Practical Anonymous Two-Party Gradient Boosting Decision Tree

Chenyu Huang*¶ Fan Zhang*¶✉, Minxin Du†✉, Sherman S. M. Chow‡,
Huangxun Chen§, Huaming Rao*, Danqing Huang*, Bo Qian* and Peng Chen*
{*chenyuhuang, zxfanzhang*}*@tencent.com, minxin.du@polyu.edu.hk*
**Tencent*, †*Hong Kong Polytechnic University*, ‡*Chinese University of Hong Kong*, §*HKUST-GZ*

***Abstract*—Structured data is well handled by gradient-boosted decision trees (GBDT), which are usually trained on vertically partitioned features across mutually distrustful parties. High speed and interpretability make GBDTs popular in finance and healthcare, where neural networks may fall short. Enabling secure computation for GBDTs poses unique challenges, requiring secure record alignment for comparison. Relying on private set intersection (PSI) is a *de facto* approach. Mistaking PSI for a safety measure actually exposes which record identifiers (IDs) are shared between the datasets. Although circuit-PSI could help, it is costly for generic uses. New ideas are needed to efficiently train in a "dark forest."**

Aiming to hide the IDs, we initiate the study of anonymous GBDT training on split data held by two parties. Dual circuit-PSI in our design lets the parties alternate as receiver to run pick-then-sum over local features. Via oblivious programmable pseudorandom functions, we propagate circuit-PSI outputs as shared state across runs. Avoiding universal alignment, we resolve the neglected dilemma that ID hiding incurs a cost that scales with domain size. Next, we halve the cost of ciphertext packing used to convert single-instruction multiple-data homomorphic encryption from (ring) learning with errors in prior secure GBDT (Usenix Security '23) and related secure machine-learning computations. Comparative experiments show our protocol remains competitive with leaky approaches in efficiency. Enabling ID-hiding aggregation, our techniques can extend to other vertically partitioned analytics.

1. Introduction

Structured data is well served by gradient-boosted decision trees (GBDT), an interpretable, state-of-the-art learner for domains like finance [1], healthcare [2], advertising [3], and beyond, even amid the rise of deep neural networks [4]. However, privacy regulations preclude plaintext centralization across distrustful silos, motivating private *vertical* training on disjoint feature sets over overlapping populations. Even with secure multiparty computation, different datasets seldom align record-by-record, for example, a bank seeking payment-provider signals often finds that rosters that only partially overlap and differ in order (specifically, row i on one side need not be the same as row i on the other). Reconciling them requires privately computing the identifier (*e.g.*, national ID) intersection and agreeing on a common order, and the mapping must be refreshed as rosters churn. Most secure two-party protocols [5], [6], [7], [8] address this by running private set intersection (PSI) [9], [10] for *pre-alignment*, a setup step that determines which identifiers are shared across the datasets while hiding others. Although common, revealing intersection membership is sensitive: linking credit-card transaction IDs to advertising data enables re-identification [11], [12], and revealing which patients two hospitals share can leak geographic or medical information, *e.g.*, oncology referrals in a small suburb. Noting that GBDT needs only *features*, not identifiers, we seek to keep *all* IDs and their alignment private throughout collaborative training, a regime we call *anonymous training*.

1.1. Securely Sewing Metadata via *Secret Match*

Shielding identifiers entirely, we introduce the first *anonymous* two-party GBDT protocol that never reveals them, intersecting or not. Matching that goal, circuit-PSI [13], [14], [15] is a natural starting point because it can evaluate arbitrary circuits on payloads attached to the hidden intersection. Current uses, however, are limited to order-independent aggregates like sums [12] or simple statistics [16]; even counting the intersection size incurs $\sim$20$\times$ more cost than vanilla PSI [17], let alone modeling an entire GBDT as an arithmetic circuit.

The difficulty stems from an optimization in circuit-PSI: the parties employ asymmetric hashing. The receiver inserts each ID into a *cuckoo* table with *one* bucket per ID, and a stash handles overflows from collisions. In contrast, the sender places each ID in *all* corresponding buckets by *simple hashing*, creating a one-to-many mapping. PSI then reduces to per-bucket private set membership (PSM) testing [18]. Such asymmetry poses two challenges for GBDT training:

I) **Gradient histograms**: Each party holds n samples with m real-valued features, discretized into an $mB \times n$ binary matrix $\mathbf{M}$ (B bins per feature). Each node maintains a

¶Co-first Authors, ✉ Co-corresponding

Minxin Du is supported in part by the HK PolyU Start-up Fund (P0052722). His work is partly done at CUHK, supported by Direct Grant (4055238). Sherman Chow is supported in part by the General Research Funds (CUHK 14210825, 14210621) from the Research Grant Council, the Hong Kong Special Administrative Region of the People's Republic of China; and by Direct Grant (4055238) and the Strategic Impact Enhancement Fund (3135517) from CUHK. Chow contributes to scholarly work, but administrative, business, or compliance matters fall outside his purview.

sample indicator $\mathbf{b} \in \{0,1\}^n$ tracking which samples reach it. Gradients are filtered by $\mathbf{b}$ and aggregated by multiplying $\mathbf{M}$ to form histograms. The receiver, knowing the one-to-one mapping from columns to gradients, can run a secure two-party functionality $\mathcal{F}_{\mathsf{BinMatVec}}$ for binary matrix-vector multiplication to multiply $\mathbf{M}$ with gradient shares. However, the sender cannot because one-to-many hashing obscures which gradient aligns to which column.

II) **Indicator synchronization**: A Boolean indicator vector tracks which samples reach each node, and PSI-based schemes start with $\mathbf{1}^n$. To keep it secret while allowing efficient local updates (Section 4.1), Squirrel resorts to AND-sharing [5], which is leaky in this setting. More importantly, the indicator synchronization for children fails when the sender owns the split due to the one-to-many mapping.

These challenges prompt a broader question: *"Can we perform order-dependent computation on circuit-PSI outputs without generic secure multiparty computation?"* The techniques here answer affirmatively, enabling practical anonymous GBDT and informing other secure computations.

1.2. Technical Overview

Base solution: Circuit-PSI [13], [14], [15] enables oblivious processing of *payloads* attached to intersecting IDs. In our base design, the *sender* attaches its binary feature matrix as the payload. After PSI, the matrix is secret-shared and aligned with the joint ID list. Gradient histograms are then built by a pick-then-sum procedure with secure multiplexer $\mathcal{F}_{\mathsf{mux}}$ [19]; this entails $O(nmB)$ oblivious transfers (OTs).

To update the per-node sample indicator, we introduce a new OT-based *oblivious indicator synchronization* (OIS) protocol. If the *receiver* owns the best split, it directly shares one column of its feature matrix; otherwise, the two parties execute a 1-out-of-mB OT for every row so that the sender can retrieve the required column without revealing the index. In either case, secure AND [19] and (plaintext) string XOR then derive the children's indicator shares (Section 4.2.2). While feasible and anonymous, the base design is OT-heavy.

Dual-circuit-PSI (which "symmetrizes" alignment): Our final design, with codename OTSA, reduces OTs and symmetrizes alignment via a new dual-circuit-PSI framework: two circuit-PSI instances run with swapped roles, so each party learns an ID-to-bucket mapping once. No feature matrix needs to be sent as a payload; both remain local, and gradient aggregation switches from $\mathcal{F}_{\mathsf{mux}}$ to communication-friendly $\mathcal{F}_{\mathsf{BinMatVec}}$. We use AnonGBDT to refer to both the base design and the final extension.

Fast ciphertext packing (halving costs): Our instantiation uses homomorphic encryption instantiated from the learning-with-error (LWE) assumption and its ring variant, ring LWE (RLWE) assumption.[1] In $\mathcal{F}_{\mathsf{BinMatVec}}$, gradient shares are converted to RLWE ciphertexts, homomorphically picked-and-summed, and output as multiple LWE ciphertexts that are packed into one RLWE ciphertext. Our FastPackLWEs subroutine (Section 5.2) first packs each adjacent LWE pair directly into one RLWE (eliminating lifting), then recursively merges RLWEs. This streamlines the raw LWEDimLift-then-PackLWEs approach [5], hence halving inputs and runtime, where the former is for lifting an LWE ciphertext to a larger lattice dimension and the latter is for homomorphically merging LWE ciphertexts into one RLWE ciphertext that decrypts to a related polynomial [20].

Batched OPPRF-based OIS: Keeping both feature matrices local invalidates the row-wise OT-based OIS in the base protocol. An OIS based on oblivious programmable pseudorandom function (OPPRF) is introduced (Section 5.3): the split owner programs an OPPRF so the counterparty obtains exact assignment shares for all its cuckoo buckets. Level-wise batching yields a batched variant $\mathcal{F}_{\mathsf{BOIS}}$ with $O(D)$ cost instead of $O(2^D)$ for tree depth D.

We present independent optimizations (Sections 4.2.3 and 5.4), *e.g.*, better sigmoid approximation, gradient packing, and lightweight argmax, which also accelerate Squirrel. These refinements help the dual-circuit-PSI design outperform the base design and match *non-anonymous* Squirrel, despite partial per-instance recomputation, *e.g.*, gradients.

Our main contributions are summarized below.

1) To our knowledge, AnonGBDT is the first two-party protocol that enables *anonymous* GBDT training (and inference [21]). AnonGBDT builds on circuit-PSI [14], [15], which keeps every common ID hidden and maintains private alignment throughout training, for anonymity that prior PSI-based approaches did not provide.

2) Circuit-PSI asymmetric hashing (cuckoo vs. simple) creates two challenges, namely, gradient aggregation and indicator synchronization. In our base design, the sender's feature matrix is carried as a payload. Gradients are selected via the secure multiplexer [19], and indicators are reconciled via a custom OIS protocol. Both are functional yet OT-intensive.

3) $\mathsf{AnonGBDT}^{\mathsf{OTSA}}$ uses a new dual-circuit-PSI framework to symmetrize the workflow. *Both* parties now build histograms with the OT-free $\mathcal{F}_{\mathsf{BinMatVec}}$, optimized by our FastPackLWEs subroutine. Since the base OIS no longer applies, we propose an OPPRF-based variant with level-wise batching, which reduces communication logarithmically.

4) We evaluate on real-world and synthetic datasets. All executable code is open source[2]. $\mathsf{AnonGBDT}^{\mathsf{OTSA}}$ is up to $20\times$ faster and $44\times$ lighter than the base version, and it matches Squirrel under local area network (LAN) settings. Its runtime and traffic never exceed $1.8\times$ those of Squirrel for wide area networks (WAN), while providing strictly stronger privacy. (We exclude the naïve baseline that embeds the *entire* GBDT training in circuit-PSI, which is orders of magnitude slower.)

2. Preliminaries

$|S|$ is the size of the set S. Sampling from a finite set S or a distribution $\mathcal{D}$ is denoted by $x \leftarrow_{\$} S$ or $x \leftarrow \mathcal{D}$. $\perp$ is

1. Hereinafter, we may use shorthand LWE/RLWE to refer to ciphertexts produced by an encryption scheme under the LWE/RLWE assumption.

2. https://zenodo.org/records/17373936

TABLE 1. NOTATIONS WITH DESCRIPTIONS

Symbol	Description
$\langle\cdot\rangle_l^{\mathsf{A}}, \langle\cdot\rangle_l^{\mathsf{B}}$	P_l's arithmetic and boolean shares
$[\![\cdot]\!]$	RLWE HE-encrypted variable
$\mathbf{ID}_l$, $\mathbf{X}_l$	Sample IDs and features owned by party P_l
n_l, m_l	Number of samples and features in $\mathbf{X}_l$
$\mathbf{M}_l$	Discrete feature binary matrix of $\mathbf{X}_l$
B	Number of bins in $\mathbf{M}_l$
$\mathbf{y}$	Label set
T	Number of trees
D	Depth of tree
z_*, u_*	Best split: u_*-th bin of z_*-th feature
$\mathbf{q}$	Result of circuit-PSI
$\mathbf{g}^{(k)}$, $\mathbf{h}^{(k)}$	1st and 2nd order of gradients of node k
$\mathbf{b}^{(k)}$	Sample indicator of node k
$\mathbf{w}$	Leaf weight
$\mathbf{q}^l$	Result of l-th circuit-PSI
$\mathbf{g}^{(k,l)}, \mathbf{h}^{(k,l)}$	Gradients of node k based on $\mathbf{q}^{(l)}$
$\mathbf{b}^{(k,l)}$	Sample indicator of node k based on $q^{(l)}$
$\mathbf{T}_{\mathsf{ch}}^l, \mathbf{T}_{\mathsf{sh}}^l$	Cuckoo and simple hash tables of l-th circuit-PSI

the error symbol. The computational and statistical security parameters are λ and κ, respectively. The indicator function $\mathbf{1}\{x\}$ equals 1 if x is true; 0, otherwise.

Bold lowercase letters denote vectors (*e.g.*, $\mathbf{a}$) and bold uppercase letters denote matrices (*e.g.*, $\mathbf{M}$). For indices, $\mathbf{a}[i]$ is the i-th entry of $\mathbf{a}$, $\mathbf{M}[i]$ (or $\mathbf{M}_i$) is the i-th row of $\mathbf{M}$, and $\mathbf{M}[i,j]$ is the entry in row i, column j. Polynomials are written as $\widehat{a}$, where $\widehat{a}[i]$ is its i-th coefficient. Define the polynomial ring $\mathcal{R} = \mathbb{Z}[X]/(X^N+1)$, where N is a power of 2. For a modulus $Q \in \mathbb{Z}$, set $\mathcal{R}_Q = \mathcal{R}/Q\mathcal{R} = \mathbb{Z}_Q[X]/(X^N+1)$. Table 1 lists all notations.

2.1. Gradient Boosting Decision Tree

A GBDT is an ensemble of T decision trees $\{\mathcal{T}_t\}_{t=1}^T$ that are trained sequentially via gradient boosting [22]. Each tree partitions the feature space at internal nodes and assigns a real-valued weight to each leaf. For input $\mathbf{x}$, the ensemble prediction is the sum of leaf weights: $\tilde{y} = \sum_{t=1}^T \mathcal{T}_t(\mathbf{x})$.

Setup and gradient computation: Let $\mathcal{D} = \{(\mathbf{X}[i], \mathbf{y}[i])\}_{i=1}^n$ be a dataset of n samples with m real-valued features $\mathbf{X}[i] \in \mathbb{R}^m$ and ground-truth label $\mathbf{y}[i]$. We focus on binary classification with logistic cross-entropy loss, which is twice differentiable and well suited to second-order boosting [22].

After $t-1$ trees, for sample i, let $\tilde{\mathbf{y}}[i] = \sum_{j=1}^{t-1} \mathcal{T}_j(\mathbf{x})$ and $\mathbf{p}[i] = \mathsf{sigmoid}\big(\tilde{\mathbf{y}}[i]\big)$. The corresponding first- and second-order statistics used to fit the t-th tree are respectively

$$\mathbf{g}[i] = \mathbf{y}[i] - \mathbf{p}[i], \quad \mathbf{h}[i] = \mathbf{p}[i] \cdot (1 - \mathbf{p}[i]), \tag{1}$$

i.e., the canonical gradient and Hessian of logistic loss [22].

Node splitting: Consider a node with index set $\mathbf{I}$. For feature z and threshold ϕ, define the partition $\mathbf{I}_L = \{i \mid \mathbf{X}[i,z] \le \phi\}$ and $\mathbf{I}_R = \mathbf{I} \setminus \mathbf{I}_L$. With $\alpha \ge 0$ denoting L_2 regularization on leaf weights and $\gamma \ge 0$ the complexity penalty per split, the *split gain* (or "score") $\mathbf{L}_{\mathsf{sp}}$ is

$$\frac{1}{2}\left[\frac{\left(\sum_{i\in\mathbf{I}_L}\mathbf{g}[i]\right)^2}{\sum_{i\in\mathbf{I}_L}\mathbf{h}[i]+\alpha} + \frac{\left(\sum_{i\in\mathbf{I}_R}\mathbf{g}[i]\right)^2}{\sum_{i\in\mathbf{I}_R}\mathbf{h}[i]+\alpha} - \frac{\left(\sum_{i\in\mathbf{I}}\mathbf{g}[i]\right)^2}{\sum_{i\in\mathbf{I}}\mathbf{h}[i]+\alpha}\right] - \gamma. \tag{2}$$

The best split maximizes $\mathbf{L}_{\mathsf{sp}}$ over *all* candidates ϕ for *all* z.

Histogram-based Search: Exhaustive threshold scanning per feature is costly. We apply histogram binning [22], [23] to discretize each feature into B ordered *bins*. Let $\mathbf{M} \in \{0,1\}^{Bm\times n}$ be the "one-hot bin indicator" matrix, where $\mathbf{M}[zB+u, i] = 1$ iff sample i falls into bin u of feature z.[3]

Aggregating $\{\mathbf{g}[i], \mathbf{h}[i]\}$ into per-bin histograms supports efficient split evaluation by scanning bin boundaries, and the optimal one is denoted by $(z_\star, u_\star)$. For convenience, we pre-compute an exclusive-prefix matrix $\widetilde{\mathbf{M}} \in \{0,1\}^{Bm\times n}$ with rows $\widetilde{\mathbf{M}}[zB+u, i] = \bigvee_{v<u} \mathbf{M}[zB+v, i]$ so that any candidate left-child assignment $\mathbf{I}_L$ directly equals a row of $\widetilde{\mathbf{M}}$, avoiding runtime comparisons with bin boundaries.

Leaf weights: At maximum depth D, a leaf k has weight

$$\mathbf{w}[k] = -\textstyle\sum_{i\in\mathbf{I}_k}\mathbf{g}[i] / (\sum_{i\in\mathbf{I}_k}\mathbf{h}[i] + \alpha), \tag{3}$$

which is the closed-form minimizer of the second-order objective with L_2 regularization [22]. The prediction for sample i is then updated by adding $\mathbf{w}[k]$ if $i \in \mathbf{I}_k$.

2.2. Cryptographic Primitives

2.2.1. Secret Sharing. We adopt $(2,2)$-arithmetic additive secret sharing [24]. To share an ℓ-bit integer $x \in \mathbb{Z}_{2^\ell}$, party P_0 samples $r \leftarrow_{\$} \mathbb{Z}_{2^\ell}$, sends r to P_1, and sets $\langle x\rangle_0^{\mathsf{A}} = x - r$, $\langle x\rangle_1^{\mathsf{A}} = r$. Reconstruction is $\langle x\rangle_0^{\mathsf{A}} + \langle x\rangle_1^{\mathsf{A}} \mod \mathbb{Z}_{2^\ell}$.

A real $\widetilde{x} \in \mathbb{R}$ is encoded as $x = \lfloor \widetilde{x}\, 2^f \rceil \in [-2^{\ell-1}, 2^{\ell-1})$ using fixed-point representation with precision f.

- Addition: $\langle z\rangle^{\mathsf{A}} = \langle x\rangle^{\mathsf{A}} + \langle y\rangle^{\mathsf{A}}$ is computed by P_c locally.
- Multiplication: $\langle z\rangle^{\mathsf{A}} = \mathcal{F}_{\mathsf{mul}}(\langle x\rangle^{\mathsf{A}}, \langle y\rangle^{\mathsf{A}})$, where $\mathcal{F}_{\mathsf{mul}}$ is a secure two-party protocol using Beaver's triples [25].
- Division: $\langle z\rangle^{\mathsf{A}} = \mathcal{F}_{\mathsf{div}}(\langle x\rangle^{\mathsf{A}}, \langle y\rangle^{\mathsf{A}})$, where $\mathcal{F}_{\mathsf{div}}$ is a secure two-party protocol using the Goldschmidt method [26].
- Sigmoid: $\langle y\rangle^{\mathsf{A}} = \mathcal{F}_{\mathsf{sigmoid}}(\langle x\rangle^{\mathsf{A}})$ is a secure two-party protocol for efficiently evaluating the sigmoid function via three-segment Fourier approximation.
- ArgMax: $\langle z\rangle^{\mathsf{A}} = \mathcal{F}_{\mathsf{argmax}}(\langle \mathbf{x}\rangle^{\mathsf{A}})$ is a secure two-party protocol to output the shared index z of $\max \mathbf{x}$ [27].

$(2,2)$-boolean sharing [24] is also used. A boolean value $x \in \{0,1\}$ is XOR-shared as $x = \langle x\rangle_0^{\mathsf{B}} \oplus \langle x\rangle_1^{\mathsf{B}}$: Party P_0 samples $r \in \mathbb{Z}_2$, sets $\langle x\rangle_0^{\mathsf{B}} = x \oplus r$, and sends $\langle x\rangle_1^{\mathsf{B}} = r$ to P_1. The same generalizes to ℓ-bit strings with $r \leftarrow_{\$} \mathbb{Z}_{2^\ell}$.

- XOR: $\langle z\rangle^{\mathsf{B}} = \langle x\rangle^{\mathsf{B}} \oplus \langle y\rangle^{\mathsf{B}}$ is computed by P_c locally.
- NOT: $\langle y\rangle^{\mathsf{B}} = \neg\langle x\rangle^{\mathsf{B}}$, where one party locally computes $\neg$ on its share, while the other keeps its share unchanged.

3. For example, consider features age: [<30, ≥30] and weight: [<70kg, ≥70kg], discretized into $B = 2$ bins. When given $n = 4$ samples: $\{(25,65),(35,75),(45,85),(28,68)\}$, $\mathbf{M} = [1010|0101|1010|0101]^\top$.

• AND: $\langle z\rangle^{\mathsf{B}} = \mathcal{F}_{\mathsf{and}}(\langle x\rangle^{\mathsf{B}}, \langle y\rangle^{\mathsf{B}})$ is achieved by a secure two-party protocol using precomputed boolean triples [24].

• OR: $\langle z\rangle^{\mathsf{B}} = \mathcal{F}_{\mathsf{or}}(\langle x\rangle^{\mathsf{B}}, \langle y\rangle^{\mathsf{B}})$ via $x \wedge y \oplus (\neg x \oplus \neg y)$.

