$.

C Security Proofs

Proof 2 (Theorem 1 (Sketch)) *The correctness follows trivially in the $\mathcal{F}_{\text{mul}}$-, $\mathcal{F}_{\text{greater}}$- and $\mathcal{F}_{\text{COT}}$ hybrid. For privacy, the view of P_l during the execution of* Seg3Sigmoid *only consists of the messages received from $\mathcal{F}_{\text{mul}}$, $\mathcal{F}_{\text{greater}}$ and $\mathcal{F}_{\text{COT}}$, and no message is revealed. Thus the privacy follows by simply invoking the corresponding simulators of $\mathcal{F}_{\text{mul}}$, $\mathcal{F}_{\text{greater}}$ and $\mathcal{F}_{\text{COT}}$ on uniform random values.*

Proof 3 (Theorem 2 (Sketch)) *The correctness follows trivially by the additive homomorphic property of (R)LWE HEs. For privacy, the view of P_l during the execution of* BinMatVec *consists of the $\lceil n/N \rceil$ RLWE ciphertexts (under P_{1-l}'s key) received in Step 1 only. We construct the simulator $\mathcal{S}_l(\langle \mathbf{M} \cdot \mathbf{g}\rangle_l, \mathsf{pk}_{1-l})$ as follows:*

1. *On* A2H *in Step 1, writes $\lceil n/N \rceil$ ciphertexts of zero $\mathsf{RLWE}^{N,q,2^\ell}_{\mathsf{pk}_{1-l}}(0)$ to $\mathcal{V}^\Pi_l$ using the public key pk_{1-l}.*
2. *On* Output *in Step 9, sends the shares $\langle \mathbf{M} \cdot g\rangle_l$ ot P_l.*

The privacy against an adversary P_l is directly reduced to the semantic security of RLWE.

On the other hand, the view of P_{1-l} consists of RLWE ciphertexts of uniformly randomized polynomials from H2A. *We construct the simulator $\mathcal{S}_{1-l}(\langle \mathbf{M} \cdot \mathbf{g}\rangle_{1-l}, \mathsf{pk}_{1-l})$ as follows:*

1. *On* H2A *in Step 9, writes $\{(\hat{b}_j + \hat{r}_j, \hat{a}_j)\}_{j\in[\lceil n/N \rceil]}$ to $\mathcal{V}^\Pi_{1-l}$ where the tuple $(\hat{b}_j, \hat{a}_j) \leftarrow \mathsf{RLWE}^{N,q,2^\ell}_{\mathsf{pk}_{1-l}}(0)$, and the polynomial $\hat{r}_j$ is sampled from $\mathbb{A}_{N,q}$ uniformly at random.*
2. *On* Output *in Step 9, sends the shares $\langle \mathbf{M} \cdot g\rangle_{1-l}$ to P_{1-l}.*

The privacy against an adversary P_{1-l} follows a similar argument in [34] that the uniform polynomial $\hat{r}$ hides the noise term in RLWE. As a result, even P_{1-l} can decrypt the ciphertexts, what he can see are all uniformly distributed.