• Multiplexer: $\langle z\rangle^{\mathsf{A}} = \mathcal{F}_{\mathsf{mux}}(\langle x\rangle^{\mathsf{B}}, \langle y\rangle^{\mathsf{A}})$ takes *arithmetic* shares $\langle y\rangle^{\mathsf{A}}$ and *boolean* shares of a choice bit $\langle x\rangle^{\mathsf{B}}$ as input. It outputs $\langle z\rangle^{\mathsf{A}}$ with $z = y$ if $x = 1$; else $z = 0$. It can be realized via two $\mathcal{F}_{\mathsf{OT}}$ calls [19] (introduced below).

• Comparison: $\langle z\rangle^{\mathsf{B}} = \mathcal{F}_{\mathsf{greater}}(\langle x\rangle^{\mathsf{A}}, \langle y\rangle^{\mathsf{A}})$. If $x > y$, then z is 1; else z is 0. When the bitlength of x and y is small, the secure comparison can be efficiently realized with only $\mathcal{F}_{\mathsf{OT}}$ [19]. Otherwise, one can recursively divide them as $x = x_1||x_2, y = y_1||y_2$ and compares them based on $\mathbf{1}\{x > y\} = \mathbf{1}\{x_1 > y_1\} \oplus (\mathbf{1}\{x_1 = y_1\} \wedge \mathbf{1}\{x_2 > y_2\})$, where $\wedge$ and $>, =$ are respectively realized by $\mathcal{F}_{\mathsf{and}}$ and $\mathcal{F}_{\mathsf{OT}}$ [19].

2.2.2. Oblivious Transfer (OT). A 1-out-of-n OT [28] runs between a sender with n messages and a receiver with choice $i \in [0, n-1]$. The ideal functionality is $\mathcal{F}_{\mathsf{OT}}$; the receiver only learns the i-th message, and the sender learns nothing about i. Ferret OT [29] is used for its low communication.

2.2.3. Circuit-PSI. Private set intersection (PSI) allows two parties to learn $X \cap Y$ of their input sets while hiding non-intersecting elements. Circuit-PSI [13], [14] further enables evaluating an arbitrary circuit f on payloads attached to items in $X \cap Y$, without revealing the intersection itself.

Hashing phase: Fix e hash functions $h_1, \ldots, h_e$. The *receiver* inserts each $\mathbf{x} \in X$ into a cuckoo-hash table $\mathbf{T}_{\mathsf{ch}}$ of $(1+\varepsilon)n$ buckets: first empty among $\{h_i(\mathbf{x})\}$, with evictions as needed. With suitable (ε, e), no stash is required [13]. The *sender* inserts each $\mathbf{y} \in Y$ into all corresponding buckets of a simple-hash table $\mathbf{T}_{\mathsf{sh}}$, permitting multiple items per bucket (vs. at most one in $\mathbf{T}_{\mathsf{ch}}$). Both tables are padded to a common maximum bucket size. Shared hashes ensure any common item lands in one identical bucket on both sides, reducing PSI to per-bucket *private set membership* (PSM).

PSM via OPPRF: An oblivious programmable pseudo-random function functionality $\mathcal{F}_{\mathsf{OPPRF}}$ realizes PSM [13], [14]. For bucket i, the sender programs PRF F_i such that $F_i(\mathbf{y}) = r_i$ for every $\mathbf{y}$ in $\mathbf{T}_{\mathsf{sh}}[i]$ and outputs random values elsewhere. Both parties query $\mathcal{F}_{\mathsf{OPPRF}}$ on the receiver's items from $\mathbf{T}_{\mathsf{ch}}$; the receiver obtains r'_i. A secure equality test $\mathcal{F}_{\mathsf{equal}}$ then yields shared bits of $r_i \stackrel{?}{=} r'_i$.

Evaluation with payloads: Each item may carry a payload, enabling f to be computed on payloads of the (hidden) intersection [13]. Since a sender's bucket can contain multiple payloads, two batched OPPRFs select the correct one [13].

Looking ahead, our base design attaches the receiver's labels and the sender's (binary-matrix encoded) features as payloads to $\mathcal{F}_{\mathsf{CPSI}}$. AnonGBDT$^{\mathsf{OTSA}}$ alternates receiver roles and uses OPPRF to synchronize intermediate states; only labels are carried as payloads, improving efficiency.

2.2.4. Additive Homomorphic Encryption (HE). Additive HE supporting homomorphic additions is instantiated from LWE and the ring variant, RLWE [30]. Public parameters

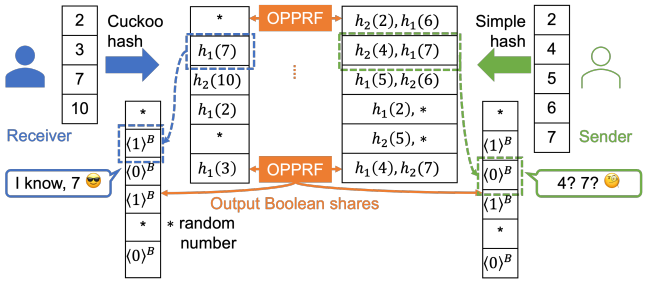


Figure 1. Circuit-PSI asymmetry (cuckoo vs. simple hash): the receiver learns 7 is in the second bucket, but the sender cannot tell which item (4 or 7) is in the second one.

(N, Q) specify the lattice dimension and modulus. Define $\mathcal{R}_Q = \mathbb{Z}_Q[X]/(X^N+1)$. Let χ_{sk} and χ_{err} denote the secret and error distributions (*e.g.*, $\{-1, 0, 1\}^N$).

• **KeyGen**: It generates a key pair $(\mathsf{sk}, \mathsf{pk})$ for RLWE ($\mathsf{sk} \leftarrow \chi_{\mathsf{sk}}$ and $\mathsf{pk} \in \mathcal{R}_Q^2$). Identify the LWE secret key $\mathbf{s} \in \mathbb{Z}_Q^N$ as the coefficient vector of sk, *i.e.*, $\mathbf{s}[i] = \mathsf{sk}[i]$, $\forall i \in [N]$.

• **Encryption**: LWE encryption of $m \in \mathbb{Z}$ is given as $\mathsf{LWE}_{Q,\mathbf{s}}(m) = (\mathbf{a}, b) = (\mathbf{a}, \langle \mathbf{a}, \mathbf{s}\rangle + e + m) \in \mathbb{Z}_Q^{N+1}$ with $\mathbf{a} \leftarrow_{\$} \mathbb{Z}_Q^N$ and $e \leftarrow \chi_{\mathsf{err}}$. RLWE encryption of $\widehat{m} \in \mathcal{R}_Q$ is $\mathsf{RLWE}_{Q,\mathsf{sk}}(\widehat{m}) = (\widehat{a}, -\widehat{a} \cdot \mathsf{sk} + \widehat{e} + \widehat{m}) \in \mathcal{R}_Q \times \mathcal{R}_Q$ (or simply $[\![\widehat{m}]\!]$), where $\widehat{a} \leftarrow_{\$} R_Q$ and $\widehat{e}[i] \leftarrow \chi_{\mathsf{err}}$.

• **Decryption**: An RLWE ciphertext $\widehat{ct} = \mathsf{RLWE}_{Q,\mathsf{sk}}(\widehat{m}) = (\widehat{a}, \widehat{b}) \in \mathcal{R}_Q \times \mathcal{R}_Q$ is decrypted by $\mathsf{RLWE}^{-1}_{Q,\mathsf{sk}}(\widehat{a}, \widehat{b}) = \widehat{a} \cdot \mathsf{sk} + \widehat{b} = \widehat{m} + \widehat{e} \in \mathcal{R}_Q$. (We omit the subscript Q when clear from context.)

• **PackLWEs**: Given a set of LWE ciphertexts $\{\mathsf{LWE}_{\mathbf{s}}(m_i)\}$, an FFT-style automorphism-and-keyswitch procedure can pack them into a single RLWE ciphertext [20]. For $\widehat{m}(X)$, apply the automorphism $\mathsf{EvalAuto}(\mathsf{RLWE}_{\widehat{\mathsf{sk}}}(\widehat{m}(X)), t)$ that applies $X \mapsto X^t$ to both ciphertext and key, then perform

$$\mathsf{KeySwitch}_{\widehat{\mathsf{sk}}_t \to \widehat{\mathsf{sk}}}\colon \mathsf{RLWE}_{\widehat{\mathsf{sk}}_t}(\widehat{m}_t) \longmapsto \mathsf{RLWE}_{\widehat{\mathsf{sk}}}(\widehat{m}_t),$$

where $\widehat{\mathsf{sk}}_t = \widehat{\mathsf{sk}}(X^t)$ and $\widehat{m}_t = \widehat{m}(X^t)$.

• **Arithmetic share to HE**: $\mathcal{F}_{\mathsf{A2H}}$. It converts $\langle \mathbf{a}\rangle^{\mathsf{A}}$ to an RLWE ciphertext $[\![\mathbf{a}]\!]$. P_l transforms $\langle \mathbf{a}[i]\rangle_l^{\mathsf{A}}$ from $\mathbb{Z}_{2^\ell}$ to $\mathbb{Z}_Q$, then encodes it to the polynomial $\widehat{a}_l = \sum_{i=0}^{N-1} \langle \mathbf{a}[i]\rangle_l^{\mathsf{A}} X^i$. P_c sends $\mathsf{RLWE}_{\mathsf{sk}_c}(\widehat{a}_c)$ to P_{1-c} encrypted under sk_c. P_{1-c} gets $[\![\mathbf{a}]\!]$ by homomorphically adding $\widehat{a}_{1-c}$ to $\mathsf{RLWE}_{\mathsf{pk}_c}(\widehat{a})$ [5].

• **HE to Arithmetic share**: $\mathcal{F}_{\mathsf{H2A}}$. It converts RLWE $[\![\mathbf{a}]\!]$ encrypted by P_c to $\langle \mathbf{a}\rangle_l^{\mathsf{A}}$. P_{1-c} samples a random $\widehat{r} \leftarrow_{\$} \mathcal{R}_Q$, and gets $\widehat{ct}$ by homomorphically subtracting it from $[\![\mathbf{a}]\!]$. P_c decrypts $\widehat{ct}$ and sets its coefficient vector as $\langle \mathbf{a}\rangle_c^{\mathsf{A}}$. P_{1-c} sets $\langle \mathbf{a}[i]\rangle_{1-c}^{\mathsf{A}}$ by transforming $\widehat{r}[i]$ from Q to $\mathbb{Z}_{2^\ell}$ [5].

3. System and Security Models

Two parties P_0 and P_1 hold vertically partitioned data $(\mathbf{ID}_l, \mathbf{X}_l)$ for $l \in \{0, 1\}$, where $\mathbf{ID}_l$ draws from a common identifier domain. Identifier lists may overlap, while feature spaces are disjoint: $\mathbf{X}_l \in \mathbb{R}^{n_l \times m_l}$ with distinct column sets. For simplicity, we assume $m_0 = m_1 = m$ and $n_0 = n_1 = n$

Functionality $\mathcal{F}_{\mathsf{GBDT}}(\mathbf{Input}_0^{\Pi}, \mathbf{Input}_1^{\Pi}, \mathbf{pp})$

$\mathbf{Input}_0^{\Pi} = \{\mathbf{ID}_0, \mathbf{X}_0, \mathbf{y}\}, \mathbf{Input}_1^{\Pi} = \{\mathbf{ID}_0, \mathbf{X}_1\}$
$\mathbf{Output}_0^{\Pi} = \{\mathbf{C}_0, \{\langle \mathbf{w}[k] \rangle_0^{\mathsf{A}}\}_{k \geq 2^{D-1}}\}$, where
$\mathbf{C}_0[k] = (z_*^{(k)}, u_*^{(k)})$ if $z_*^{(k)} < m$; otherwise, $\mathbf{C}_0[k] = \perp$
$\mathbf{Output}_1^{\Pi} = \{\mathbf{C}_1, \{\langle \mathbf{w}[k] \rangle_1^{\mathsf{A}}\}_{k \geq 2^{D-1}}\}$, where
$\mathbf{C}_1[k] = (z_*^{(k)}, u_*^{(k)})$ if $z_*^{(k)} \geq m$; otherwise, $\mathbf{C}_1[k] = \perp$

Figure 2. Ideal private GBDT training functionality: Outputs reveal only per-party best splits $\mathbf{C}$ and secret-shared weights $\langle \mathbf{w} \rangle^{\mathsf{A}}$.

(unbalanced $n_0 \neq n_1$ are considered in experiments). Let $\mathbf{ID}_l[i]$ be the i-th identifier of P_l and $\mathbf{X}_l[i]$ its associated feature vector. The parties run a two-party protocol Π to train a GBDT on the (implicit) join of their datasets.

All prior work [5], [7], [8] assumes pre-aligned data, *i.e.*, $\mathbf{X}_0[i]$ and $\mathbf{X}_1[i]$ refer to the same sample. This implicitly discloses $\mathbf{ID}_0 \cap \mathbf{ID}_1$, (in the worst case, all identifiers), enabling re-identification. In contrast, our designs reveal *no* identifiers, *no* membership in the intersection, *not* even the cardinality $|\mathbf{ID}_0 \cap \mathbf{ID}_1|$ nor whether the intersection equals the entire dataset; it also hides all alignment information (*e.g.*, for a common item at $\mathbf{ID}_0[i]$, the counterparty's index j such that $\mathbf{ID}_0[i] = \mathbf{ID}_1[j]$ is never learned).

Figure 2 specifies the ideal functionality $\mathcal{F}_{\mathsf{GBDT}}$ for one tree in $\mathcal{T}$ with two parties' data and public parameters $\mathbf{pp}$. Without loss of generality, P_0 supplies labels $\mathbf{y}$. Each tree is complete and balanced with depth D; internal nodes are indexed by $1 \leq k < 2^{D-1}$ and leaves by $2^{D-1} \leq k < 2^D$. Internal computation is done on the implicit join $\mathbf{ID}_0 \cap \mathbf{ID}_1$: Gradients/Hessians are computed with sigmoid approximation and then aggregated as histograms per bin (Section 2.1). For each internal node k and candidate (z, u), the split gain $\mathbf{L}_{\mathsf{sp}}$ is evaluated as in Eq. (2); the best split $(z_*^{(k)}, u_*^{(k)})$ maximizing $\mathbf{L}_{\mathsf{sp}}$ is revealed to P_l owning $z_*^{(k)}$. Given the split, sample indicators update via exclusive-prefix matrices. Upon completion, P_l receives its own best splits $\mathbf{C}_l[k]$ for $k < 2^{D-1}$, and the leaf weights are additively secret-shared.

We consider static and semi-honest probabilistic polynomial time (PPT) adversaries [5], [7], [8], which follow the protocol without deviation but try to learn more beyond the outputs. Formally, let Π be a two-party protocol computing a deterministic functionality $\mathcal{F}\colon \{0,1\}^* \times \{0,1\}^* \mapsto \{0,1\}^* \times \{0,1\}^*$. The output of $\mathcal{F}$ is a pair, $\mathcal{F}_0(x_0, x_1)$ to P_0 and $\mathcal{F}_1(x_0, x_1)$ to P_1. For $i \in \{0,1\}$, the view of P_i during an execution of Π on (x_0, x_1) is denoted by $\mathcal{V}_i^{\Pi}(x_0, x_1)$.

Definition 3.1. For a function $\mathcal{F}$, Π privately computes $\mathcal{F}$ if there are PPT algorithms $\mathcal{S}_0$ and $\mathcal{S}_1$ such that, $i \in \{0,1\}$,

$$\mathcal{S}_i(x_i, \mathcal{F}_i(x_0, x_1)) \approx_c \mathcal{V}_i^{\Pi}(x_0, x_1), \forall x_0, x_1 \in \{0,1\}^*,$$

where $\approx_c$ denotes computational indistinguishability.

Theorem 3.1 (Modular Composition [31])**.** *Let $\mathcal{F}_1, \ldots, \mathcal{F}_m$ be ideal two-party functionalities, and let Π be a two-party protocol in the hybrid model that can invoke at most one of $\mathcal{F}_1, \ldots, \mathcal{F}_m$ per round. For $i \in [m]$, let f_i be a real two-party protocol that realizes $\mathcal{F}_i$ in the computational setting. Then, for any PPT passive adversary $\mathcal{A}$ and any PPT environment $\mathcal{Z}$, there exists a PPT passive simulator $\mathcal{S}$ such that* $\mathsf{Sim}_{\Pi,\mathcal{S},\mathcal{Z}}^{\mathcal{F}_1,\mathcal{F}_2,\ldots,\mathcal{F}_m} \approx_c \mathsf{Exec}_{\Pi^{(f_1,f_2,\ldots,f_m)},\mathcal{A},\mathcal{Z}}$.

4. AnonGBDT: Base Design

Squirrel [5], a recent two-party secure GBDT protocol, presumes a priori alignment: $\mathbf{X}_0[i]$ and $\mathbf{X}_1[i]$ are the same sample i, enabling *direct* training on $\mathbf{X}_0 \,\|\, \mathbf{X}_1$. In practice, independently maintained data only partially overlap; perfect alignment is uncommon, rendering Squirrel impractical without a preceding private-matching step.

Circuit-PSI is utilized to reconcile $\mathbf{ID}_0$ and $\mathbf{ID}_1$ before training, without revealing identifiers. Subsequent training then follows Squirrel's workflow (using primitives in Section 2.2), with targeted adaptations to accommodate circuit-PSI's asymmetric hashing and payload interface. A brief recap of Squirrel is first given, preceding our base design. We follow their convention of underlining, *e.g.*, $\underline{sk}$ denotes a secret key under the lifted HE parameters such as $\underline{N}$.

4.1. Revisit of Squirrel

Data pre-processing: Each party P_l discretizes raw features $\mathbf{X}_l$ into a one-hot bin-indicator matrix $\mathbf{M}_l$ and its exclusive-prefix variant $\widetilde{\mathbf{M}}_l$, as detailed in Section 2.1.

Private sample tracking: At each node k, an n-bit *indicator* $\mathbf{b}^{(k)}$ (initialized to $\mathbf{1}^n$ at the root) is maintained, where $\mathbf{b}^{(k)}[i] = 1$ iff sample i reaches node k. To keep $\mathbf{b}^{(k)}$ private, Squirrel uses "AND-style" sharing[4]: $\mathbf{b}^{(k)} = \mathbf{b}_0^{(k)} \wedge \mathbf{b}_1^{(k)}$.

If party P_c[5] owns the best split (z_*, u_*), it locally forms the (plaintext) assignment $\mathbf{b}_*^{(k)} = \widetilde{\mathbf{M}}_c[z_* B + u_*]$ and updates

$$\mathbf{b}_c^{(2k)} = \mathbf{b}_c^{(k)} \wedge \mathbf{b}_*^{(k)}, \quad \mathbf{b}_c^{(2k+1)} = \mathbf{b}_c^{(k)} \oplus \mathbf{b}_c^{(2k)}, \quad (4)$$

while P_{1-c} copies its share without $\mathbf{b}_*^{(k)}$. This AND-sharing update is incompatible with XOR shares produced after circuit-PSI alignment; Section 4.2.2 introduces an *oblivious indicator synchronization* (OIS) for XOR-shared $\mathbf{b}^{(k)}$.

Gradient computation: Fourier-series sigmoid [32] and other nonlinear functions [33] have been studied for efficient and accurate secure computation. To obtain arithmetic shares $\langle \mathbf{g} \rangle_l^{\mathsf{A}}$ and $\langle \mathbf{h} \rangle_l^{\mathsf{A}}$ (Eq. (1)), Squirrel adopts a Fourier-series sigmoid approximation that reduces secure multiplications (hence communication) compared to spline-based methods [7], [34], yet still contributes $\approx 50\%$ of total traffic.

Specifically, for $x \in [-\mu, \mu]$, it evaluates $a_0 + \sum_{j=1}^{J} a_j \sin\left(2\pi j x / 2^{L+1}\right)$, with $\mu = 5.6$, degree $J = 8$, scaling factor $L = \lceil \log_2 \mu \rceil$, and maximal error 0.022; outside this range, outputs are clipped to 0 (for $x < -\mu$) or 1 (for $x > \mu$). This requires only secure comparisons with $\pm\mu$ and secure multiplications for the series. Section 4.2.3

4. This sharing cannot "perfectly" hide $\mathbf{b}^{(k)}$: If a party's share equals 0, it immediately learns the corresponding secret bit is 0.

5. P_c: asymmetric steps; only P_c acts. P_l: symmetric steps; both act.

optimizes such an approximation by lowering J and reusing primitives in $\pm\mu$ comparisons.

Gradient aggregation: We describe the processing for $\mathbf{g}$ (it applies identically to $\mathbf{h}$). Non-reachable samples are filtered by $\mathbf{g}^{(k)} = \mathbf{b}^{(k)} \odot \mathbf{g}$, where $\odot$ is element-wise multiplication, realized via a secure multiplexer:

$$\langle \mathbf{g}^{(k)} \rangle_l^{\mathsf{A}} = \mathcal{F}_{\mathsf{mux}}(\langle \mathbf{b}^{(k)} \rangle_l^{\mathsf{B}}, \langle \mathbf{g} \rangle_l^{\mathsf{A}}). \tag{5}$$

For the left child $2k$, Squirrel halves the usage of OT based on $\mathbf{g}^{(2k)} = \mathbf{b}^{(2k)} \odot \mathbf{g} = (\mathbf{b}^{(k)} \wedge \mathbf{b}_*^{(k)}) \odot \mathbf{g} = \mathbf{b}_*^{(k)} \odot \mathbf{g}^{(k)}$ given non-shared $\mathbf{b}_*^{(k)}$ (vs. simply using $\mathcal{F}_{\mathsf{mux}}$ over AND-shared $\mathbf{b}^{(2k)}$). The right child is $\mathbf{g}^{(2k+1)} = \mathbf{g}^{(k)} - \mathbf{g}^{(2k)}$.

Filtered gradients are then binned by *pick-then-sum*: $\widetilde{\mathbf{g}}^{(k,c)}[j] = \sum_{i|\mathbf{M}_c[j,i]=1} \mathbf{g}^{(k)}[i]$ (for $i \in [n], j \in [1, mB]$), or equivalently $\mathbf{M}_c \cdot \mathbf{g}^{(k)}$, which is realized by the secure binary-matrix-vector multiplication $\mathcal{F}_{\mathsf{BinMatVec}}$:
i) convert $\langle \mathbf{g}^{(k)} \rangle_l^{\mathsf{A}}$ to RLWE-encrypted $[\![\mathbf{g}^{(k)}]\!]$ by $\mathcal{F}_{\mathsf{A2H}}$,
ii) achieve efficient homomorphic aggregation on extracted LWE ciphertext from $[\![\mathbf{g}^{(k)}]\!]$, and pack to $[\![\widetilde{\mathbf{g}}^{(k)}]\!]$ within the RLWE domain via PackLWEs. which we optimize;
iii) convert $[\![\widetilde{\mathbf{g}}^{(k)}]\!]$ back to shares $\langle \widetilde{\mathbf{g}}^{(k)} \rangle_l^{\mathsf{A}}$ via $\mathcal{F}_{\mathsf{H2A}}$.

In our anonymous setting, the circuit-PSI sender does not know the correspondence between $\mathbf{M}[:, i]$ and $\mathbf{g}^{(k)}[i]$, making $\mathcal{F}_{\mathsf{BinMatVec}}$ inapplicable as is; Section 4.2.1 addresses this by treating $\mathbf{M}$ as an additional payload.

Best-split selection: For each candidate (z, u), the split gain $\mathbf{L}_{\mathsf{sp}}$ (Eq. (2)) is securely evaluated using $\mathcal{F}_{\mathsf{div}}$ and $\mathcal{F}_{\mathsf{mul}}$. $\mathcal{F}_{\mathsf{argmax}}$ reveals the maximizer (z_*, u_*) to its owner.

4.2. Our Base Design

We begin with the stash-less circuit-PSI [15], using P_0 as receiver and P_1 as sender, to privately reconcile $\mathbf{ID}_0$ and $\mathbf{ID}_1$. The output is XOR-shared $\mathbf{b}^{(1)} \in \{0,1\}^{(1+\varepsilon)n}$ over the receiver's cuckoo-hash order (Section 2.2.3), replacing the $\mathbf{1}^n$ root indicator in Squirrel. Subsequent training avoids leaky AND-sharing by maintaining XOR shares.

Encoding the entire training pipeline as one circuit with payloads $(\mathbf{M}_0, \widetilde{\mathbf{M}}_0, \mathbf{y})$ and $(\mathbf{M}_1, \widetilde{\mathbf{M}}_1)$ is impractical. We minimize payloads and exploit circuit-PSI's order alignment to enable efficient, order-dependent steps.

4.2.1. Gradient Histogram Aggregation. Order after circuit-PSI aligns elements for the receiver: the cuckoo-hash permutation fixes a one-to-one correspondence between the intersection positions and local samples. Concretely:

Receiver side P_0. As $\mathbf{M}_0$ is aligned (in cuckoo order) with samples' gradient shares from $\mathcal{F}_{\mathsf{sigmoid}}$ and $\mathcal{F}_{\mathsf{mul}}$ (Line 4, Figure 4), P_0 can build histograms with the communication-friendly $\mathcal{F}_{\mathsf{BinMatVec}}$ as in Squirrel; only $\mathbf{y}$ is payload.

Sender side P_1. The sender's simple hashing creates a one-to-many mapping, so it does not learn which gradient index matches which column of $\mathbf{M}_1$. To restore compatibility, $\mathbf{M}_1$ is carried as a circuit-PSI payload, yielding XOR-shared

Functionality $\mathcal{F}_{\mathsf{OIS}}$

P_0**'s Input:** $\langle \widetilde{\mathbf{M}}_1 \rangle_0^{\mathsf{B}}$, $\langle \mathbf{b} \rangle_0^{\mathsf{B}}$, $\widetilde{\mathbf{M}}_0$, and $\{(z_*, u_*)\ /\ \perp\}$
P_1**'s Input:** $\langle \widetilde{\mathbf{M}}_1 \rangle_1^{\mathsf{B}}$, $\langle \mathbf{b} \rangle_1^{\mathsf{B}}$, and $\{\perp\ /(z_*, u_*)\}$
Output: $\langle \mathbf{b}^L \rangle_l^{\mathsf{B}}$ and $\langle \mathbf{b}^R \rangle_l^{\mathsf{B}}$ for 'L'eft and 'R'ight children.

Protocol Π_{OIS}

1: P_0: draw a random mask $\langle \mathbf{b}_* \rangle_0^{\mathsf{B}} := \mathbf{r} \leftarrow_{\$} \{0,1\}^{(1+\epsilon)n}$.
2: **if** P_0 owns (z_*, u_*) **then**
3: $\quad P_0$: set $\mathbf{b}_* = \widetilde{\mathbf{M}}_0[z_* B + u_*]$.
4: $\quad P_0$: share $\langle \mathbf{b}_* \rangle_1^{\mathsf{B}} = \mathbf{b}_* \oplus \mathbf{r}$ with P_1.
5: **else**
6: $\quad$ **for** $i \in [(1+\epsilon) \cdot n]$ **do**
7: $\quad\quad P_0$: $\widetilde{\mathbf{r}}[j] = \mathbf{r}[i] \oplus \langle \widetilde{\mathbf{M}}_1 \rangle_0^{\mathsf{B}}[j, i], \forall j \in [m \cdot B]$.
8: $\quad\quad P_0$ (as OT sender): run $\mathcal{F}_{\mathsf{OT}}(\widetilde{\mathbf{r}})$.
9: $\quad\quad P_1$ (as OT receiver): $\mathbf{r}'[i] \leftarrow \mathcal{F}_{\mathsf{OT}}(z_* B + u_*)$
10: $\quad\quad P_1$: $\langle \mathbf{b}_*[i] \rangle_1^{\mathsf{B}} = \mathbf{r}'[i] \oplus \langle \widetilde{\mathbf{M}}_1 \rangle_1^{\mathsf{B}}[z_* B + u_*, i]$.
11: P_l computes $\langle \mathbf{b}^L \rangle_l^{\mathsf{B}} = \mathcal{F}_{\mathsf{and}}(\langle \mathbf{b}_* \rangle_l^{\mathsf{B}}, \langle \mathbf{b} \rangle_l^{\mathsf{B}})$ and $\langle \mathbf{b}^R \rangle_l^{\mathsf{B}} = \langle \mathbf{b}^L \rangle_l^{\mathsf{B}} \oplus \langle \mathbf{b} \rangle_l^{\mathsf{B}}$. ▷ Secure evaluation of Eq. (4)

Figure 3. Ideal functionality and protocol of OIS

entries $\langle \mathbf{M}_1 \rangle^{\mathsf{B}}$ aligned to the receiver's order. Gradients are then aggregated by pick-then-sum using secure multiplexers:

$$\forall (z, u) \colon \textstyle\sum_i \mathcal{F}_{\mathsf{mux}} \left(\langle \mathbf{M}_1[zB + u, i] \rangle_l^{\mathsf{B}}, \langle \mathbf{g}[i] \rangle_l^{\mathsf{A}} \right), \tag{6}$$

and likewise for $\mathbf{h}$. This avoids generic secure matrix multiplication (which would invoke $\mathcal{F}_{\mathsf{mul}}$ per entry) while using aligned order: $\langle \mathbf{M}_1[\cdot, i] \rangle_l^{\mathsf{B}}$, $\langle \mathbf{g}[i] \rangle_l^{\mathsf{A}}$ refer to the same sample.

Root filtering uses $\mathbf{b}^{(1)}$: both parties compute $\langle \mathbf{g}^{(1)} \rangle_l^{\mathsf{A}} = \mathcal{F}_{\mathsf{mux}}(\langle \mathbf{b}^{(1)} \rangle_l^{\mathsf{B}}, \langle \mathbf{g} \rangle_l^{\mathsf{A}})$ and similarly for $\mathbf{h}$ (Line 5, Figure 4).

Complexity: For $\mathbf{M}_0$, $\mathcal{O}(Bmn)$ invocations of $\mathcal{F}_{\mathsf{mux}}$, each using two $\mathcal{F}_{\mathsf{OT}}$ calls (that send $\mathcal{O}(\ell)$ bits per OT); the total communication round is $\mathcal{O}(1)$. For $\mathbf{M}_1$, $\mathcal{F}_{\mathsf{BinMatVec}}$ incurs $\mathcal{O}(n \log Q)$ bits via $\mathcal{F}_{\mathsf{A2H}}$ and $\mathcal{O}(2Bm \log Q)$ bits via $\mathcal{F}_{\mathsf{H2A}}$; the communication uses two rounds.

4.2.2. Oblivious Indicator Synchronization. Sample indicators are updated from the chosen split via the exclusive-prefix matrix $\widetilde{\mathbf{M}}$. As a circuit-PSI receiver, P_0 knows which column of $\widetilde{\mathbf{M}}_0$[6] matches the correct indicator index. The sender P_1 does not; it needs to carry $\widetilde{\mathbf{M}}_1$ as a payload.

Because both $\widetilde{\mathbf{M}}_1$ and indicators $\mathbf{b}^{(k)}$ are XOR-shared, the plaintext (local) update Eq. (4) is inapplicable. To this end, we introduce an OT-based *oblivious indicator synchronization* (OIS) protocol, whose functionality $\mathcal{F}_{\mathsf{OIS}}$ appears in Figure 3: If P_0 owns (z_*, u_*), it secret-shares $\mathbf{b}_* = \widetilde{\mathbf{M}}_0[z_* B + u_*]$ with P_1 (Lines 1–4). Otherwise, P_0 (as OT sender) masks its share of $\widetilde{\mathbf{M}}_1$ and, for *each* row, the two parties execute a 1-out-of-mB OT such that P_1 (as OT receiver) obtains correct $\langle \mathbf{b}_* \rangle_1^{\mathsf{B}}$ given its private choice (z_*, u_*). In either case, the left-child indicator is then computed with secure AND $\mathcal{F}_{\mathsf{and}}$, and the right-child indicator by bitwise XOR (Section 4.1).

6. It is resized to $mB \times (1+\varepsilon)n$ after circuit-PSI cuckoo hashing.

Base AnonGBDT $\Pi^{\mathsf{Base}}_{\mathsf{GBDT}}(\mathbf{Input}^{\Pi}_0, \mathbf{Input}^{\Pi}_1, \mathbf{pp})$

P_0**'s Input:** Identifier lists $\mathbf{ID}_0$, raw feature matrix $\mathbf{X}_0$, and labels $\mathbf{y}$ (Initially, $\tilde{\mathbf{y}} = \mathbf{0}$).
P_1**'s Input:** Identifier lists $\mathbf{ID}_1$ and raw feature matrix $\mathbf{X}_1$, and $\underline{\mathsf{sk}}_1$ for $\mathcal{F}_{\mathsf{A2H}}$, sk_1 for $\mathcal{F}_{\mathsf{H2A}}$.
Public parameters: $\mathbf{pp} = \{T, D, B\}$, and lifting key $\mathsf{LK}_{\underline{\mathsf{sk}}_1 \to \mathsf{sk}_1}$ for KeySwitch.
Output: Output^{Π}_0 for P_0 and Output^{Π}_1 for P_1.

P_l discretizes $\mathbf{X}_l$ to $\mathbf{M}_l \in \{0,1\}^{m \cdot B \times n}$, which is aggregated to $\widetilde{\mathbf{M}}_l$. ▷ Data pre-processing as in Squirrel
P_l: $\langle \mathbf{b}^{(1)} \rangle^{\mathsf{B}}_l, \langle \mathbf{y} \rangle^{\mathsf{A}}_l, \langle \mathbf{M}_1 \rangle^{\mathsf{B}}_l, \langle \widetilde{\mathbf{M}}_1 \rangle^{\mathsf{B}}_l \leftarrow \mathcal{F}_{\mathsf{CPSI}}(\mathbf{ID}_0; \mathbf{ID}_1)$, with P_0's payload $\mathbf{y}$ and P_1's payloads $(\mathbf{M}_1, \widetilde{\mathbf{M}}_1)$.
for tree $t \in \mathcal{T}_t$ **do**
  P_l computes gradients $\langle \mathbf{g} \rangle^{\mathsf{A}}_l, \langle \mathbf{h} \rangle^{\mathsf{A}}_l$ using $\mathcal{F}_{\mathsf{sigmoid}}(\langle \tilde{\mathbf{y}} \rangle^{\mathsf{A}}_l)$ and $\mathcal{F}_{\mathsf{mul}}$ based on Eq. (1). ▷ Per-sample gradient
  P_l filters gradients $\langle \mathbf{g}^{(1)} \rangle^{\mathsf{A}}_l, \langle \mathbf{h}^{(1)} \rangle^{\mathsf{A}}_l$ at the root using $\mathcal{F}_{\mathsf{mux}}$ given $\langle \mathbf{b}^{(1)} \rangle^{\mathsf{B}}_l$ based on Eq. (5).
  for internal nodes $k \in \{1, 2, \ldots, 2^{D-1} - 1\}$ **do**
    if k is 1 or even **then** ▷ Per-bin gradient
      P_l aggregates gradients over $\mathbf{M}_0$ (as in Squirrel):
      $\langle \widetilde{\mathbf{g}}^{(k,0)} \rangle^{\mathsf{A}}_l \| \langle \widetilde{\mathbf{h}}^{(k,0)} \rangle^{\mathsf{A}}_l = \mathcal{F}_{\mathsf{BinMatVec}}(\{\langle \mathbf{g}^{(k)} \rangle^{\mathsf{A}}_0 \| \langle \mathbf{h}^{(k)} \rangle^{\mathsf{A}}_0, \mathbf{M}_0\}, \{\langle \mathbf{g}^{(k)} \rangle^{\mathsf{A}}_1 \| \langle \mathbf{h}^{(k)} \rangle^{\mathsf{A}}_1, \mathsf{sk}_1\})$.
      P_l aggregates gradients over $\langle \mathbf{M}_1 \rangle^{\mathsf{B}}_l$ using our $\mathcal{F}_{\mathsf{mux}}$-based approach (see Eq. (6)):
      $\forall j \in [mB]$: $\langle \widetilde{\mathbf{g}}^{(k,1)}[j] \rangle^{\mathsf{A}}_l (/ \langle \widetilde{\mathbf{h}}^{(k,1)}[j] \rangle^{\mathsf{A}}_l) = \sum_i \mathcal{F}_{\mathsf{mux}}(\langle \mathbf{M}_1[j,i] \rangle^{\mathsf{B}}_l, \langle \mathbf{g}^{(k)}[i] \rangle^{\mathsf{A}}_l (/ \langle \mathbf{h}^{(k)}[i] \rangle^{\mathsf{A}}_l))$.
      P_l locally concatenates the shares as $\langle \widetilde{\mathbf{g}}^{(k)} \rangle^{\mathsf{A}}_l = \langle \widetilde{\mathbf{g}}^{(k,0)} \rangle^{\mathsf{A}}_l \| \langle \widetilde{\mathbf{g}}^{(k,1)} \rangle^{\mathsf{A}}_l$, $\langle \widetilde{\mathbf{h}}^{(k)} \rangle^{\mathsf{A}}_l = \langle \widetilde{\mathbf{h}}^{(k,0)} \rangle^{\mathsf{A}}_l \| \langle \widetilde{\mathbf{h}}^{(k,1)} \rangle^{\mathsf{A}}_l$.
    else P_l gets $\langle \widetilde{\mathbf{g}}^{(k)} \rangle^{\mathsf{A}}_l = \langle \widetilde{\mathbf{g}}^{(k/2)} \rangle^{\mathsf{A}}_l - \langle \widetilde{\mathbf{g}}^{(k-1)} \rangle^{\mathsf{A}}_l$, $\langle \widetilde{\mathbf{h}}^{(k)} \rangle^{\mathsf{A}}_l = \langle \widetilde{\mathbf{h}}^{(k/2)} \rangle^{\mathsf{A}}_l - \langle \widetilde{\mathbf{h}}^{(k-1)} \rangle^{\mathsf{A}}_l$ (as in Squirrel).
    P_l computes the splitting scores $\langle \mathbf{L}^{(k)}_{\mathsf{sp}} \rangle^{\mathsf{A}}_l$ using $\mathcal{F}_{\mathsf{mul}}$ and $\mathcal{F}_{\mathsf{div}}$ for all (z, u) based on Eq. (2).
    P_l gets $\langle z^{(k)}_* \rangle^{\mathsf{A}}_l, \langle u^{(k)}_* \rangle^{\mathsf{A}}_l \leftarrow \mathcal{F}_{\mathsf{argmax}}(\langle \mathbf{L}^{(k)}_{\mathsf{sp}} \rangle^{\mathsf{A}}_l)$. ▷ Best split selection
    Open a bit $c \leftarrow \mathcal{F}_{\mathsf{greater}}(\langle z^{(k)}_* \rangle^{\mathsf{A}}_l, m-1)$ and reveal $(z^{(k)}_*, u^{(k)}_*)$ to P_c, who writes it to Output^{Π}_c.
    P_l updates $\langle \mathbf{b}^{(2k)} \rangle^{\mathsf{B}}_l, \langle \mathbf{b}^{(2k+1)} \rangle^{\mathsf{B}}_l \leftarrow \mathcal{F}_{\mathsf{OIS}}(\{\langle \widetilde{\mathbf{M}}_1 \rangle^{\mathsf{B}}_0, \langle \mathbf{b}^{(k)} \rangle^{\mathsf{B}}_0, \widetilde{\mathbf{M}}_0\}; \{\langle \widetilde{\mathbf{M}}_1 \rangle^{\mathsf{B}}_1, \langle \mathbf{b}^{(k)} \rangle^{\mathsf{B}}_1\})$ with P_c's $(z^{(k)}_*, u^{(k)}_*)$
    P_l updates left-child gradients: $\langle \mathbf{g}^{(2k)} \rangle^{\mathsf{A}}_l = \mathcal{F}_{\mathsf{mux}}(\langle \mathbf{b}^{(2k)} \rangle^{\mathsf{B}}_l, \langle \mathbf{g}^{(k)} \rangle^{\mathsf{A}}_l)$, $\langle \mathbf{h}^{(2k)} \rangle^{\mathsf{A}}_l = \mathcal{F}_{\mathsf{mux}}(\langle \mathbf{b}^{(2k)} \rangle^{\mathsf{B}}_l, \langle \mathbf{h}^{(k)} \rangle^{\mathsf{A}}_l)$.
    P_l updates right-child gradients: $\langle \mathbf{g}^{(2k+1)} \rangle^{\mathsf{A}}_l = \langle \mathbf{g}^{(k)} \rangle^{\mathsf{A}}_l - \langle \mathbf{g}^{(2k)} \rangle^{\mathsf{A}}_l$, $\langle \mathbf{h}^{(2k+1)} \rangle^{\mathsf{A}}_l = \langle \mathbf{h}^{(k)} \rangle^{\mathsf{A}}_l - \langle \mathbf{h}^{(2k)} \rangle^{\mathsf{A}}_l$.
  P_l computes leaf weights $\langle \mathbf{w} \rangle^{\mathsf{A}}_l$ using $\mathcal{F}_{\mathsf{div}}$ based on Eq. (3) and writes them in Output^{Π}_l.
  P_l updates $\langle \tilde{\mathbf{y}}[i] \rangle^{\mathsf{A}}_l$ by accumulating the result of $\mathcal{F}_{\mathsf{mux}}(\langle \mathbf{b}^{(k)}[i] \rangle^{\mathsf{B}}_l, \langle \mathbf{w}[k] \rangle^{\mathsf{A}}_l)$ $(\forall i)$ over all leaves k.

Figure 4. Base AnonGBDT (with $\mathcal{F}_{\mathsf{mux}}$-based gradient histogram aggregation and new $\mathcal{F}_{\mathsf{OIS}}$ for sample indicator updates)

Complexity: If P_0 holds the best split, it sends $\mathcal{O}(n\ell)$ bits and the round is 1; if P_1 holds the best split, they invoke n 1-out-of-mB $\mathcal{F}_{\mathsf{OT}}$ (each sends $\mathcal{O}(mB\ell)$ bits) with total communication round of $\mathcal{O}(\log n)$), and n $\mathcal{F}_{\mathsf{and}}$ (each sends $O(\ell)$ bits) with total communication round of $\mathcal{O}(\log n + 2)$.

4.2.3. Improved Sigmoid Approximation. We replace the integral-derived Fourier coefficients with least-squares fitting to estimate $a_{j \in [0,J]}$, allowing us to reduce the order to $J = 3$ while achieving better accuracy and efficiency. We retain $\mu = 5.6$ and evaluate only with $\mathcal{F}_{\mathsf{mul}}$, using an implementation optimized for trigonometric inputs within $[-1, 1]$; see Appendix A.1. The resulting average/maximum errors are 0.002/0.015 (vs. 0.022 in Squirrel); see Appendix A.2.

We also streamline secure comparisons with $\pm\mu$. Under fixed-point representation with precision 2^f, the two thresholds share their $f - \delta$ least-significant bits; hence, within $\mathcal{F}_{\mathsf{greater}}$, we reuse $\mathcal{F}_{\mathsf{OT}}$ and $\mathcal{F}_{\mathsf{and}}$ on the shared bits. Overall, our new sigmoid costs **33%** of runtime and **16%** of communication of the Squirrel variant.

Figure 4 presents the base AnonGBDT workflow, with differences from Squirrel highlighted in blue. Two substitutions enable anonymous training under circuit-PSI alignment: i) gradient-histogram aggregation at the sender via $\mathcal{F}_{\mathsf{mux}}$ over payload-shared $\mathbf{M}_1$ (Section 4.2.1); ii) indicator synchronization via $\mathcal{F}_{\mathsf{OIS}}$ (Section 4.2.2), ensuring XOR-shared indicators remain aligned and usable by both parties.

5. AnonGBDT: OT-light, Symmetric Alignment

The base design conceals intersection membership, cardinality, and alignment information throughout training, offering *stronger* privacy than prior work [5], [7], [8]. Its communication remains substantial: i) histogram construction via $\mathcal{F}_{\mathsf{mux}}$ requires $O(mnB)$ many 1-out-of-2 OTs, and ii) $\mathcal{F}_{\mathsf{OIS}}$ runs one 1-out-of-mB per row over $(1+\varepsilon)n$ rows (with ε the cuckoo-hash parameter). The histogram term dominates. We eliminate it via a dual-circuit-PSI framework.