Proof 4 (Theorem 3 (Sketch)) *We construct a simulator $\mathcal{S}_l$ for P_l's view $\mathcal{V}^\Pi_l$. Remind that $\mathcal{S}_l$ takes as input of* Input_l *and* Output_l*. The main issue in the proof involves showing that the control flow can be predicted from $\mathcal{S}_l$'s input. It is easily to see all the steps, excluding from Step 9 to Step 13, of Fig. 7 are predictable with* Input_l*. The control flow from Step 9 to Step 13 are also predictable from* Output_l*. For the simulator $\mathcal{S}_l$, on receiving a* Open *command in Step 9, $\mathcal{S}_l$ checks, by looking at* Output_l*, if $C_l[k] \neq \perp$, then $\mathcal{S}_l$ writes the bit l to $\mathcal{V}^\Pi_l$. Otherwise, $\mathcal{S}_l$ writes $1-l$ to $\mathcal{V}^\Pi_{-l}$. In Step 10, if $C_l[k] \neq \perp$, $\mathcal{S}_l$ writes $C_l[k]$ to $\mathcal{V}^\Pi_l$. Step 11 and Step 12 are local computations. Finally, on receiving the* COT *command in Step 13, $\mathcal{S}_l$ checks if $C_l[k] \neq \perp$, then $\mathcal{S}_l$ sends $(\mathbf{b}^{(k)}_*, \mathsf{Recv})$ to the $\mathcal{F}_{\text{COT}}$ as the receiver and writes what it has received from $\mathcal{F}_{\text{COT}}$ into $\mathcal{V}^\Pi_l$. Otherwise, $\mathcal{S}_l$ sends $(\mathbf{x}, \mathsf{Send})$ to $\mathcal{F}_{\text{COT}}$ as the sender on a uniform random $\mathbf{x} \in \mathbb{Z}^n_{2^\ell}$, and writes what it has received from $\mathcal{F}_{\text{COT}}$ into $\mathcal{V}^\Pi_l$. The computation then continues to the next node. This completes the proof.*

D Noise Analysis

A2H introduces an initial noise e_0 whose variance $V(e_0) = \sigma^2$. After performing n' LWE additions, the noise becomes e_1 with the variance $V(e_1) = n'\sigma^2$. In SEAL, we apply the special prime technique [29] for key-switching (KS). Specifically, given an RLWE ciphertext with a noise variance v, after the KS the noise variance becomes $V_{\text{ks}}(v) = \frac{1}{12P^2} N v \sum_i q_i^2 + \frac{N}{24}$, where P is the special prime. LWEDimLift is also a kind of KS. Thus, the noise after LWEDimLift becomes e_2 with a variance $V(e_2) = V_{\text{ks}}(V(e_1))$. According to [14], the noise after PackLWEs is e_3 with a variance $\frac{1}{3}(N^2-1)V_{\text{ks}}(v')$ where v' is the input noise variance i.e., $v' = V(e_2)$ in *Squirrel*. To have a correct decryption in H2A, we need the noise e_3 is bounded by $q/2^{\ell+1}$. The value $6\sqrt{V(e_3)}$ is usually used as the heuristic upper bound of e_3. Putting in our parameters $q_0 \approx 2^{55}$, $q_1 \approx 2^{54}$, $P \approx 2^{60}$, $N = 8192$, $\ell = 64$, and $\sigma = 3.2$ we know $n' < 5 \times 10^{16}$. In other words, our HE parameters can support a lossless gradient aggregation up to 5×10^{16} samples which is fairly enough for any practical application.

E Fourier Coefficients

The specific Fourier coefficients in (5) are given as follows.

$\omega_0 = 0.5$

$\omega_1 = 0.6172949043536653$, $\omega_2 = -0.0341990021261339$

$\omega_3 = 0.1693788502244572$, $\omega_4 = -0.0460333847898619$

$\omega_5 = 0.0816712796122188$, $\omega_6 = -0.0433475059227459$

$\omega_7 = 0.0507073237098216$, $\omega_8 = -0.0369643373243371$

Under these coefficients, the range of (5) is bounded by 1 within the interval $[-5.6, 5.6]$ (see Fig. 4).

F A Potential Risk in the Previous H2A

Both `Pivot` [58] and `HEP-XGB` [22] use a Paillier-like HE for the H2A conversion. Briefly, they compute $\mathsf{HE}(x) + r$ using a random mask $r \in \mathbb{Z}$ to "re-share" the value $x \in \mathbb{Z}$ to $\langle x \rangle$. The secure range of r is a function of x, e.g., $|r| = |x| \cdot 2^{40}$. For the GBDT statistic aggregation, the upper bound of x is $O(|x|) = n \cdot 2^\ell$. However, `Pivot` and `HEP-XGB` assume $|x| < 2^\ell$ which is not the case becase a Paillier-like HE is not supporting modulo 2^ℓ homomorphically. Using a such small random mask r might leak the most significant bits of x. For example, this might give the adversary a hint that how many samples are added on that tree node.