5.1. Our *Dual*-circuit-PSI Framework

Circuit-PSI is *asymmetric*: only the receiver learns the (bucketed) input-output correspondence. We "symmetrize" this by running two circuit-PSI instances *in parallel* with swapped roles, so each party is a receiver once and learns the correspondence for exactly one instance.

This allows both parties to keep $\mathbf{M}_0$ and $\mathbf{M}_1$ locally (no payload for $\mathbf{M}_1$), and to aggregate gradients on non-shared matrices solely with $\mathcal{F}_{\mathsf{BinMatVec}}$ (as in Squirrel), removing the many calls to $\mathcal{F}_{\mathsf{mux}}$. We further accelerate $\mathcal{F}_{\mathsf{BinMatVec}}$

by replacing its subroutines, PackLWEs-then-LWEDimLift, with our FastPackLWEs (see Section 5.2).

Figure 5 overviews the dual-circuit-PSI. Superscripts $(k,0)$ and $(k,1)$ respectively denote the first and second instances at node k. In instance 0, P_0 (receiver) inputs $\mathbf{ID}_0$ with payload $\mathbf{y}$, yielding $\langle \mathbf{b}^{(1,0)} \rangle^{\mathsf{B}}$ and $\langle \mathbf{y}^0 \rangle^{\mathsf{A}}$. Roles are reversed in instance 1, producing $\langle \mathbf{b}^{(1,1)} \rangle^{\mathsf{B}}$ and $\langle \mathbf{y}^1 \rangle^{\mathsf{A}}$.

Each instance applies its own cuckoo permutation, so *order-dependent* steps must be done per instance, *e.g.*, gradient computation uses both $\langle \mathbf{y}^0 \rangle^{\mathsf{A}}$ and $\langle \mathbf{y}^1 \rangle^{\mathsf{A}}$, whereas (order-independent) aggregation runs once locally over $\mathbf{M}_0$ or $\mathbf{M}_1$.

Base OIS limitation: With dual-circuit-PSI, updating the two XOR-shared indicators $\mathbf{b}^{(k,0)}$ and $\mathbf{b}^{(k,1)}$ is non-trivial. If the split owner P_c is the *receiver* for an instance, it can update as in base OIS (Lines 1–4 and 11, Figure 3). If it is the sender for the other instance, it lacks the share of $\widetilde{\mathbf{M}}_{1-c}$ (no longer a payload of P_{1-c}), so base OIS (Lines 6–11) fails. We resolve this with a new OPPRF-based OIS (Section 5.3), which supports level-wise *batching* for efficiency.

With these changes and engineering optimizations (Section 5.4), AnonGBDT$^{\mathsf{OTSA}}$ significantly reduces runtime and communication versus the base one (Section 6).

5.2. Faster LWE Ciphertext Packing

The secure matrix-vector multiplication $\mathcal{F}_{\mathsf{BinMatVec}}$ uses RLWE encryption for computational efficiency and outputs $\underline{n}$ LWE ciphertexts, which incur large communication because $\mathcal{F}_{\mathsf{H2A}}$ must convert these LWE ciphertextsto shares. Communication can be cut by packing several LWE ciphertexts with a smaller dimension (N) into a single RLWE ciphertext with a larger dimension ($\underline{N}$). Squirrel does so in two stages: LWEDimLift (lift dimension $\underline{N} \to N$) followed by PackLWEs (recursive merging).

We replace this flow with FastPackLWEs followed by PackRLWEs (Algorithms 1 and 2). Figure 7 shows the adapted version. We also modify $\mathcal{F}_{\mathsf{BinMatVec}}$ from Squirrel with changes boxed (Line 6).

Let $\{\mathsf{LWE}_{\mathbf{s}}(m_i)\}_{0 \le i < \underline{n}}$, $\underline{n} = 2^\tau$, be encrypted under secret $\mathbf{s}$ (the vector representation of $\mathcal{F}_{\mathsf{A2H}}$'s $\underline{\mathsf{sk}}$). For each adjacent pair $\mathsf{LWE}_{\mathbf{s}}(m_{2j}) = (\mathbf{a}_1, b_1)$, $\mathsf{LWE}_{\mathbf{s}}(m_{2j+1}) = (\mathbf{a}_2, b_2)$, we can manipulate them as a new RLWE ciphertext pair:

$$\widehat{a} = \widehat{a}_1 + \widehat{a}_2 \cdot X^{N/2}, \;\; \widehat{b} = b_1 + b_2 \cdot X^{N/2}, \tag{7}$$

under a new secret $\widehat{s} = \underline{\mathsf{sk}}[0] - \sum_{k=1}^{N-1} \underline{\mathsf{sk}}[k] \cdot X^{N-k}$, where $\widehat{a}_i = \sum_{k=0}^{N-1} \mathbf{a}_i[k] \cdot X^k$ $(i \in \{1,2\})$.

The 0- and $(N/2)$-th coefficients, when decrypting $(\widehat{a}, \widehat{b})$ by $\widehat{s}$, are exactly m_{2j} and m_{2j+1}, *i.e.*, the two messages are packed within one RLWE ciphertext without lifting their dimensions individually. As $\widehat{s}$ is insecure (half its coefficients are zero), we immediately switch it to a valid RLWE secret key sk with one KeySwitch call (in Line 3). By merging all adjacent ones, we thus obtain $\underline{n}/2$ RLWE ciphertexts.

To integrate this idea, we remove the first recursion of PackLWEs, leading to PackRLWEs (Algorithm 2). Also, at Line 5, we change t from $N/2^\tau$ to $N/2^{\tau+1}$.

Algorithm 1 FastPackLWEs (Fast LWE Packing)

Input: LWE ciphertexts $\{\mathsf{LWE}_{\mathbf{s}}(m_i)\}_{0 \le i < \underline{n}}$ (for $\underline{n} = 2^\tau < N$) and lifting key $\mathsf{LK}_{\underline{\mathsf{sk}} \to \mathsf{sk}}$.
Output: An RLWE ciphertext $\widehat{ct}$.
1: **for** $i \in [0, \underline{n})$ with step-size as 2 **do**
2: Build an RLWE ciphertext $\widehat{ct^*}_j = (\widehat{a}_j, \widehat{b}_j)$, where $j = i/2$ and $\widehat{a}_j, \widehat{b}_j$ are given as Eq. (7).
3: $\widehat{ct}_j = \mathsf{KeySwitch}_{\underline{\mathsf{sk}} \to \mathsf{sk}}(\widehat{ct^*}_j)$ based on $\mathsf{LK}_{\underline{\mathsf{sk}} \to \mathsf{sk}}$.
4: **return** $\widehat{ct} \leftarrow \mathsf{PackRLWEs}(\{\widehat{ct}_j\})$.

Algorithm 2 PackRLWEs (RLWE Ciphertext Packing)

Input: RLWE ciphertexts $\{\widehat{ct}_j\}$ for some $j \in [2^\tau]$.
Output: An RLWE ciphertext $\widehat{ct}$.
1: **if** $\tau = 0$ **then**
2: **return** $\widehat{ct} \leftarrow \widehat{ct}_0$.
3: $\widehat{ct}_{even} \leftarrow \mathsf{PackRLWEs}(\{\widehat{ct}_{2j} | j \in [0, 2^{\tau-1} - 1]\})$.
4: $\widehat{ct}_{odd} \leftarrow \mathsf{PackRLWEs}(\{\widehat{ct}_{2j+1} | j \in [0, 2^{\tau-1} - 1]\})$.
5: **return** $\widehat{ct} \leftarrow (\widehat{ct}_{even} + X^t \cdot \widehat{ct}_{odd}) + \mathsf{EvalAuto}(\widehat{ct}_{even} - X^t \cdot \widehat{ct}_{odd}, N + t)$, where $t = N/2^{\tau+1}$.

Complexity: FastPackLWEs processes $\underline{n}/2$ pairs, cutting the cost of LWEDimLift [5] in half. PackRLWEs now requires only $\tau - 1$ recursions, again halving the workload compared with PackLWEs [5]. Consequently, the overall ciphertext-packing time of $\mathcal{F}_{\mathsf{BinMatVec}}$ is reduced by **50%**.

5.3. OPPRF-based OIS

The dual framework is *symmetric* across parties, but each instance remains asymmetric. Assume P_c owns the best split (z_*, u_*). Figure 8 shows our new OPPRF-based OIS (OOIS). If P_c is the *receiver*, it sets $\mathbf{b}_*^{(c)} = \widetilde{\mathbf{M}}_c[z_* B + u_*]$ secret-shares it with P_{1-c}, from which the children's $\langle \mathbf{b}^{(L,c)} \rangle_l^{\mathsf{B}}, \langle \mathbf{b}^{(R,c)} \rangle_l^{\mathsf{B}}$ are derived (Lines 1–4) as in base OIS.

If P_c is the *sender*, synchronizing the children's indicator shares given $\mathbf{b}_*^{(c)}$ is tricky. Recall that circuit-PSI facilitates the inclusion of payloads through two batch OPPRFs. The first identifies intersecting items via PSM testing. The second distinguishes sender-side payloads of items within each bucket and selects the "correct" one for computation: If $\mathbf{T}_{\mathsf{ch}}[i]$ is j-th item in $\mathbf{T}_{\mathsf{sh}}[i]$, then the programmed OPPRF output at the receiver is a share of $\mathbf{T}_{\mathsf{sh}}[i,j]$'s payload.

The remaining OOIS steps (Lines 5–10) build atop this concept: Firstly, P_c reorders $\mathbf{b}_*^{(c)}$ from its cuckoo-hash order to the raw order in $\mathbf{ID}_c$, producing a temporary indicator $\mathbf{b}'_*$ for the other instance. It then samples a random vector $\mathbf{r}^{1-c}$ and forms XOR share of $\mathbf{b}'_*$. Each sample $\mathbf{ID}_c[i']$ resides in some bucket $\mathbf{T}_{\mathsf{sh}}^{1-c}[i,j]$ under the other instance's e simple hashes. This lets P_c build a key-value table $\mathbf{T}$ with those items as keys and the corresponding shares of $\mathbf{b}_*^{(1-c)}$ as values (derived from $\mathbf{b}'_*$ and $\mathbf{r}^{1-c}$). A single call to $\mathcal{F}_{\mathsf{OPPRF}}$ supplies P_{1-c} with the complementary shares of $\mathbf{b}_*^{(1-c)}$. Finally, both parties obtain the left- and right-child indicators via $\mathcal{F}_{\mathsf{and}}$ and string XOR as in Line 4.

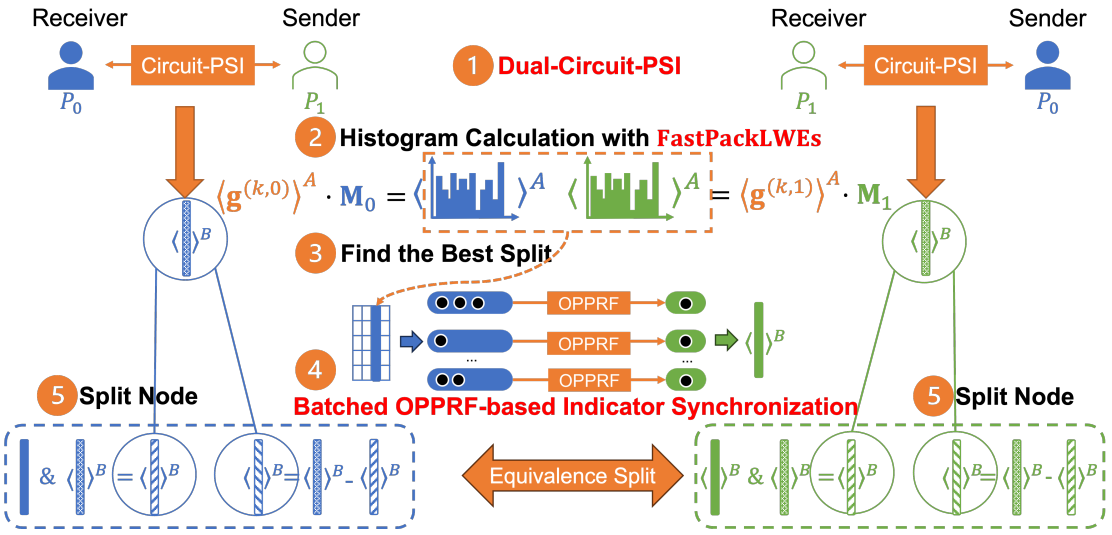


Figure 5. Overview of AnonGBDT$^{\text{OTSA}}$ (with our key contributions highlighted in red)

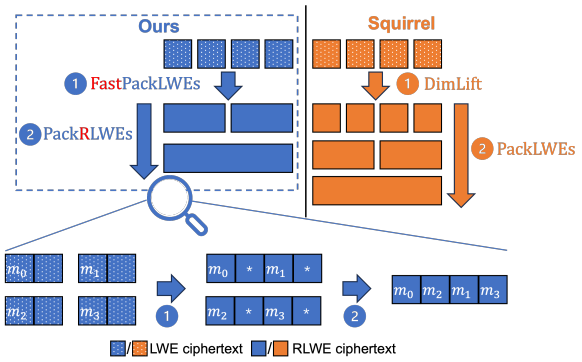


Figure 6. Example of FastPackLWEs ($\underline{n} = 4$, $N = 4$, and $\underline{N} = 2$): The adjacent LWE ciphertexts are first packed into an RLWE ciphertext via Algorithm 1. The RLWE ciphertexts are then recursively packed into a single RLWE ciphertext via Algorithm 2.

Abstraction: Conceptually, our OPPRF call expects the same ideal functionality as oblivious transfer for a sparse array [35] (with the same index domain and range). Sender-chosen values are returned on a sparse set of indices and pseudorandom fillers appear elsewhere. We keep the OPPRF interface because our pipeline already uses batch OPPRF at two call sites. This preserves batching in $\mathcal{F}_{\text{BOIS}}$.

Level batching: All internal nodes share $\mathbf{ID}_c$ (as the hash table's key set), so we pack all node indicators per level into one OPPRF value. The *batched* variant (BOIS, Figure 12 in Appendix B) reduces the $O(2^D)$ cost to $O(D)$ for depth D.

Complexity: For a batch of nodes, it first calls one $\mathcal{F}_{\text{OPPRF}}$, costs $O(n\ell)$ bits with $\mathcal{O}(1)$ rounds. Then, it calls $2n$ $\mathcal{F}_{\text{and}}$ (each sends $O(\ell)$ bits) with $\mathcal{O}(1)$ rounds.

Theorem 5.1. *Π_{BOIS} realizing $\mathcal{F}_{\text{BOIS}}$ in Figure 12 is a secure protocol under $\mathcal{F}_{\text{OPPRF}}$ and $\mathcal{F}_{\text{and}}$ hybrids.*

We defer the proof to Appendix D due to space limits.

Functionality $\mathcal{F}_{\text{BinMatVec}}$

P_c**'s Input:** gradients share $\langle \mathbf{g} \rangle_c^{\mathsf{B}}$, binary feature representation $\mathbf{M}_c \in \{0,1\}^B$
P_{1-c}**'s Input:** gradients share $\langle \mathbf{g} \rangle_{1-c}^{\mathsf{A}}$, RLWE $\underline{\mathsf{sk}}_{1-c}$ for $\mathcal{F}_{\text{A2H}}$, and sk_{1-c} for $\mathcal{F}_{\text{H2A}}$
Public parameters: KeySwitch lifting key LK
Output: $\langle \mathbf{M}_c \cdot \mathbf{g} \rangle_l^{\mathsf{A}}$ to P_l.

Protocol $\Pi_{\text{BinMatVec}}$

1: Jointly run $\mathcal{F}_{\text{A2H}}$, P_{1-c} inputs $\langle \mathbf{g} \rangle_{1-c}^{\mathsf{A}}$, $\underline{sk}_{1-c}$, and P_c inputs $\langle \mathbf{g} \rangle_c^{\mathsf{B}}$; P_c gets $[\![\mathbf{g}]\!]$, and P_{1-c} gets nothing.
2: P_c initializes a size-$(B \cdot m)$ array of LWE encryption of 0, denoted by $\mathbf{ct}[j]$ for $j \in [B \cdot m]$.
3: **for** all position (j, i) such that $\mathbf{M}[j, i] = 1$ **do**
4: $\quad P_c$ extracts LWE $\bar{\mathbf{ct}}[j, i]$ from $[\![\mathbf{g}]\!]$ that decrypts to $\mathbf{g}[i]$ under P_{1-c}'s secret key.
5: $\quad P_c$: $\mathbf{ct}[j] = \mathbf{ct}[j] + \bar{\mathbf{ct}}[j, i]$ ▷ pick-then-sum
6: P_c: $[\![\mathbf{M_c} \cdot \mathbf{g}]\!] \leftarrow \mathsf{FastPackLWEs}(\{\mathbf{ct}[j]\}_{j \in [m \cdot B)}, \mathsf{LK})$
7: Jointly invoke $\mathcal{F}_{\text{H2A}}$, P_c inputs $[\![\mathbf{M_c} \cdot \mathbf{g}]\!]$ while P_{1-c} inputs sk_{1-c}. Both parties return the resulting arithmetic share $\langle \mathbf{M}_c \cdot \mathbf{g} \rangle^{\mathsf{A}}$.

Figure 7. Functionality and protocol of BinMatVec

5.4. Further Optimizations

• **High-precision score computation**: With fixed-point precision $f = 20$, intermediate shares (*e.g.*, $\mathcal{F}_{\text{mul}}$ in Eq. (2)) can exceed $\mathbb{Z}_{2^\ell}$ when n is large. We prescale inputs to $[1, 2]$ and rescale the result, preventing overflow at negligible cost.

• **Gradient packing in $\mathcal{F}_{\text{mux}}$**: Filtering the (secret-shared) gradients $\mathbf{g}$ and $\mathbf{h}$ at k-th node requires the same sample indicator $\mathbf{b}^{(k)}$ (Eq. (5)). Real values (*e.g.*, every entry of $\mathbf{g}$) are encoded as ℓ-bit integers by fixed-point representation. As the security parameter is $\lambda = 2\ell = 128$, we can halve

Functionality $\mathcal{F}_{\mathsf{OOIS}}$

P_c**'s Input:** $\widetilde{\mathbf{M}}_c$, $\langle \mathbf{b}^{(0)} \rangle_c^{\mathsf{B}}$, $\langle \mathbf{b}^{(1)} \rangle_c^{\mathsf{B}}$, (z_*, u_*), and simple hash table $\mathbf{T}_{\mathsf{sh}}^{1-c}$.

P_{1-c}**'s Input:** $\langle \mathbf{b}^{(0)} \rangle_{1-c}^{\mathsf{B}}$, $\langle \mathbf{b}^{(1)} \rangle_{1-c}^{\mathsf{B}}$, and cuckoo hash table $\mathbf{T}_{\mathsf{ch}}^{1-c}$.

Output: $\langle \mathbf{b}^{(L,0)} \rangle^{\mathsf{B}}, \langle \mathbf{b}^{(R,0)} \rangle^{\mathsf{B}}, \langle \mathbf{b}^{(L,1)} \rangle^{\mathsf{B}}, \langle \mathbf{b}^{(R,1)} \rangle^{\mathsf{B}}$ for the 'L'eft and 'R'ight children of the two instances

Protocol Π_{OOIS}

1: P_c gets a random mask $\langle \mathbf{b}_* \rangle_c^{\mathsf{B}} := \mathbf{r}^c \leftarrow_{\$} \{0,1\}^{(1+\epsilon)n}$.
2: P_c sets $\mathbf{b}_*^{(c)} = \widetilde{\mathbf{M}}_c[z_* B + u_*]$.
3: P_c shares $\langle \mathbf{b}_*^{(c)} \rangle_{1-c}^{\mathsf{B}} \leftarrow \mathbf{b}_*^{(c)} \oplus \mathbf{r}^c$ with P_{1-c}.
4: P_l derives $\langle \mathbf{b}^{(L,c)} \rangle_l^{\mathsf{B}} = \mathcal{F}_{\mathsf{and}}(\langle \mathbf{b}_*^{(c)} \rangle_l^{\mathsf{B}}, \langle \mathbf{b}^{(c)} \rangle_l^{\mathsf{B}})$ and $\langle \mathbf{b}^{(R,c)} \rangle_l^{\mathsf{B}} = \langle \mathbf{b}^{(L,c)} \rangle_l^{\mathsf{B}} \oplus \langle \mathbf{b}^{(c)} \rangle_l^{\mathsf{B}}$.
5: P_c "reorders" $\mathbf{b}_*^{(c)}$ to $\mathbf{b}'_*$ to align with $\mathbf{ID}_c$.
6: P_c samples $\langle \mathbf{b}_*^{(1-c)} \rangle_c^{\mathsf{B}} = \mathbf{r}^{1-c} \leftarrow_{\$} \{0,1\}^{(1+\epsilon)n}$.
7: $\forall j \in [0, e), \forall i \in [0, (1+\epsilon)n)$: P_c creates a table $\mathbf{T} = \{(\mathbf{T}_{\mathsf{sh}}^{1-c}[i,j], \langle \mathbf{b}_*^{1-c}[i] \rangle_{1-c}^{B}\}$, where $\langle \mathbf{b}_*^{(1-c)}[i] \rangle_{1-c}^{B} = \mathbf{b}'_*[i'] \oplus \mathbf{r}^{1-c}[i]$ and $\mathbf{ID}_c[i']$ is placed in $\mathbf{T}_{\mathsf{sh}}^{1-c}[i,j]$ via simple hashing.
8: $\langle \mathbf{b}_*^{(1-c)} \rangle_{1-c}^{\mathsf{B}} \leftarrow \mathcal{F}_{\mathsf{OPPRF}}(\{\mathbf{T}\}, \{\mathbf{T}_{\mathsf{ch}}^{1-c}\})$.
9: P_l runs $\langle \mathbf{b}^{(L,1-c)} \rangle_l^{\mathsf{B}} = \mathcal{F}_{\mathsf{and}}(\langle \mathbf{b}_*^{(1-c)} \rangle_l^{\mathsf{B}}, \langle \mathbf{b}^{(1-c)} \rangle_l^{\mathsf{B}})$.
10: P_l runs $\langle \mathbf{b}^{(R,1-c)} \rangle_l^{\mathsf{B}} = \langle \mathbf{b}^{(L,1-c)} \rangle_l^{\mathsf{B}} \oplus \langle \mathbf{b}^{(1-c)} \rangle_l^{\mathsf{B}}$.

Figure 8. Ideal functionality and protocol of OOIS

the number of OT used in $\mathcal{F}_{\mathsf{mux}}$ by entry-wise packing of $\mathbf{g}$ and $\mathbf{h}$, compared to transferring them individually.

• **Gradient packing in $\mathcal{F}_{\mathsf{A2H}}$**: In $\mathcal{F}_{\mathsf{A2H}}$, we can encode gradients $\langle \mathbf{g} \rangle^{\mathsf{A}}$ and $\langle \mathbf{h} \rangle^{\mathsf{A}}$ within one RLWE ciphertext, *i.e.*, $\widehat{a}_l = \sum_{i=0}^{\underline{N}/2-1} \langle \mathbf{g}[i] \rangle_l^{\mathsf{A}} X^i + \sum_{i=\underline{N}/2}^{\underline{N}-1} \langle \mathbf{h}[i] \rangle_l^{\mathsf{A}} X^i$ (see Section 2.2). After pick-then-sum, two LWE ciphertexts (of dimension $\underline{N}$) can be directly packed in one RLWE ciphertext (of dimension $\underline{N}$) without KeySwitch, thereby reducing one recursion of FastPackLWEs.

• **Skipping $\mathcal{F}_{\mathsf{A2H}}$ for the split owner**: If party P_c holds the best split at node k, it locally allocates the RLWE-encrypted $\mathbf{g}^{(k,c)}$ and $\mathbf{h}^{(k,c)}$ to bins at node $2k$ and aggregates them, eliminating the costly share-to-ciphertext conversion.

• **Dummy-free aggregation**: In $\mathcal{F}_{\mathsf{BinMatVec}}$, ciphertexts follow the cuckoo-hash order, including the εn dummies. By realigning $\mathbf{M}_l$ and the ciphertext list with the original row order of $\mathbf{X}_l$, each party sums only genuine samples, avoiding dummy-related overhead.

• **Lightweight oblivious** argmax: Pairwise comparisons use OTs, outperforming GC variants [19]. Because the split-index space is public, the final selection is obtained directly with $\mathcal{F}_{\mathsf{OT}}$, removing the extra $\mathcal{F}_{\mathsf{mux}}$ step of Squirrel [5].

• **Erasing superfluous coefficients in** FastPackLWEs: To prevent leakage, the prior work [20] zeroizes useless coefficients between $X^{i*N/2^\tau}$ and $X^{(i+1)*N/2^\tau}$ for $i \in [2^\tau - 1]$ by the trace function, which recursively evaluates EvalAuto on the ciphertexts after PackLWEs. Instead, we just add randomness to nullify these coefficients (without EvalAuto but sharing a similar purpose) to improve efficiency.

TABLE 2. COMMUNICATION COMPLEXITY AND ROUNDS OF EACH COMPONENT OF AnonGBDT (BASE VERSUS FINAL)

Component	Complexity	Round
Dual-circuit-PSI	$\mathcal{O}(n\ell)$	$\mathcal{O}(\log \ell) + 1$
GHA (Base)	$\mathcal{O}(2Bmn\ell + n \log Q + 2Bm \log Q)$	$\mathcal{O}(1)$
GHA (Final)	$\mathcal{O}(2n \log Q + 4Bm \log Q)$	2
$\mathcal{F}_{\mathsf{OIS}}$ (Base)	$\mathcal{O}(Bmn\ell)$ per node	$\mathcal{O}(1)$
$\mathcal{F}_{\mathsf{BOIS}}$ (Final)	$\mathcal{O}(3n\ell)$ per level	$\mathcal{O}(1)$

5.5. Bringing Everything Together

Figure 9 presents AnonGBDT$^{\mathsf{OTSA}}$. Two circuit-PSI instances run in parallel with swapped roles. For instance $i \in \{0,1\}$, inputs are $\mathbf{ID}_0$ and $\mathbf{ID}_1$, and the only payload is $\mathbf{y}$, producing the root indicator shares $\langle \mathbf{b}^{(1,i)} \rangle_l^{\mathsf{B}}$ and label shares $\langle \mathbf{y}^i \rangle_l^{\mathsf{A}}$. Unlike our base scheme, $\mathbf{M}_i, \widetilde{\mathbf{M}}_i$ stay local.

For each tree, both parties derive and filter $\langle \mathbf{g}^{(1,i)} \rangle_l^{\mathsf{A}}$ and $\langle \mathbf{h}^{(1,i)} \rangle_l^{\mathsf{A}}$ per instance (Lines 6–7). Nodes are processed level-wise using BOIS (Figure 12). At depth d, two public sets $\mathbf{k}_0$, $\mathbf{k}_1$ record the owner of each node's best split.

Histogram construction: For roots and left children, both parties aggregate gradients with $\mathcal{F}_{\mathsf{BinMatVec}}$ over $\mathbf{M}_0$ and $\mathbf{M}_1$ (accelerated by FastPackLWEs) and then concatenate locally. For right children, histograms are obtained by subtracting the parent and left-child values. Aggregation is order-independent, so it runs only once (not per instance).

Split selection: Split gains for all candidates are evaluated with $\mathcal{F}_{\mathsf{mul}}$ and $\mathcal{F}_{\mathsf{div}}$ (Line 16. $\mathcal{F}_{\mathsf{argmax}}$ reveals $(z_*^{(k)}, u_*^{(k)})$ to its owner P_c, which adds k to $\mathbf{k}_c$. Given $\mathbf{k}_0, \mathbf{k}_1$, $\mathcal{F}_{\mathsf{BOIS}}$ updates level-wise indicators for the next level's filtering.

Leaf weights: For each leaf k, the weight is derived from whichever gradient share $\langle \mathbf{g}^{(k,i)} \rangle_l^{\mathsf{A}}$ has fewer samples (for order independence), and the predictions are accumulated.

Confining heavy computation to local data removes the OT-intensive histogram phase of the base design. The trade-off is maintaining one "logical" tree per instance, increasing total communication to at most $2(1+\varepsilon)\times$ that of Squirrel.

5.5.1. Complexity. Table 2 reports the per-component complexity for AnonGBDT. AnonGBDT$^{\mathsf{OTSA}}$ achieves its performance gains primarily due to: i) In base gradient histogram aggregation (GHA), the dominant cost is $\mathcal{F}_{\mathsf{mux}}$ with complexity $\mathcal{O}(Bmnl)$, which is eliminated in AnonGBDT$^{\mathsf{OTSA}}$, and ii) Unlike the base $\mathcal{F}_{\mathsf{OIS}}$, which is invoked per node and scales linearly with Bm, its batch extension $\mathcal{F}_{\mathsf{BOIS}}$ is invoked per level.

Theorem 5.2. *$\Pi_{\mathsf{GBDT}}^{\mathsf{OTSA}}$ (Figure 9) is anonymous GBDT training under $\mathcal{F}_{\mathsf{argmax}}$, $\mathcal{F}_{\mathsf{BOIS}}$, $\mathcal{F}_{\mathsf{BinMatVec}}$ (with $\mathcal{F}_{\mathsf{A2H}}$, $\mathcal{F}_{\mathsf{H2A}}$), $\mathcal{F}_{\mathsf{CPSI}}$, $\mathcal{F}_{\mathsf{div}}$, $\mathcal{F}_{\mathsf{greater}}$, $\mathcal{F}_{\mathsf{mul}}$, $\mathcal{F}_{\mathsf{mux}}$, $\mathcal{F}_{\mathsf{sigmoid}}$-hybrid and semantically-secure RLWE encryption.*

We defer the proof to Appendix D due to space limits.

AnonGBDT $\Pi^{\mathsf{OTSA}}_{\mathsf{GBDT}}(\mathbf{Input}^{\Pi_0}_0, \mathbf{Input}^{\Pi_0}_1, \mathbf{pp})$

P_0**'s Input:** $\mathbf{ID}_0$, feature $\mathbf{X}_0$ and label $\mathbf{y}$.
P_1**'s Input:** $\mathbf{ID}_1$, feature $\mathbf{X}_1$.
Public parameters: $\mathbf{pp} = \{D, B, T, (\underline{sk}_l, \underline{pk}_l)$ for $\mathcal{F}_{\mathsf{A2H}}, (sk_l, pk_l)$ for $\mathcal{F}_{\mathsf{H2A}}$, KeySwitch Lifting Key $\mathsf{LK}_{\underline{sk}_l \to \mathsf{sk}_l}\}$.
Output: Output^{Π}_0 for P_0 and Output^{Π}_1 for P_1

P_l discretizes its raw data $\mathbf{X}_l$ to $\mathbf{M}_l \in \{0,1\}^{m \cdot B \times n}$, which is aggregated to $\widetilde{\mathbf{M}}_l$.
For instance $i = 0$: $\langle \mathbf{b}^{(1,0)} \rangle^{\mathsf{B}}_l, \langle \mathbf{y}^0 \rangle^{\mathsf{A}}_l \leftarrow \mathcal{F}^0_{\mathsf{CPSI}}(\{\mathbf{ID}_0\, \mathbf{y}\}; \{\mathbf{ID}_1\})$, with P_0's $\mathbf{T}^0_{\mathsf{ch}}$ and P_1's $\mathbf{T}^0_{\mathsf{sh}}$.
For instance $i = 1$: $\langle \mathbf{b}^{(1,1)} \rangle^{\mathsf{B}}_l, \langle \mathbf{y}^1 \rangle^{\mathsf{A}}_l \leftarrow \mathcal{F}^1_{\mathsf{CPSI}}(\{\mathbf{ID}_1\}; \{\mathbf{ID}_0, \mathbf{y}\})$, with P_1's $\mathbf{T}^1_{\mathsf{ch}}$ and P_0's $\mathbf{T}^1_{\mathsf{sh}}$.
for tree $t \in \mathcal{T}_t$ **do**
  For $i \in \{0,1\}$: P_l derives gradients $\langle \mathbf{g}^{(1,i)} \rangle^{\mathsf{A}}_l, \langle \mathbf{h}^{(1,i)} \rangle^{\mathsf{A}}_l$ at root using $\mathcal{F}_{\mathsf{sigmoid}}(\langle \tilde{\mathbf{y}}^i \rangle^{\mathsf{A}}_l)$ and $\mathcal{F}_{\mathsf{mul}}$; see Eq. (1).
  For $i \in \{0,1\}$: P_l filters $\langle \mathbf{g}^{(1,i)} \rangle^{\mathsf{A}}_l, \langle \mathbf{h}^{(1,i)} \rangle^{\mathsf{A}}_l$ at root using $\mathcal{F}_{\mathsf{mux}}$ given $\langle \mathbf{b}^{(1,i)} \rangle^{\mathsf{B}}_l$; see Eq. (5).
  for level $d \in \{1, 2, \ldots, D-1\}$ of the tree **do**
    Both parties maintain two public sets $\mathbf{k}_0 \leftarrow \emptyset, \mathbf{k}_1 \leftarrow \emptyset$. ▷ Record node k's best split is at P_0 or P_1
    for the internal nodes $k \in \{2^{d-1}, \ldots, 2^d - 1\}$ at level d **do** ▷ Batch updates on level d for BOIS
      if k is 1 or even **then**
        P_l aggregates gradients over $\mathbf{M}_0$ (as in Squirrel) with our new subroutine FastPackLWEs:
        $\langle \widetilde{\mathbf{g}}^{(k,0)} \rangle^{\mathsf{A}}_l || \langle \widetilde{\mathbf{h}}^{(k,0)} \rangle^{\mathsf{A}}_l = \mathcal{F}_{\mathsf{BinMatVec}}(\{\langle \mathbf{g}^{(k,0)} \rangle^{\mathsf{A}}_0 || \langle \mathbf{h}^{(k,0)} \rangle^{\mathsf{A}}_0, \mathbf{M}_0\}, \{\langle \mathbf{g}^{(k,0)} \rangle^{\mathsf{A}}_1 || \langle \mathbf{h}^{(k,0)} \rangle^{\mathsf{A}}_1, \mathsf{sk}_1\})$.
        Similarly, P_l runs $\mathcal{F}_{\mathsf{BinMatVec}}$ with FastPackLWEs over $\mathbf{M}_1$ to get $\langle \tilde{\mathbf{g}}^{(k,1)} \rangle^{\mathsf{A}}_l$ and $\langle \tilde{\mathbf{h}}^{(k,1)} \rangle^{\mathsf{A}}_l$.
        P_l locally concatenates the shares as $\langle \widetilde{\mathbf{g}}^{(k)} \rangle^{\mathsf{A}}_l = \langle \widetilde{\mathbf{g}}^{(k,0)} \rangle^{\mathsf{A}}_l \| \langle \widetilde{\mathbf{g}}^{(k,1)} \rangle^{\mathsf{A}}_l$, $\langle \widetilde{\mathbf{h}}^{(k)} \rangle^{\mathsf{A}}_l = \langle \widetilde{\mathbf{h}}^{(k,0)} \rangle^{\mathsf{A}}_l \| \langle \widetilde{\mathbf{h}}^{(k,1)} \rangle^{\mathsf{A}}_l$.
      else P_l calculates $\langle \tilde{\mathbf{g}}^{(k)} \rangle^{\mathsf{A}}_l = \langle \tilde{\mathbf{g}}^{(k/2)} \rangle^{\mathsf{A}}_l - \langle \tilde{\mathbf{g}}^{(k-1)} \rangle^{\mathsf{A}}_l$, $\langle \tilde{\mathbf{h}}^{(k)} \rangle^{\mathsf{A}}_l = \langle \tilde{\mathbf{h}}^{(k/2)} \rangle^{\mathsf{A}}_l - \langle \tilde{\mathbf{h}}^{(k-1)} \rangle^{\mathsf{A}}_l$.
      P_l computes the splitting scores $\langle \mathbf{L}^{(k)}_{\mathsf{sp}} \rangle^{\mathsf{A}}_l$ using $\mathcal{F}_{\mathsf{mul}}$ and $\mathcal{F}_{\mathsf{div}}$ for all (z, u) based on Eq. (2).
      P_l gets $\langle z^{(k)}_* \rangle^{\mathsf{A}}_l, \langle u^{(k)}_* \rangle^{\mathsf{A}}_l \leftarrow \mathcal{F}_{\mathsf{argmax}}(\langle \mathbf{L}_{\mathsf{sp}} \rangle^{\mathsf{A}}_l)$.
      Open $c \leftarrow \mathcal{F}_{\mathsf{greater}}(\langle z^{(k)}_* \rangle, m-1)$ and reveal $(z^{(k)}_*, u^{(k)}_*)$ to P_c, who writes it to Output^{Π}_c and k to $\mathbf{k}_c$.
    For $k \in \mathbf{k}_0$: P_l runs $\mathcal{F}_{\mathsf{BOIS}}(\{\{\langle \mathbf{b}^{(k,0)} \rangle^{\mathsf{B}}_0, \langle \mathbf{b}^{(k,1)} \rangle^{\mathsf{B}}_0, (z^{(k)}_*, u^{(k)}_*)\}, \widetilde{\mathbf{M}}_0, \mathbf{T}^1_{\mathsf{sh}}\}; \{\{\langle \mathbf{b}^{(k,0)} \rangle^{\mathsf{B}}_1, \langle \mathbf{b}^{(k,1)} \rangle^{\mathsf{B}}_1\}, \mathbf{T}^1_{\mathsf{ch}}\})$
    to get $\{\langle \mathbf{b}^{(2k,i)} \rangle^{\mathsf{B}}_l, \langle \mathbf{b}^{(2k+1,i)} \rangle^{\mathsf{B}}_l\}_{i \in \{0,1\}}$.
    For $k \in \mathbf{k}_1$: P_l runs $\mathcal{F}_{\mathsf{BOIS}}$ (now flip the role with $\widetilde{\mathbf{M}}_1, \mathbf{T}^0_{\mathsf{sh}}, \mathbf{T}^0_{\mathsf{ch}}$) to get $\{\langle \mathbf{b}^{(2k,i)} \rangle^{\mathsf{B}}_l, \langle \mathbf{b}^{(2k+1,i)} \rangle^{\mathsf{B}}_l\}_{i \in \{0,1\}}$.
    For $i \in \{0,1\}$: update left-child: $\langle \mathbf{g}^{(2k,i)} \rangle^{\mathsf{A}}_l (/ \langle \mathbf{h}^{(2k,i)} \rangle^{\mathsf{A}}_l) = \mathcal{F}_{\mathsf{mux}}(\langle \mathbf{b}^{(2k,i)} \rangle^{\mathsf{B}}_l, \langle \mathbf{g}^{(k,i)} \rangle^{\mathsf{A}}_l (/ \langle \mathbf{h}^{(k,i)} \rangle^{\mathsf{A}}_l))$.
    For $i \in \{0,1\}$: update right-child: $\langle \mathbf{g}^{(2k+1,i)} \rangle^{\mathsf{A}}_l = \langle \mathbf{g}^{(k,i)} \rangle^{\mathsf{A}}_l - \langle \mathbf{g}^{(2k,i)} \rangle^{\mathsf{A}}_l$, $\langle \mathbf{h}^{(2k+1,i)} \rangle^{\mathsf{A}}_l = \langle \mathbf{h}^{(k,i)} \rangle^{\mathsf{A}}_l - \langle \mathbf{h}^{(2k,i)} \rangle^{\mathsf{A}}_l$.
  P_l computes leaf weights $\langle \mathbf{w} \rangle^{\mathsf{A}}_l$ using $\mathcal{F}_{\mathsf{div}}$ based on Eq. (3) and writes them in Output^{Π}_l. ▷ Order-independent
  For $i \in \{0,1\}$: P_l updates $\langle \tilde{\mathbf{y}}^i[j] \rangle^{\mathsf{A}}_l$ by accumulating $\mathcal{F}_{\mathsf{mux}}(\langle \mathbf{b}^{(k,i)}[j] \rangle^{\mathsf{B}}_l, \langle \mathbf{w}[k] \rangle^{\mathsf{A}}_l)$ $(\forall j)$ over all leaves k.

Figure 9. AnonGBDT$^{\mathsf{OTSA}}$ (with key differences to our base solution highlighted in blue)

5.5.2. Anonymous Batch Inference. In inference [21], each party P_l holds a vertically partitioned dataset (no labels), (plaintext) best splits $\mathbf{C}_l$, and weight shares $\langle \mathbf{w} \rangle^{\mathsf{A}}_l$. One party finally learns the prediction probabilities $\mathbf{p}$ on the joint data. Prior protocols often assume PSI-aligned inputs.

Evaluation using a secure multiplexer can proceed as follows: i) local traversal: each P_l computes a binary vector $\mathbf{b}^{(k,l)}$ (AND-based share $\langle \mathbf{b}^{(k)} \rangle^{\mathsf{B}}_l$ in Squirrel) by comparing features with local split criteria; $\mathbf{b}^{(k,l)}[i] = 1$ indicates that sample i *may* reach node k; ii) conjunction: the true indicator is $\mathbf{b}^{(k)} = \mathbf{b}^{(k,0)} \wedge \mathbf{b}^{(k,1)}$, and the tree output is

$$\sum\nolimits_k \mathcal{F}_{\mathsf{mux}}(\langle \mathbf{b}^{(k)} \rangle^{\mathsf{B}}_l, \langle \mathbf{w}[k] \rangle^{\mathsf{A}}_l).$$

To keep inference-time identifiers hidden, we can run circuit-PSI preceding the above procedure, which induces misaligned local indicators. An OPPRF-based synchronization step then aligns indicators across parties: P_1 transfers a share of $\mathbf{b}^{(k,1)}$ to P_0, yielding aligned shares $\langle \mathbf{b}^{(k,1)} \rangle^{\mathsf{B}}_l$ and $\langle \mathbf{b}^{(k,0)} \rangle^{\mathsf{B}}_l$ on which $\mathcal{F}_{\mathsf{and}}$ applies. The full algorithm is in Appendix C.

5.6. Generalizability

Our dual-circuit-PSI framework generalizes to two-party *vertical* FL (VFL) models that exchange intermediate computations over aligned samples. These models require a data alignment step that, if using standard PSI, leaks sensitive intersection membership. Our framework provides a blueprint for *anonymous* VFL by replacing this leaky pre-alignment.

1) Anonymous Handshake: It establishes a "symmetric private" alignment in which each party, and only that party, learns its own cuckoo-hash permutation of the intersecting samples, so subsequent computations are performed on the same set of entities without revealing their identities.

2) Synchronization and computation: P_l, with its own secret permutation, re-aligns its local information and continues the computation with the secret shared payload (*e.g.*, completing a forward pass). For data flowing in the reverse direction (*e.g.*, encrypted/noised gradients), our $\mathcal{F}_{\mathsf{BOIS}}$ can be adapted to securely transfer the aligned information.

This workflow can be instantiated in various VFL architectures, *e.g.*, federated transformers [36] and split-NN [37].

TABLE 3. MICROBENCHMARKS OF AnonGBDT, SINGLE-THREAD

Protocol	instance	Time (s) WAN	Time (s) LAN	Comm. (MB)
$\mathcal{F}_{\mathsf{argmax}}$ [7]	10^5	4.9	3.0	13.4
$\mathcal{F}_{\mathsf{greater}}$	10^6	22.2	21.2	81.9
$\mathcal{F}_{\mathsf{mux}}$	10^6	12.4	1.7	17.5
$\mathcal{F}_{\mathsf{BOIS}}$	10^6	12.3	8.4	68.6
$\mathcal{F}_{\mathsf{BinMatVec}}$ [8]	1	3.7	1.8	29.6
$\mathcal{F}_{\mathsf{sigmoid}}$ (ours)	10^5	**13.4**	**4.8**	**38.4**
$\mathcal{F}_{\mathsf{sigmoid}}$ [5]	10^6	31.5	15.5	235.8

Local	# of inputs	Time (s)
FastPackLWEs (ours)	128	**0.22**
FastPackLWEs (ours)	512	**0.87**
LWEDimLift + PackLWEs [5]	128	1.02
LWEDimLift + PackLWEs [5]	512	3.80

6. Evaluation

Parameters: The security parameters are $\lambda = 128, \kappa = 40$. Our designs are implemented in C++ based on Cheetah [38]. We use the stash-less circuit-PSI [15] with 3 hash functions and $\epsilon = 1.3$, and $\ell = 64$-bit arithmetic sharing with precision $f = 20$. For GBDT, $\alpha = 0.001$ and $\gamma = 0$. RLWE is realized using Cheon–Kim–Kim–Song [39] of SEAL 4.1 [40] with $p = 2^{\ell}, N = 8192, \underline{N} = 4096, Q \approx 2^{109}$ [5].

Testbed: All experiments run on two identical hosts, each with an 8-core 2.5GHz Intel Xeon Platinum 8255C CPU and 64GB RAM under: i) **LAN**: 1Gbps bandwidth with a latency (ping time) ∼2ms, and ii) **WAN**: we emulate a 200Mbps link with 20ms latency using the traffic control utility (`tc`).

Metrics: We measure the end-to-end (offline and online) runtime and the total communication costs.

Microbenchmarks: Table 3 shows the performance breakdown. Interactive primitives, requiring message exchange between parties, include the protocols for argmax (taking 10 inputs), greater, mux, BOIS, BinMatVec (taking two 10^5 matrices), and our optimized sigmoid (Section 4.2.3).

For local primitives, we compare our FastPackLWEs with Squirrel's subroutine. Combined with the optimizations (Section 5.4), we achieve a $\sim 5\times$ runtime speedup.

6.1. Overhead of AnonGBDT$^{\mathsf{OTSA}}$

We ran benchmarks on synthetic datasets with varying n, m, and D, using a single thread. Table 4 reports efficiency results. For runtime, AnonGBDT$^{\mathsf{OTSA}}$ is consistently faster by $\sim 3\times$ to $20\times$, and the gap widens as n, m, or D grows. For communication, our base AnonGBDT scales linearly in n and m, eventually out-of-memory (OOM), due to the OT-heavy $\mathcal{F}_{\mathsf{mux}}$-based gradient aggregation non-batched $\mathcal{F}_{\mathsf{OIS}}$. AnonGBDT$^{\mathsf{OTSA}}$ removes these bottlenecks, achieving up to $44\times$ lower traffic and avoiding OOM even at the largest dataset sizes tested.

7. Each run with 10 inputs.

8. Input vector size of 128 vector size and matrix size of 128×10^5.

TABLE 4. EFFICIENCY OF AnonGBDT ($n = n_0 = n_1 = 10^5$, $m = 50$, $B = 16$, $D = 5$, SINGLE THREAD)

Parameters	WAN Time (s)		LAN Time (s)		Comm. (GB)	
	Base	Final	Base	Final	Base	Final
$n = 100$K	3873.5	195.2	567.0	112.5	26.1	0.6
$n = 250$K	OOM	346.5	OOM	227.0	OOM	1.1
$n = 500$K	OOM	602.6	OOM	422.2	OOM	2.1
$n = 1$M	OOM	1120.9	OOM	817.1	OOM	4.0
$D = 3$	1077.0	72.3	142.8	37.3	6.6	0.2
$D = 4$	1999.2	117.1	271.2	64.5	13.1	0.4
$D = 6$	7462.7	357.6	1024.6	203.6	51.8	0.9
$m = 10$	969.2	91.2	144.0	36.9	5.5	0.4
$m = 25$	2061.5	131.1	303.1	67.1	13.2	0.5
$m = 100$	OOM	334.0	OOM	214.1	OOM	0.7

TABLE 5. EFFICIENCY OF AnonGBDT FOR $n_0 \neq n_1$ ($n_1 = 10^5$, $m = 50$, $B = 16$, $D = 5$, SINGLE THREAD)

Size n_0	WAN Time (s)		LAN Time (s)		Comm. (GB)	
	Base	Final	Base	Final	Base	Final
$0.10n_1$	439.3	98.5	72.4	49.1	3.4	0.34
$0.20n_1$	780.4	105.6	121.3	54.5	5.6	0.37
$0.50n_1$	2156.3	138.8	267.4	69.7	15.2	0.45
$0.75n_1$	3047.2	167.4	413.8	87.3	21.3	0.54
$1.00n_1$	3873.5	195.2	567.0	112.5	26.1	0.6

AnonGBDT is also evaluated under unbalanced settings ($n_0 \neq n_1$), as shown in Table 5. The base design benefits more than AnonGBDT$^{\mathsf{OTSA}}$, with LAN speedups of $1.5-5\times$ as the imbalance ratio ($\frac{n_0}{n_1}$) decreases. This is because AnonGBDT$^{\mathsf{OTSA}}$ must run $\mathcal{F}_{\mathsf{BOIS}}$, $\mathcal{F}_{\mathsf{sigmoid}}$ twice, processing to align with each party's inputs separately.

6.2. Comparisons with Prior Arts

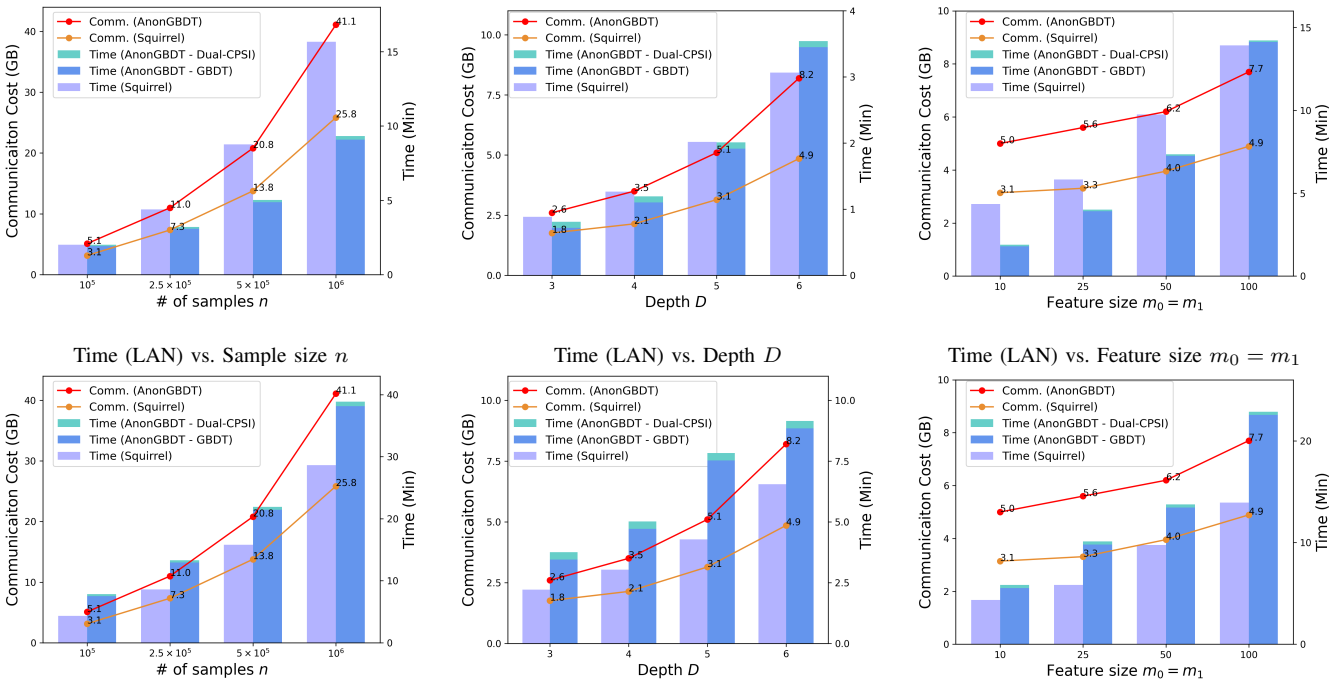
Efficiency: AnonGBDT$^{\mathsf{OTSA}}$ is evaluated on the same hardware as Squirrel [5], across two hosts with 8 CPU cores each. Pivot [8] and HEP-XGB [7] were evaluated under more favorable conditions (*e.g.*, 32 cores for HEP-XGB) and heterogeneous network settings; we quoted their reported results in Table 6. Despite the stronger privacy guarantees (no identifier leakage), our runtime is only $\sim 0.7\times$–$1.7\times$ that of Squirrel and remains competitive with the other less private schemes (which would require prohibitively costly generic tools for similar privacy).

Effectiveness: We train models via AnonGBDT$^{\mathsf{OTSA}}$ on six public datasets [41] and evaluate with our inference protocol Π_{Infer} in Figure 13. XGBoost [42] serves as a plaintext baseline. Using 5-fold cross-validation, $T = 10$ trees, and depth $D = 5$, we attain the highest F1 score across all tasks (Table 7), primarily due to our improved sigmoid approximation (Section 4.2.3).

6.3. Scalability

We assess scalability by running AnonGBDT$^{\mathsf{OTSA}}$ with eight threads (unlike the settings for comparison in Section 6.2) on synthetic datasets that vary in sample count n,

9. Pivot [8] uses 32 cores. Numbers of threads follow prior works [5].



Time (LAN) vs. Sample size n | Time (LAN) vs. Depth D | Time (LAN) vs. Feature size $m_0 = m_1$

Time (WAN) vs. Sample size n | Time (WAN) vs. Depth D | Time (WAN) vs. Feature size $m_0 = m_1$

Figure 10. Scalability of AnonGBDT$^{\text{OTSA}}$ vs. Squirrel ($T = 10$, $n = 10^5$, $m = 10$, $B = 16$, $D = 5$, and 8 threads)

TABLE 6. RUNNING TIME OF AnonGBDT$^{\text{OTSA}}$ AND PRIOR PRIVATE (BUT "LEAKY") SOLUTIONS ($T = 10$)

Approach	Parameters	Settings	Time (s)
Ours	$n = 5 \times 10^4$	LAN	**39.0**
[5]	$D = 4$, $B = 8$	6 threads	60.0
[8]	$m_0 = 8$, $m_1 = 7$		1680.0
Ours	$n = 2 \times 10^5$	LAN	**96.5**
[5]	$D = 4$, $B = 8$	6 threads	111.0
[8]	$m_0 = 8$, $m_1 = 7$		448.0
Ours	$n = 1.4 \times 10^5$	LAN	139.1
[5]	$D = 5, B = 10$	32 threads[9]	**114.0**
[7]	$m_0 = 7, m_1 = 16$		476.0
Ours	$n = 1.4 \times 10^5$	100Mbps	668.7
[5]	$D = 5, B = 10$	32 threads[9]	**400.0**
[7]	$m_0 = 7, m_1 = 16$		1510.0

TABLE 7. F1 SCORE COMPARISON

Dataset	Ours	Squirrel	Pivot	HEP-XGB	XGB-Boost
breast-cancer	**0.920**	0.917	0.919	0.889	0.947
phishing	**0.957**	0.957	0.957	0.951	0.961
a9a	**0.654**	0.651	0.653	0.643	0.654
cod-rna	**0.882**	0.402	0.408	0.403	0.882
skin_nonskin	**0.988**	0.742	0.743	0.741	0.989
covtype.binary	**0.782**	0.556	0.572	0.552	0.796

feature count m, and tree depth D. Figure 10 reports runtime and traffic; Squirrel's numbers are directly quoted.

Thanks to FastPackLWEs and the other engineering optimizations (Section 5.4), our LAN runtime is on par with Squirrel. Because the improved sigmoid scales sublinearly with n, our scheme grows more gracefully as the sample size increases. Communication rises linearly with n, but its growth with m and D is markedly flatter: i) dual-circuit-PSI allows us to use the communication-friendly $\mathcal{F}_{\text{BinMatVec}}$, keeping the cost in m low, and ii) our $\mathcal{F}_{\text{BOIS}}$ reduces the indicator-sync complexity from $\mathcal{O}(2^D)$ to $\mathcal{O}(D)$. Under WAN conditions, runtime is 2–4× higher than in LAN, consistent with the added latency and reduced bandwidth.

7. Discussion

7.1. Gradient-based One-Side Sampling (GOSS)

LightGBM [23] accelerates histogram construction by prioritizing "informative" samples with large (first-order) gradient magnitudes. Given the per-sample gradients $\{g_i\}_{i=1}^n$ at a node, GOSS first sorts the samples by $|g_i|$ in descending order. It keeps the top fraction a (high-gradient set $\mathbb{H}$) *intact*. From the remaining set $\mathbb{L}$, it randomly samples a fraction b. To correct the sampling bias, it multiplies the gradients and Hessians of the sampled low-gradient points by $(1-a)/(1-b)$ before building the histogram. This reduces the working-set size from the sample size n to $|\mathbb{H}| + b|\mathbb{L}|$, speeding up training in the plaintext setting.

Limitations in Secure Two-party Computation: Identifying H requires an *oblivious sort* of the secret-shared gradients, which costs $O(n \log n)$ time and substantial bandwidth (*cf.*, ∼4GB for one million records with $\ell = 32$ [43]). The subsequent random sampling must also be oblivious, adding

further overhead. Consequently, the gains of GOSS in the clear are outweighed by the expense of these secure primitives, making it unsuitable for our secure GBDT pipeline.

7.2. Private-ID

Private-ID protocols [17], [44] return *pseudo-random* (or secret-shared) identifiers for the intersection, rather than a Boolean mask over *all* records (even those not in the intersection) as in circuit-PSI. They have two limitations.

Cardinality leakage: They explicitly reveal $|\mathbf{ID}_0 \cap \mathbf{ID}_1|$, whereas circuit-PSI does not. Hiding $|\mathbf{ID}_0 \cap \mathbf{ID}_1|$ motivates threshold PSI designs [45] that return intersection only if its size exceeds a threshold, whereas prior threshold PSI protocols may leak this size.

Loss of ordering: Neither party learns how the output IDs align with its local feature matrix $\mathbf{M}$; otherwise, the true IDs could be inferred from the feature order. As a result, any subsequent operations must operate on *secret-shared* versions $\langle \mathbf{M} \rangle_l^{\mathsf{B}}$. For secure GBDT, this means gradient histograms must be built with the OT-heavy multiplexer, replicating the inefficiency of our base design.

In contrast, circuit-PSI reveals the ID-feature correspondence to the *receiver*. With our elaborated dual-circuit-PSI design, both parties possess a plaintext copy of their own $\mathbf{M}$ aligned to the intersection, enabling both sides to use the more communication-efficient $\mathcal{F}_{\mathsf{BinMatVec}}$.

7.3. Hiding the Splits

All known secure two-party decision-tree protocols, including ours (with data alignment hidden), reveal the best splits at internal nodes [5], [7], [8], [46], a leakage previously noted [47]. A recent work [48] attempts to mitigate this by obliviously permuting the raw matrix $\mathbf{M}$ to $\langle \mathbf{M}' \rangle^{\mathsf{A}}$, secret-sharing a one-hot vector $\langle \mathbf{s} \rangle^{\mathsf{A}}$ for the chosen bin, and revealing, for each sample i, the indicator $\mathbf{I}_s[i] = \mathbf{M}'[i] \cdot \mathbf{s}$ that determines the branch. Since the permutation is hidden, the split index is assumed private. Yet, the protocol leaks $|\mathbf{ID}_0 \cap \mathbf{ID}_1|$ and enables each party to correlate revealed left/right counts with its own prefix matrix $\widetilde{\mathbf{M}}$, allowing recovery of the feature/bin and contradicting privacy claims.

A direct remedy is to keep split indices secret-shared. Three-party designs [49], [50], [51] realize this via group-based data structures and per-layer oblivious sorting, but such sorting is prohibitively expensive with two parties. Devising an efficient 2-party variant remains an open problem.

8. Related Work

Some federated-learning (FL) frameworks target private GBDT training. SecureBoost(+) [52] employs Paillier homomorphic encryption, and VF^{2}Boost [53] refines the pipeline for large datasets. They only protect the *gradient aggregation*, leaving internal artifacts, including split points, leaf updates, *etc.*, in plaintext. Such leakage enables inference attacks [54], [55], for instance, if an income feature determines a split, a party lacking that feature can deduce a sample's income range from its branch assignment. Takahashi *et al.* [55] use local differential privacy to curb label leakage. Although computationally light, differential privacy adds noise that trades accuracy for privacy.

Secure multiparty computation has been widely adopted. Lindell and Pinkas [46] propose secure multiparty computation for decision-tree training using garbled circuits and OT. Hoogh *et al.* [56] use secret sharing for multiparty GBDT training. Abspoel *et al.* [57] propose an oblivious sorting network for GBDT training with continuous feature values. Due to high communication costs, these designs are inefficient for large datasets. Solutions using function secret sharing [6], [58] achieve efficient online decision tree training, but they require offline correlated random number generation to support secure multiplication.

The most relevant work [5], [7], [8] to ours is the hybrid approaches. Pivot [8] incorporates threshold Paillier HE and secret sharing for tree training. Fang *et al.* [7] propose private large-scale XGBoost training. However, their scheme heavily relies on a semi-honest third party, *e.g.*, trusted execution environments; otherwise, the performance will degrade significantly. Squirrel [5] improves the histogram-calculation efficiency via RLWE-based (instead of Paillier) HE and proposes a new method to approximate the sigmoid function using a Fourier series. Instead, its follow-up [59] realizes pick-then-sum with pure secret sharing, shifting most costs to an offline phase. See the systematization of Chatel *et al.* [60] for related results, including earlier work.

Some selected schemes develop anonymous vertical FL. Sun *et al.* [61] suggest using private set union instead of PSI to obscure the common IDs and generating synthetic features for samples outside the intersection, which can degrade model accuracy. OpenVFL [62] proposes a labeled PSI that returns only homomorphically encrypted features within the intersection. This conceals the common elements and confines the subsequent training to the encrypted domain under homomorphic encryption, which is costly when GBDT pays heavily for secure comparisons. OpenVFL [62] implemented logistic regression only and is not open source.

9. Conclusion

Existing private GBDT protocols run standard PSI once to align the two datasets on their intersection, then train on the aligned records, which inevitably discloses the intersection. Circuit-PSI can avoid this leakage in principle, but its asymmetric interface clashes with order-dependent training flows, and embedding the training loop as a generic circuit makes it impractically slow.

We introduce AnonGBDT, the first anonymous GBDT framework. The base variant adds two new primitives: $\mathcal{F}_{\mathsf{OIS}}$ for indicator synchronization, and $\mathcal{F}_{\mathsf{mux}}$-based gradient aggregation. Although more private, it incurs considerable communication and memory overheads.

Our final scheme AnonGBDT$^{\mathsf{OTSA}}$ resolves these bottlenecks through a dual-circuit-PSI architecture that symmetrizes ownership of the ID-feature mapping. It enables

efficient $\mathcal{F}_{\mathsf{BinMatVec}}$ aggregation, further accelerated by our FastPackLWEs subroutine. Moreover, we propose a batched OPPRF-based OIS protocol. Empirically, AnonGBDT$^{\mathsf{OTSA}}$ is up to an order of magnitude faster than the base design.

Beyond GBDT, our dual-circuit-PSI can integrate with other vertical-learning frameworks, *e.g.*, Split-NN [37], to add anonymity at modest overheads.

References

[1] S. Cao, X. Yang, C. Chen, J. Zhou, X. Li, and Y. Qi, "TitAnt: Online real-time transaction fraud detection in Ant Financial," *Proc. VLDB Endow.*, vol. 12, no. 12, pp. 2082–2093, 2019.

[2] D. J. Hunter and C. Holmes, "Where medical statistics meets artificial intelligence," *New England Journal of Medicine*, vol. 389, no. 13, pp. 1211–1219, 2023.

[3] X. Ling, W. Deng, C. Gu, H. Zhou, C. Li, and F. Sun, "Model ensemble for click prediction in Bing search ads," in *WWW (Companion)*, 2017, pp. 689–698.

[4] L. K. L. Ng and S. S. M. Chow, "SoK: Cryptographic neural-network computation," in *S&P*, 2023, pp. 497–514.

[5] W. Lu, Z. Huang, Q. Zhang, Y. Wang, and C. Hong, "Squirrel: A scalable secure two-party computation framework for training gradient boosting decision tree," in *USENIX Security*, 2023, pp. 6435–6451.

[6] H. Chen, H. Li, Y. Wang, M. Hao, G. Xu, and T. Zhang, "PriVDT: An efficient two-party cryptographic framework for vertical decision trees," *IEEE Trans. Inf. Forensics Secur.*, vol. 18, pp. 1006–1021, 2023.

[7] W. Fang, D. Zhao, J. Tan, C. Chen, C. Yu, L. Wang, L. Wang, J. Zhou, and B. Zhang, "Large-scale secure XGB for vertical federated learning," in *CIKM*, 2021, pp. 443–452.

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

[9] M. J. Freedman, K. Nissim, and B. Pinkas, "Efficient private matching and set intersection," in *EUROCRYPT*, 2004, pp. 1–19.

[10] V. Kolesnikov, R. Kumaresan, M. Rosulek, and N. Trieu, "Efficient batched oblivious PRF with applications to private set intersection," in *CCS*, 2016, pp. 818–829.

[11] Y.-A. D. Montjoye, L. Radaelli, V. K. Singh, and A. S. Pentland, "Unique in the shopping mall: On the reidentifiability of credit card metadata," *Science*, vol. 347, no. 6221, pp. 536–539, 2015.

[12] M. Ion, B. Kreuter, A. E. Nergiz, S. Patel, S. Saxena, K. Seth, M. Raykova, D. Shanahan, and M. Yung, "On deploying secure computing: Private intersection-sum-with-cardinality," in *EuroS&P*, 2020, pp. 370–389.

[13] B. Pinkas, T. Schneider, O. Tkachenko, and A. Yanai, "Efficient circuit-based PSI with linear communication," in *EUROCRYPT Part III*, 2019, pp. 122–153.

[14] P. Rindal and P. Schoppmann, "VOLE-PSI: Fast OPRF and circuit-PSI from vector-OLE," in *EUROCRYPT Part II*, 2021, pp. 901–930.

[15] S. Raghuraman and P. Rindal, "Blazing fast PSI from improved OKVS and subfield VOLE," in *CCS*, 2022, pp. 2505–2517.

[16] J. H. M. Ying, S. Cao, G. S. Poh, J. Xu, and H. W. Lim, "PSI-Stats: Private set intersection protocols supporting secure statistical functions," in *ACNS*, 2022, pp. 585–604.

[17] G. Garimella, P. Mohassel, M. Rosulek, S. Sadeghian, and J. Singh, "Private set operations from oblivious switching," in *PKC Part II*, 2021, pp. 591–617.

[18] J. P. K. Ma and S. S. M. Chow, "Secure-computation-friendly private set intersection from oblivious compact graph evaluation," in *AsiaCCS*, 2022, pp. 1086–1097.

[19] D. Rathee, M. Rathee, N. Kumar, N. Chandran, D. Gupta, A. Rastogi, and R. Sharma, "CrypTFlow2: Practical 2-party secure inference," in *CCS*, 2020, pp. 325–342.

[20] H. Chen, W. Dai, M. Kim, and Y. Song, "Efficient homomorphic conversion between (ring) LWE ciphertexts," in *ACNS*, 2021, pp. 460–479.

[21] J. P. K. Ma, R. K. H. Tai, Y. Zhao, and S. S. M. Chow, "Let's stride blindfolded in a forest: Sublinear multi-client decision trees evaluation," in *NDSS*, 2021.

[22] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *KDD*, 2016, pp. 785–794.

[23] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T. Liu, "LightGBM: A highly efficient gradient boosting decision tree," in *NeurIPS*, 2017, pp. 3146–3154.

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

[25] D. Beaver, "Efficient multiparty protocols using circuit randomization," in *CRYPTO*, 1991, pp. 420–432.

[26] O. Catrina and A. Saxena, "Secure computation with fixed-point numbers," in *FC*, 2010, pp. 35–50.

[27] V. Kolesnikov, A. Sadeghi, and T. Schneider, "Improved garbled circuit building blocks and applications to auctions and computing minima," in *CANS*, 2009, pp. 1–20.

[28] M. O. Rabin, "How to exchange secrets with oblivious transfer," IACR Cryptol. ePrint Arch. 2005/187, 2005.

[29] K. Yang, C. Weng, X. Lan, J. Zhang, and X. Wang, "Ferret: Fast extension for correlated OT with small communication," in *CCS*, 2020, pp. 1607–1626.

[30] Z. Brakerski and V. Vaikuntanathan, "Fully homomorphic encryption from ring-lwe and security for key dependent messages," in *CRYPTO*, 2011, pp. 505–524.

[31] R. Canetti, "Security and composition of multiparty cryptographic protocols," *J. Cryptol.*, vol. 13, no. 1, pp. 143–202, 2000.

[32] Y. Zheng, Q. Zhang, S. S. M. Chow, Y. Peng, S. Tan, L. Li, and S. Yin, "Secure softmax/sigmoid for machine-learning computation," in *ACSAC*, 2023, pp. 463–476.

[33] A. Y. L. Kei and S. S. M. Chow, "SHAFT: Secure, handy, accurate, and fast transformer inference," in *NDSS*, 2025.

[34] A. Amit, P. Stanislav, R. Mariana, S. Phillipp, and S. Karn, "Communication-efficient secure logistic regression," in *EuroS&P*, 2024, pp. 440–467.

[35] Y. Zhao and S. S. M. Chow, "Are you the one to share? Secret transfer with access structure," *Proc. Priv. Enhancing Technol.*, vol. 2017, no. 1, pp. 149–169, 2017.

[36] Z. Wu, J. Hou, Y. Diao, and B. He, "Federated transformer: Multi-party vertical federated learning on practical fuzzily linked data," in *NeurIPS*, 2024.

[37] O. Gupta and R. Raskar, "Distributed learning of deep neural network over multiple agents," *J. Netw. Comput. Appl.*, vol. 116, pp. 1–8, 2018.

[38] SecretFlow, "Cheetah: Lean and fast secure two-party deep neural network inference," https://github.com/secretflow/spu/tree/main/libspu/mpc/cheetah, 2024.

[39] J. H. Cheon, A. Kim, M. Kim, and Y. S. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *ASIACRYPT Part I*, 2017, pp. 409–437.

[40] K. Laine, "Microsoft SEAL (release 4.1)," https://github.com/Microsoft/SEAL, 2023, Microsoft Research, Redmond, WA, USA.

[41] C.-J. Lin, "LIBSVM data: Classification (binary class)," https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html, 2024.

[42] D. D. M. L. Community, "eXtreme gradient boosting," https://github.com/dmlc/xgboost, 2024.

[43] S. Peceny, S. Raghuraman, P. Rindal, and H. Shah, "Efficient permutation correlations and batched random access for two-party computation," in *PKC Part IV*, 2025, pp. 76–109.

[44] P. Buddhavarapu, A. Knox, P. Mohassel, S. Sengupta, E. Taubeneck, and V. Vlaskin, "Private matching for compute," IACR Cryptol. ePrint Arch. 2020/599, 2020.

[45] Y. Zhao and S. S. M. Chow, "Can you find the one for me?" in *WPES, co-located with CCS*, 2018, pp. 54–65.

[46] Y. Lindell and B. Pinkas, "Privacy preserving data mining," in *CRYPTO*, 2000, pp. 36–54.

[47] Z. Zhu and W. Du, "Understanding privacy risk of publishing decision trees," in *DBSec*, 2010, pp. 33–48.

[48] Z. Han, X. Cheng, W. Zhao, J. Fu, Z. He, and S. Su, "SecureXGB: A secure and efficient multi-party protocol for vertical federated XGBoost," *SIGMOD*, vol. 3, no. 1, pp. 1–26, 2025.

[49] K. Hamada, D. Ikarashi, R. Kikuchi, and K. Chida, "Efficient decision tree training with new data structure for secure multi-party computation," *Proc. Priv. Enhancing Technol*, pp. 343–364, 2023.

[50] D. Bhardwaj, S. Saravanan, N. Chandran, and D. Gupta, "Securely training decision trees efficiently," in *CCS*, 2024.

[51] G. Lin, W. Han, W. Ruan, R. Zhou, L. Song, B. Li, and Y. Shao, "Ents: An efficient three-party training framework for decision trees by communication optimization," in *CCS*, 2024, pp. 4376–4390.

[52] K. Cheng, T. Fan, Y. Jin, Y. Liu, T. Chen, D. Papadopoulos, and Q. Yang, "SecureBoost: A lossless federated learning framework," *IEEE Intell. Syst.*, vol. 36, no. 6, pp. 87–98, 2021.

[53] F. Fu, Y. Shao, L. Yu, J. Jiang, H. Xue, Y. Tao, and B. Cui, "VF^{2}Boost: Very fast vertical federated gradient boosting for cross-enterprise learning," in *SIGMOD*, 2021, pp. 563–576.

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

[55] H. Takahashi, J. Liu, and Y. Liu, "Eliminating label leakage in tree-based vertical federated learning," arXiv:2307.10318, 2023.

[56] S. de Hoogh, B. Schoenmakers, P. Chen, and H. op den Akker, "Practical secure decision tree learning in a teletreatment application," in *FC*, 2014, pp. 179–194.

[57] M. Abspoel, D. Escudero, and N. Volgushev, "Secure training of decision trees with continuous attributes," *Proc. Priv. Enhancing Technol.*, vol. 2021, no. 1, pp. 167–187, 2021.

[58] Y. Jiang, F. Mei, T. Dai, and Y. Li, "SiGBDT: Large-scale gradient boosting decision tree training via function secret sharing," in *AsiaCCS*, 2024, pp. 274–288.

[59] T. Dai, Y. Jiang, Y. Li, and F. Mei, "NodeGuard: A highly efficient two-party computation framework for training large-scale gradient boosting decision tree," in *Deep Learning Security and Privacy (DLSP) Workshop, co-located with S&P*, 2024, pp. 95–103.

[60] S. Chatel, A. Pyrgelis, J. R. Troncoso-Pastoriza, and J. Hubaux, "SoK: Privacy-preserving collaborative tree-based model learning," *Proc. Priv. Enhancing Technol.*, vol. 2021, no. 3, pp. 182–203, 2021.

[61] J. Sun, X. Yang, Y. Yao, A. Zhang, W. Gao, J. Xie, and C. Wang, "Vertical federated learning without revealing intersection membership," arXiv:2106.05508, 2021.

[62] Y. Yang, X. Chen, Y. Pan, J. Shen, Z. Cao, X. Dong, X. Li, J. Sun, G. Yang, and R. H. Deng, "OpenVFL: A vertical federated learning framework with stronger privacy-preserving," *IEEE Trans. Inf. Forensics Secur.*, vol. 19, pp. 9670–9681, 2024.

[63] D. Rathee, T. Schneider, and K. K. Shukla, "Improved multiplication triple generation over rings via RLWE-based AHE," in *CANS*, 2019, pp. 347–359.

[64] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(Leveled) fully homomorphic encryption without bootstrapping," *ACM Trans. Comput. Theory*, vol. 6, no. 3, pp. 13:1–13:36, 2014.

[65] C. Zhang, Y. Chen, W. Liu, M. Zhang, and D. Lin, "Linear private set union from multi-query reverse private membership test," in *USENIX Security*, 2023, pp. 337–354.

Appendix A. Optimizations in Sigmoid

A.1. Secure Decimal Multiplication for Trigonometric Functions

$\mathcal{F}_{\mathsf{mul}}$ relies on Beaver triples [25] for $a \cdot b = c$ in secret shares, which can be efficiently implemented in RLWE-based HE [63], *e.g.*, Brakerski–Gentry–Vaikuntanathan [64]. Specifically, P_0 generates random $a \leftarrow_{\$} \mathbb{Z}_{2^{64}}$ and sends ciphertext $[\![a]\!]$ to P_1. P_1 generates another random $b \leftarrow_{\$} \mathbb{Z}_{2^{64}}$ and gets $[\![c]\!] = [\![a]\!] \cdot b$. However, the ciphertext length N is usually 8192, and the plaintext modulus is no more than a 32-bit prime number (for efficient homomorphic operations), which cannot encode c with a bit size of 128 to fit $\mathbb{Z}_{2^{64}}$. The literature [63] solves this by splitting the large plaintext modulus p into four smaller p_i where $p = \prod p_i$ via the Chinese remainder theorem (CRT). Four instances (with modulus p_i respectively) of the above protocol are executed to get outputs $c_0, \ldots, c_3$, which can be combined into the output c (with modulus p) via CRT and then truncated into $\mathbb{Z}_{2^{64}}$. Note that each p_i corresponds to Q of bit size.

In our $\mathcal{F}_{\mathsf{sigmoid}}$, the range of floating numbers $\sin(x)$, $\cos(x)$, and $\sin(x) \cdot \cos(x)$ are all within $[-1, 1]$ (encoded as $[-2^f, 2^f]$). Recalling the average error of our sigmoid approximation is 0.002, we can set $f = 15$ in multiplication. Therefore, the plaintext modulus when encrypting $\sin(x)$ and $\cos(x)$ can be less than 2^{32}, using only one p instead of four p_i. The communication/computation cost of one $\mathcal{F}_{\mathsf{mul}}$ in $\mathcal{F}_{\mathsf{sigmoid}}$ is now only $1/4$ of the original $\mathcal{F}_{\mathsf{mul}}$.

A.2. Configuration of Fourier Series

We use the following coefficients and parameters

$$a_0 = 0.5, \qquad a_1 = 1.642327,$$
$$a_2 = -1.070336, \qquad a_3 = 0.5510985,$$

and $L = 3$ such that the Fourier series output is bounded by 1 within $[-5.6, 5.6]$. Figure 11 shows the comparison between our sigmoid with Squirrel and the ground truth.

A.3. Complexity Comparison

Table 8 compares our $\mathcal{F}_{\mathsf{sigmoid}}$ with the corresponding protocol of Squirrel. With $\lambda = 128$, $f = 20$, and $\delta = 4$, our total communication is $2,976$ bits (vs. $19,816$ for Squirrel).

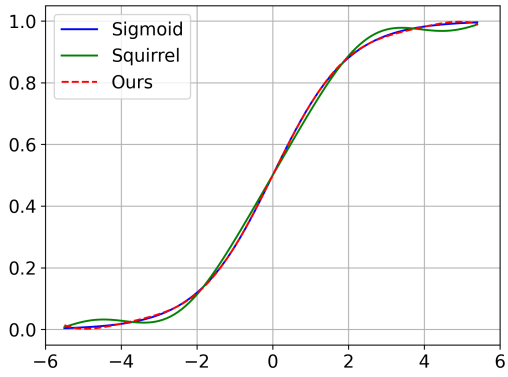


Figure 11. Comparison of sigmoid approximations

TABLE 8. AMORTIZED BIT COMMUNICATION (COMM.) COMPLEXITY COMPARISON OF $\mathcal{F}_{\mathsf{sigmoid}}$ BETWEEN OURS AND SQUIRREL'S

$\mathcal{F}_{\mathsf{sigmoid}}$	Ours	Squirrel
Comm. complexity of $\mathcal{F}_{\mathsf{mul}}$	288	$288 \cdot 4$
Comm. complexity of two $\mathcal{F}_{\mathsf{greater}}$	$\approx 1384 - 4\log_2\lfloor\frac{f-\delta}{4}\rfloor - 32\lfloor\frac{f-\delta}{4}\rfloor$	≈ 1384
# of terms in Fourier series	4	9
Comm. complexity of $\mathcal{F}_{\mathsf{sigmoid}}$	$\approx 1384 - 4\log_2\lfloor\frac{f-\delta}{4}\rfloor - 32\lfloor\frac{f-\delta}{4}\rfloor + 288 \cdot 6$	$\approx 1384 + 288 \cdot 4 \cdot 16$

Appendix B. Batched OPPRF-based OIS (BOIS)

The functionality and protocol of batched OPPRF-based BOIS are shown in Figure 12.

For completeness, we note an alternative realization of $\mathcal{F}_{\mathsf{BOIS}}$ via oblivious transfer for a sparse array (OTSA) [35], which provides an alternative view of the OPPRF step as a sparse-array transfer. Namely, it stores the programmed shares at the same bucket-index domain and uses fixed-length fillers sampled from the output range.

For comparison of array-transfer abstractions, Zhang *et al.* [65] compare OTSA with oblivious key-value stores [15] and observe that OTSA additionally hides the query-to-output correspondence.

Appendix C. Anonymous Batch Inference

The functionality and the protocol of our anonymous batch inference are shown in Figure 13.

Appendix D. Security Proof

Theorem D.1. *Π_{OIS} (Figure 3) is a secure protocol, which follows Definition 3.1 under ($\mathcal{F}_{\mathsf{OT}}$, $\mathcal{F}_{\mathsf{and}}$)-hybrid.*

Proof for Theorem D.1 (Sketch). **Case 1: P_0 is corrupted**: The simulator $\mathcal{S}$ interacts with P_0 as follows.

Functionality $\mathcal{F}_{\mathsf{BOIS}}$

P_c**'s Input:** $\{(z_*^{(k)}, u_*^{(k)})\}_{k\in\mathbf{k}}$, $\{\langle \mathbf{b}^{(k,0)}\rangle_c^{\mathsf{B}}, \langle \mathbf{b}^{(k,1)}\rangle_c^{\mathsf{B}}\}_{k\in\mathbf{k}}$, $\widetilde{\mathbf{M}}_c$, and the simple hash table $\mathbf{T}_{\mathsf{sh}}^{1-c}$.
P_{1-c}**'s Input:** $\{\langle \mathbf{b}^{(k,0)}\rangle_{1-c}^{\mathsf{B}}, \langle \mathbf{b}^{(k,1)}\rangle_{1-c}^{\mathsf{B}}\}_{k\in\mathbf{k}}$, and the cuckoo hash table $\mathbf{T}_{\mathsf{ch}}^{1-c}$.
Public parameters: A set of nodes $\mathbf{k}$ to be synchronized
Output: $\{\langle \mathbf{b}^{(2k,i)}\rangle^{\mathsf{B}}, \langle \mathbf{b}^{(2k+1,i)}\rangle^{\mathsf{B}}\}_{i\in\{0,1\},k\in\mathbf{k}}$.

Protocol Π_{BOIS}

1: **for** each node $k \in \mathbf{k}$ **do**
2: P_c samples $\langle \mathbf{b}_*^{(k,c)}\rangle_c^{\mathsf{B}} = \mathbf{r}^c \leftarrow_{\$} \{0,1\}^{(1+\epsilon)n}$.
3: P_c sets $\mathbf{b}_*^{(k,c)} = \widetilde{\mathbf{M}}_c[z_*^{(k)}B + u_*^{(k)}]$.
4: P_c shares $\langle \mathbf{b}_*^{(k,c)}\rangle_{1-c}^{\mathsf{B}} \leftarrow \mathbf{b}_*^{(k,c)} \oplus \mathbf{r}^c$ with P_{1-c}.
5: P_l: $\langle \mathbf{b}^{(2k,c)}\rangle_l^{\mathsf{B}} = \mathcal{F}_{\mathsf{and}}(\langle \mathbf{b}_*^{(k,c)}\rangle_l^{\mathsf{B}}, \langle \mathbf{b}^{(k,c)}\rangle_l^{\mathsf{B}})$ and $\langle \mathbf{b}^{(2k+1,c)}\rangle_l^{\mathsf{B}} = \langle \mathbf{b}^{(2k,c)}\rangle_l^{\mathsf{B}} \oplus \langle \mathbf{b}^{(k,c)}\rangle_l^{\mathsf{B}}$.
6: P_c samples $\mathbf{r}^{1-c} \leftarrow_{\$} \{\mathbb{Z}_{2^{|\mathbf{k}|}}\}^{(1+\epsilon)n}$, and initializes packed assignment $\widetilde{\mathbf{b}}'_* = \mathbf{0}^{(1+\epsilon)n}$.
7: **for** each node $k \in \mathbf{k}$ **do**
8: P_c "reorders" $\mathbf{b}_*^{(k,c)}$ to $\mathbf{b}'_*$ to align with $\mathbf{ID}_c$.
9: P_c packs $\widetilde{\mathbf{b}}'_* = \widetilde{\mathbf{b}}'_* \oplus (\mathbf{b}'_* \ll k)$.
10: P_c sets $\langle \mathbf{b}_*^{(k,1-c)}\rangle_c^{\mathsf{B}} = (\mathbf{r}^{1-c} \gg k)\&\mathbf{1}^{(1+\epsilon)n}$.
11: $\forall j \in [0,e), \forall i \in [0,(1+\epsilon)n)$: P_c sets a table $\mathbf{T} = \{(\mathbf{T}_{\mathsf{sh}}^{1-c}[i,j], \widetilde{\mathbf{b}}'_*[i'] \oplus \mathbf{r}^{1-c}[i]\}$, and $\mathbf{ID}_c[i']$ is placed in $\mathbf{T}_{\mathsf{sh}}^{1-c}[i,j]$ via simple hashing.
12: P_l runs $\mathcal{F}_{\mathsf{OPPRF}}(\{\mathbf{T}\}, \{\mathbf{T}_{\mathsf{ch}}^{1-c}\})$; P_{1-c} gets $\widetilde{\mathbf{b}}_*^{(1-c)}$
13: **for** each node $k \in \mathbf{k}$ **do**
14: P_{1-c}: $\langle \mathbf{b}_*^{(k,1-c)}\rangle_{1-c}^{\mathsf{B}} = (\widetilde{\mathbf{b}}_*^{(1-c)} \gg k)\&\mathbf{1}^{(1+\epsilon)n}$.
15: P_l: $\langle \mathbf{b}^{(2k,1-c)}\rangle_l^{\mathsf{B}} = \mathcal{F}_{\mathsf{and}}(\langle \mathbf{b}_*^{(k,1-c)}\rangle_l^{\mathsf{B}}, \langle \mathbf{b}^{(k,1-c)}\rangle_l^{\mathsf{B}})$.
16: P_l: $\langle \mathbf{b}^{(2k+1,1-c)}\rangle_l^{\mathsf{B}} = \langle \mathbf{b}^{(2k,1-c)}\rangle_l^{\mathsf{B}} \oplus \langle \mathbf{b}^{(k,1-c)}\rangle_l^{\mathsf{B}}$.

Figure 12. Ideal functionality and protocol of BOIS

1) If (z_*, u_*) is owned by P_0, $\mathcal{S}$ receives $\langle \mathbf{b}_*\rangle_1^{\mathsf{B}}$ from P_0 (Line 3). Then $\mathcal{S}$ computes $\langle \mathbf{b}_*\rangle_0^{\mathsf{B}} = \langle \mathbf{b}_*\rangle_1^{\mathsf{B}} \oplus \widetilde{\mathbf{M}}_0[z_*B + u_*]$; else $\mathcal{S}$ plays the sender of $\mathcal{F}_{\mathsf{OT}}$ with input of the XOR of randomly chosen $\mathbf{r}(= \langle \mathbf{b}_*\rangle_0^{\mathsf{B}})$ and $\langle \widetilde{\mathbf{M}}_1\rangle_0^{\mathsf{B}}$ as in Line 8.
2) $\mathcal{S}$ plays the role of $\mathcal{F}_{\mathsf{and}}$ with inputs of $\langle \mathbf{b}_*\rangle_0^{\mathsf{B}}$, $\langle \mathbf{b}\rangle_0^{\mathsf{B}}$ to obtain $\langle \mathbf{b}^L\rangle_0^{\mathsf{B}}$, then sets $\langle \mathbf{b}^R\rangle_0^{\mathsf{B}} = \langle \mathbf{b}^L\rangle_0^{\mathsf{B}} \oplus \langle \mathbf{b}\rangle_0^{\mathsf{B}}$ as Line 11.
To prove the simulation is indistinguishable from the real protocol, we consider the following hybrids.
$\mathbf{Hybrid}_0$. The same as the real protocol.
$\mathbf{Hybrid}_1$. Lines 8–9 in Protocol Π_{OIS} use replaced $\mathcal{F}_{\mathsf{OT}}$.
$\mathbf{Hybrid}_2$. Line 11 in Protocol Π_{OIS} uses replaced $\mathcal{F}_{\mathsf{and}}$.
$\mathbf{Hybrid}_3$. The same as the execution of $\mathcal{S}$ above.
Note that $\mathbf{Hybrid}_1 \approx_c \mathbf{Hybrid}_0$ as $\mathcal{F}_{\mathsf{OT}}$ is secure, and $\mathbf{Hybrid}_2 \approx_c \mathbf{Hybrid}_1$ as $\mathcal{F}_{\mathsf{and}}$ is secure. $\mathbf{Hybrid}_3$ is identical to $\mathbf{Hybrid}_2$. Therefore, $\mathcal{S} \approx_c \mathcal{V}_0^{\mathbf{Hybrid}_0}$.
Case 2: P_1 is corrupted: The simulator $\mathcal{S}$ plays the role P_0, and the interaction with P_1 is similar to the above.
In short, in both cases, the $\mathcal{F}_{\mathsf{OT}}$, $\mathcal{F}_{\mathsf{and}}$-hybrid Π_{OIS} protocol is secure in the semi-honest model. □

Proof for Theorem 5.1 (Sketch). **Case 1: The sender P_c is corrupted**: The simulator $\mathcal{S}$ interacts with P_c as follows.

Functionality $\mathcal{F}_{\mathsf{Infer}}$

P_0**'s Input:** $\mathbf{ID}_0$, $\mathbf{X}_0$, best split $\mathbf{C}_0$, weight share $\langle \mathbf{w} \rangle_0^{\mathsf{A}}$.
P_1**'s Input:** $\mathbf{ID}_1$, $\mathbf{X}_1$, best split $\mathbf{C}_1$, weight share $\langle \mathbf{w} \rangle_1^{\mathsf{A}}$.
Output: Prediction probability $\mathbf{p}$ to P_0.

Protocol Π_{Infer}

1: Jointly invoke $\langle \mathbf{q} \rangle_l^{\mathsf{B}} \leftarrow \mathcal{F}_{\mathsf{CPSI}}(\{\mathbf{ID}_0\}, \{\mathbf{ID}_1\})$, P_0 runs as receiver with cuckoo hash table $\mathbf{T}_{\mathsf{ch}}$, P_1 runs as sender with simple hash table $\mathbf{T}_{\mathsf{sh}}$.
2: **for** tree $t \in \mathcal{T}_t$ **do**
3: P_l sets root node's indicator $\mathbf{b}^{(0,\,l)} = \mathbf{1}^{(1+\epsilon)n}$
4: **for** internal node $k \in [1, 2^{D-1})$, each sample i **do**
5: **if** $\mathbf{C}_l[k]$ is not $\perp$ **then**
6: P_l gets $(z_*^{(k)}, u_*^{(k)}) \leftarrow \mathbf{C}_l[k]$
7: **if** $\mathbf{X}_l[i, z_*^{(k)}] \leq$ split value of $z_*^{(k)}, u_*^{(k)}$ **then**
8: P_l: $\mathbf{b}^{(2k,l)}[i] = \mathbf{b}^{(k,l)}[i]$.
9: P_l: $\mathbf{b}^{(2k+1,l)}[i] = 0$.
10: **else**
11: P_l: $\mathbf{b}^{(2k,l)}[i] = 0$.
12: P_l: $\mathbf{b}^{(2k+1,l)}[i] = \mathbf{b}_l^{(k,l)}[i]$.
13: **else** P_l: $\mathbf{b}^{(2k,l)}[i] = \mathbf{b}^{(2k+1,l)}[i] = \mathbf{b}^{(k,l)}[i]$
14: P_0 shares $\mathbf{b}^{(0)}$, P_l gets $\langle \mathbf{b}^{(0)} \rangle_l^{\mathsf{B}}$.
15: P_1: sets $\mathbf{r} \leftarrow_{\$} \{\mathbb{Z}_{2^{\#\text{leaves}}}\}^{(1+\epsilon)n}$, and extracts $\langle \mathbf{b}^{(k,1)} \rangle_1^{\mathsf{B}}$ which is the $k - 2^{D-1}$-th bit of $\mathbf{r}$.
16: P_1 packs all the leaves' indicators $\mathbf{b}^{(k,1)}$ as $\widetilde{\mathbf{b}}'$
17: $\forall i \in [0, (1+\epsilon)n), \forall j \in [0, e)$, P_1 creates a table $\mathbf{T} = \{(\mathbf{T}_{\mathsf{sh}}[i,j], \widetilde{\mathbf{b}}'[i'] \oplus \mathbf{r}[i])\}$ where $\mathbf{ID}[i']$ is placed in $\mathbf{T}_{\mathsf{sh}}$ via simple hashing.
18: P_l runs $\mathcal{F}_{\mathsf{OPPRF}}(\{\mathbf{T}\}, \{\mathbf{T}_{\mathsf{ch}}\})$; P_0 gets $\widetilde{\mathbf{b}}''$.
19: P_l initializes $\langle \mathbf{S} \rangle_l^{\mathsf{B}} = \mathbf{0}^{(1+\epsilon)n}$
20: **for** node $k \in \{2^{D-1}, \ldots, 2^D - 1\}$ **do**
21: P_l extract indicator $\langle \mathbf{b}^{(k,1)} \rangle_l^{\mathsf{B}}$ which is the $(k - 2^{D-1})$-th bit of $\widetilde{\mathbf{b}}''$.
22: P_l: $\langle \mathbf{b}^{(k)} \rangle_l^{\mathsf{B}} = \mathcal{F}_{\mathsf{and}}(\langle \mathbf{b}^{(k,0)} \rangle_l^{\mathsf{B}}, \langle \mathbf{b}^{(k,1)} \rangle_l^{\mathsf{B}})$
23: P_l: $\forall i \in [0, (1+\epsilon)n), \langle \mathbf{S}[i] \rangle_l^{\mathsf{A}} = \langle \mathbf{S}[i] \rangle_l^{\mathsf{A}} + \mathcal{F}_{\mathsf{mux}}(\langle \mathbf{b}^{(k)}[i] \rangle_l^{\mathsf{B}}, \langle \mathbf{w}[k] \rangle_l^{\mathsf{A}})$
24: P_l: $\langle \tilde{\mathbf{y}} \rangle_l^{\mathsf{A}} = \mathcal{F}_{\mathsf{mux}}(\langle \mathbf{q} \rangle_l^{\mathsf{B}}, \langle \mathbf{S} \rangle_l^{\mathsf{A}})$
25: P_0 recovers $\tilde{\mathbf{y}}$ and gets $\mathbf{p} = \mathsf{sigmoid}(\tilde{\mathbf{y}})$.

Figure 13. Functionality and protocol of Inference

1) $\forall k \in \mathbf{k}$, $\mathcal{S}$ receives $\langle \mathbf{b}_*^{(k,c)} \rangle_{1-c}^{\mathsf{B}}$ from P_c and computes $\mathbf{r}^c$ with $(z_*^{(k)}, u_*^{(k)})$ and $\widetilde{\mathbf{M}}_c$. Then $\mathcal{S}$ sets $\langle \mathbf{b}_*^{(k,c)} \rangle_c^{\mathsf{B}} = \mathbf{r}^c$ and plays the role of $\mathcal{F}_{\mathsf{and}}$ with $\langle \mathbf{b}_*^{(k,c)} \rangle_c^{\mathsf{B}}$ and $\langle \mathbf{b}^{(k,c)} \rangle_c^{\mathsf{B}}$. After obtaining $\langle \mathbf{b}^{(2k,c)} \rangle_c^{\mathsf{B}}$, $\mathcal{S}$ computes $\langle \mathbf{b}^{(2k+1,c)} \rangle_c^{\mathsf{B}}$ as Lines 2–5.
2) $\mathcal{S}$ constructs $\{\mathbf{T}\}$ and $\langle \mathbf{b}_*^{k,1-c} \rangle_c^{\mathsf{B}}$ ($\forall k \in \mathbf{k}$) as Lines 6–11. Then $\mathcal{S}$ plays the role of $\mathcal{F}_{\mathsf{OPPRF}}$ with $\{\mathbf{T}\}$.
3) $\forall k \in \mathbf{k}$, $\mathcal{S}$ plays the role of $\mathcal{F}_{\mathsf{and}}$ with inputs $\langle \mathbf{b}_*^{(k,1-c)} \rangle_l^{\mathsf{B}}$ and $\langle \mathbf{b}^{(k,1-c)} \rangle_l^{\mathsf{B}}$, and obtain $\langle \mathbf{b}^{2k+1,1-c} \rangle_c^{\mathsf{B}}$ as Line 15. Finally, $\mathcal{S}$ computes $\langle \mathbf{b}^{2k+1,1-c} \rangle_c^{\mathsf{B}}$ as Line 16.

For indistinguishable simulation, we consider:
$\mathbf{Hybrid}_0$. The same as the real protocol.
$\mathbf{Hybrid}_1$. Line 5 in Π_{BOIS} uses replaced $\mathcal{F}_{\mathsf{and}}$.
$\mathbf{Hybrid}_2$. Line 12 uses replaced $\mathcal{F}_{\mathsf{OPPRF}}$.
$\mathbf{Hybrid}_3$. Lines 15–16 use replaced $\mathcal{F}_{\mathsf{and}}$.
$\mathbf{Hybrid}_4$. The same as the execution of $\mathcal{S}$ above.
Note that $\mathbf{Hybrid}_1 \approx_c \mathbf{Hybrid}_0$ as $\mathcal{F}_{\mathsf{and}}$ is secure; $\mathbf{Hybrid}_2 \approx_c \mathbf{Hybrid}_1$ as $\mathcal{F}_{\mathsf{OPPRF}}$ is secure; $\mathbf{Hybrid}_3 \approx_c \mathbf{Hybrid}_2$ as $\mathcal{F}_{\mathsf{and}}$ is secure; $\mathbf{Hybrid}_4$ is identical to $\mathbf{Hybrid}_3$. Therefore, $\mathcal{S} \approx_c \mathcal{V}_c^{\mathbf{Hybrid}_0}$.

Case 2: The receiver P_{1-c} is corrupted: $\mathcal{S}$ plays the role of P_c and interacts with P_{1-c} like Case 1 except:
1) $\mathcal{S}$ ignores the process of Lines 2–4, and chooses randomly $\langle \mathbf{b}_*^{(k,c)} \rangle_{1-c}^{\mathsf{B}}$ ($\forall k \in \mathbf{k}$) and sends to P_{1-c}.
2) $\mathcal{S}$ ignores the process of Lines 6–11, and plays the role as $\mathcal{F}_{\mathsf{OPPRF}}$ with input $\mathbf{T}_{\mathsf{ch}}^{1-c}$ to obtain $\widetilde{\mathbf{b}}_*^{1-c}$. Then $\mathcal{S}$ extracts $\langle \mathbf{b}_*^{(k,1-c)} \rangle_{1-c}^{\mathsf{B}}$ $\forall k \in \mathbf{k}$ as Line 14.

In short, in both cases, the $\mathcal{F}_{\mathsf{and}}$, $\mathcal{F}_{\mathsf{OPPRF}}$-hybrid Π_{BOIS} protocol is secure in the semi-honest model. □

Proof for Theorem 5.2 (Sketch). **Case 1: P_0 is corrupted**: The simulator $\mathcal{S}$ interacts with P_0 as follows.
1) $\mathcal{S}$ plays the sender and receiver of two $\mathcal{F}_{\mathsf{CPSI}}$ with $\{\mathbf{ID}_0, \mathbf{y}\}$, respectively, as Lines 2–3. Then $\mathcal{S}$ obtains $\langle \mathbf{b}^{(1,0)} \rangle_0^{\mathsf{B}}$, $\langle \mathbf{y}^0 \rangle_0^{\mathsf{A}}$, $\langle \mathbf{b}^{(1,1)} \rangle_0^{\mathsf{B}}$ and $\langle \mathbf{y}^1 \rangle_0^{\mathsf{A}}$.
2) For tree $t \in \mathcal{T}_t$, $\mathcal{S}$ plays the role of $\mathcal{F}_{\mathsf{sigmoid}}$, $\mathcal{F}_{\mathsf{mul}}$, and $\mathcal{F}_{\mathsf{mux}}$ with input $\langle \mathbf{b}^{(1,i)} \rangle_0^{\mathsf{B}}$ and $\langle \mathbf{y}^i \rangle_0^{\mathsf{A}}$ to derive the gradients $\langle \mathbf{g}^{(1,i)} \rangle_0^{\mathsf{A}}$, $\langle \mathbf{h}^{(1,i)} \rangle_0^{\mathsf{A}}$, for $i \in \{0, 1\}$ as Lines 5–6.
3) $\forall$ internal node k of tree t, if k is a root/left node, $\mathcal{S}$ plays the role of $\mathcal{F}_{\mathsf{BinMatVec}}$ alternately to obtain $\langle \tilde{\mathbf{g}}^{(k,i)} \rangle_0^{\mathsf{A}} || \langle \tilde{\mathbf{h}}^{(k,i)} \rangle_0^{\mathsf{A}}$ for $i \in \{0, 1\}$, which are concatenated as Line 13; else $\mathcal{S}$ calculates $\langle \widetilde{\mathbf{g}}^{(k)} \rangle_0^{\mathsf{A}}$ and $\langle \widetilde{\mathbf{h}}^{(k)} \rangle_0^{\mathsf{A}}$ as Line 14.
4) $\forall$ internal node k of tree t, $\mathcal{S}$ plays the role of $\mathcal{F}_{\mathsf{mul}}$ and $\mathcal{F}_{\mathsf{div}}$ with $\langle \tilde{\mathbf{g}}^{(k)} \rangle_0^{\mathsf{A}}$, $\langle \tilde{\mathbf{h}}^{(k)} \rangle_0^{\mathsf{A}}$ to obtain $\langle \mathbf{L}_{\mathsf{sp}} \rangle_0^{\mathsf{A}}$ as Line 15.
5) $\forall$ internal node k of tree t, $\mathcal{S}$ plays the role of $\mathcal{F}_{\mathsf{argmax}}$ with input $\langle \mathbf{L}_{\mathsf{sp}} \rangle_0^{\mathsf{A}}$ to obtain $\langle z_*^{(k)} \rangle_0^{\mathsf{A}}$ and $\langle u_*^{(k)} \rangle_0^{\mathsf{A}}$ as Line 16.
6) $\forall$ internal node k of tree t, $\mathcal{S}$ plays the role of $\mathcal{F}_{\mathsf{greater}}$ with inputs $\langle z_*^{(k)} \rangle_0^{\mathsf{A}}$ to obtain $\langle c \rangle_0^{\mathsf{B}}$, and then open c to recovery $z_*^{(k)}$ and $u_*^{(k)}$ to P_c, which is exactly equal to the programmed outputs in real execution when $c = 0$. Finally, $\mathcal{S}$ appends k into $\mathbf{k}_c$ as Line 17.
7) $\forall$ level d in tree t, $\mathcal{S}$ plays the role of $\mathcal{F}_{\mathsf{BOIS}}$ to obtain $\{\langle \mathbf{b}^{(2k,i)} \rangle_0^{\mathsf{B}}, \langle \mathbf{b}^{(2k+1,i)} \rangle_0^{\mathsf{B}}\}_{k \in \mathbf{k}_i, i \in \{0,1\}}$ as Lines 18–19.
8) $\mathcal{S}$ plays the role of $\mathcal{F}_{\mathsf{mux}}$ with the inputs $\langle \mathbf{b}^{(2k,i)} \rangle_0^{\mathsf{B}}$, $\langle \mathbf{g}^{(k,i)} \rangle_0^{\mathsf{A}}$ and $\langle \mathbf{h}^{(k,i)} \rangle_0^{\mathsf{A}}$ to update the gradients of left child nodes $\langle \mathbf{g}^{(2k,i)} \rangle_0^{\mathsf{A}}$, $\langle \mathbf{h}^{(2k,i)} \rangle_0^{\mathsf{A}}$. Then update the gradients of right child nodes as Line 21.
9) For tree t, $\mathcal{S}$ plays the role of $\mathcal{F}_{\mathsf{div}}$ with inputs $\langle \mathbf{g}^{(k,0)} \rangle_0^{\mathsf{A}}, \langle \mathbf{h}^{(k,0)} \rangle_0^{\mathsf{A}}$ to obtain the weight $\langle \mathbf{w} \rangle_0^{\mathsf{A}}$ as Line 22.
10) For tree t, $\mathcal{S}$ plays the role of $\mathcal{F}_{\mathsf{mux}}$ with inputs $\langle \mathbf{b}^{(k,i)} \rangle_0^{\mathsf{B}}$, $\langle \mathbf{w}[k] \rangle_0^{\mathsf{A}}$ to update $\langle \tilde{\mathbf{y}}^i \rangle_0^{\mathsf{A}}$ ($i \in \{0, 1\}$) as Line 23.

To prove the simulation is indistinguishable from the real protocol, we consider the following hybrids.
$\mathbf{Hybrid}_0$. The same as the real protocol.
$\mathbf{Hybrid}_1$. Lines 2–3 ($\Pi_{\mathsf{GBDT}}^{\mathsf{OTSA}}$) use replaced $\mathcal{F}_{\mathsf{CPSI}}$.
$\mathbf{Hybrid}_2$. Lines 5–6 use replaced $\mathcal{F}_{\mathsf{sigmoid}}$, $\mathcal{F}_{\mathsf{mul}}$, $\mathcal{F}_{\mathsf{mux}}$.
$\mathbf{Hybrid}_3$. Lines 11–12 use replaced $\mathcal{F}_{\mathsf{BinMatVec}}$.
$\mathbf{Hybrid}_4$. Line 15 use replaced $\mathcal{F}_{\mathsf{mul}}$ and $\mathcal{F}_{\mathsf{div}}$.
$\mathbf{Hybrid}_5$. Line 16 uses replaced $\mathcal{F}_{\mathsf{argmax}}$.

$\mathbf{Hybrid}_6$. Line 17 uses replaced $\mathcal{F}_{\mathsf{greater}}$.
$\mathbf{Hybrid}_7$. Lines 18–19 use replaced $\mathcal{F}_{\mathsf{BOIS}}$.
$\mathbf{Hybrid}_8$. Line 20 uses replaced $\mathcal{F}_{\mathsf{mux}}$.
$\mathbf{Hybrid}_9$. Line 22 uses replaced $\mathcal{F}_{\mathsf{div}}$.
$\mathbf{Hybrid}_{10}$. Line 23 uses replaced $\mathcal{F}_{\mathsf{mux}}$.
$\mathbf{Hybrid}_{11}$. The same as the execution of $\mathcal{S}$ above.

Note that $\mathbf{Hybrid}_1 \approx_c \mathbf{Hybrid}_0$ as $\mathcal{F}_{\mathsf{CPSI}}$ is secure; $\mathbf{Hybrid}_2 \approx_c \mathbf{Hybrid}_1$ as $\mathcal{F}_{\mathsf{sigmoid}}$ and $\mathcal{F}_{\mathsf{mul}}$ are secure; $\mathbf{Hybrid}_3 \approx_c \mathbf{Hybrid}_2$ as $\mathcal{F}_{\mathsf{BinMatVec}}$ is secure based on the semantic security of RLWE ciphertexts; $\mathbf{Hybrid}_4 \approx_c \mathbf{Hybrid}_3$ as $\mathcal{F}_{\mathsf{mul}}$ and $\mathcal{F}_{\mathsf{div}}$ is secure; $\mathbf{Hybrid}_5 \approx_c \mathbf{Hybrid}_4$ as $\mathcal{F}_{\mathsf{argmax}}$ is secure; $\mathbf{Hybrid}_6 \approx_c \mathbf{Hybrid}_5$ as $\mathcal{F}_{\mathsf{greater}}$ is secure, and the recovered $z_*^{(k)}$ and $u_*^{(k)}$ are programmed into ideal functionality as the identical output between real execution and simulation; $\mathbf{Hybrid}_7 \approx_c \mathbf{Hybrid}_6$ as $\mathcal{F}_{\mathsf{BOIS}}$ is secure; $\mathbf{Hybrid}_8 \approx_c \mathbf{Hybrid}_7$ as $\mathcal{F}_{\mathsf{mux}}$ is secure; $\mathbf{Hybrid}_9 \approx_c \mathbf{Hybrid}_8$ as $\mathcal{F}_{\mathsf{div}}$ is secure; $\mathbf{Hybrid}_{10} \approx_c \mathbf{Hybrid}_9$ as $\mathcal{F}_{\mathsf{mux}}$ is secure. $\mathbf{Hybrid}_{11}$ is identical to $\mathbf{Hybrid}_{10}$. Thus, $\mathcal{S}_0 \approx_c \mathcal{V}_0^{\mathbf{Hybrid}_0}$.

Case 2: P_1 is corrupted: The simulator plays the role of P_0 and the interaction with P_1 is similar for the symmetric tree building of $\Pi_{\mathsf{GBDT}}^{\mathsf{OTSA}}$, except for the simulation of Lines 2–3 in $\Pi_{\mathsf{GBDT}}^{\mathsf{OTSA}}$ as input without labels, which can be simulated as $\mathcal{F}_{\mathsf{CPSI}}$ is secure.

In both cases, under the ($\mathcal{F}_{\mathsf{argmax}}$-, $\mathcal{F}_{\mathsf{BOIS}}$-, $\mathcal{F}_{\mathsf{BinMatVec}}$-, $\mathcal{F}_{\mathsf{CPSI}}$-, $\mathcal{F}_{\mathsf{div}}$-, $\mathcal{F}_{\mathsf{greater}}$-, $\mathcal{F}_{\mathsf{mul}}$-, $\mathcal{F}_{\mathsf{mux}}$-, $\mathcal{F}_{\mathsf{sigmoid}}$)-hybrid, and with semantically-secure RLWE-based encryption, $\Pi_{\mathsf{GBDT}}^{\mathsf{OTSA}}$ is secure in the semi-honest model. □

Appendix E. Meta-Review

The following meta-review was prepared by the program committee for the 2026 IEEE Symposium on Security and Privacy (S&P) as part of the review process as detailed in the call for papers.

E.1. Summary

This paper studies efficient two-party computation for the task of gradient boosting decision tree training and inference. The paper assumes that the data is vertically distributed and thus alignment is required. The most interesting optimization is how the paper optimizes circuit-PSI using homomorphic encryption to reduce the cost of alignment when the payload can be large.

E.2. Scientific Contributions

- Provides a Valuable Step Forward in an Established Field

E.3. Reasons for Acceptance

The paper makes a valuable step towards practical two-party GBDT by using homomorphic encryption to reduce the communication complexity in PSI. This leads to significant savings in their experience.