

Private Learning for Vertical Decision Trees: A Secure, Accurate, and Fast Realization

Anxiao Song , Ke Cheng , Jiaxuan Fu, Shujie Cui , Tao Zhang , Zhao Chang , *Member, IEEE*, and Yulong Shen 

I. INTRODUCTION

Abstract—Private learning for vertical decision trees (PVDT) is an emerging paradigm that allows multiple parties to execute cooperative training and inference of decision trees on vertically partitioned datasets, without revealing either party’s data or model. The state-of-the-art PVDT schemes employ the secret-sharing-based secure multi-party computation (MPC) to admit low computational cost and low bandwidth. Nevertheless, existing schemes need many communication rounds for computing concrete protocols in PVDT, like the less-than comparison, division, etc. This property is not suited for large-communication-latency networks such as WAN. In this work, we present a two-party PVDT framework, called *Swan*, to enable a secure, accurate, and fast realization of vertical decision trees. At the core of *Swan*, we design a secure and parallel protocol for N -input multiplication with one communication round. This forms the cornerstone for a series of secure and communication-efficient computation protocols specifically tailored to less-than comparison and division. Along the way, we use these optimized protocols to refine the training and inference processes of PVDT, achieving a significant reduction in both communication costs and rounds. Experimental results show *Swan* provides top-notch accuracy, and achieves a $10.2\times$ and $2.8\times$ improvement in online training and inference latency over WAN compared to prior art.

Index Terms—Privacy protection, vertical decision tree, secret sharing, secure two-party computation.

Received 11 October 2023; revised 4 June 2025; accepted 10 June 2025. Date of publication 16 June 2025; date of current version 4 November 2025. This work was supported in part by the National Natural Science Foundation of China under Grant 62402358, Grant 62302368, and Grant 62220106004, in part by the Technology Innovation Leading Program of Shaanxi under Grant 2023KXJ-033, in part by the Key R&D Program of Shandong Province of China under Grant 2023CXPT056, in part by the Young Talent Fund of Association for Science and Technology in Shaanxi, China, under Grant 20240138, in part by the Open Topics from the Lion Rock Labs of Cyberspace Security under Grant LRL24004, in part by the Open Project Foundation of Shaanxi Key Laboratory of Information Communication Network and Security under Grant ICNS202301, in part by the Fundamental Research Funds for the Central Universities under Grant ZDRC2202, Grant ZYTS25081, and Grant KYFZ25005, and in part by the Double First-Class Overseas Research Project of Xidian University. (*Corresponding authors: Ke Cheng; Tao Zhang.*)

Anxiao Song, Jiaxuan Fu, Tao Zhang, Zhao Chang, and Yulong Shen are with the School of Computer Science & Technology, Xidian University, Xi’an 710071, China, and also with the Shaanxi Key Laboratory of Network and System Security, Xi’an 710071, China (e-mail: songanxiao@stu.xidian.edu.cn; jxfu_2099@stu.xidian.edu.cn; taozhang@xidian.edu.cn; changzhao@xidian.edu.cn; ylshen@mail.xidian.edu.cn).

Ke Cheng is with the School of Computer Science and Technology, Xidian University, Xi’an 710071, China, and also with Shaanxi Key Laboratory of Information Communication Network and Security, Xi’an University of Posts and Telecommunications, Xi’an 710121, China (e-mail: chengke@xidian.edu.cn).

Shujie Cui is with the Faculty of Information Technology, Monash University, Clayton 3800, Australia (e-mail: shujie.cui@monash.edu).

This article has supplementary downloadable material available at <https://doi.org/10.1109/TDSC.2025.3579742>, provided by the authors.

Digital Object Identifier 10.1109/TDSC.2025.3579742

DECISION trees (DT [1] machine learning, applied to various domains such as e-commerce recommendations [2], spam filtering [3], and medical diagnostics [4]. Constructing an accurate decision tree model requires extensive training data drawn from diverse data owners. However, these training datasets are usually partitioned vertically among each holder, with each possessing disjoint features of the same sample [5]. More importantly, these datasets contain privacy-sensitive information, raising severe privacy concerns for data owners [6].

Motivated by these challenges, private learning for vertical decision trees (PVDT) has been introduced as an emerging paradigm [7], that allows multiple parties with vertically distributed data to jointly build a DT model without revealing their respective data. As illustrated in Fig. 1, an e-commerce company collaborates with a bank to train a DT model for the precise placement of product advertisements to targeted consumers. The bank provides financial data (e.g., account balance), while the e-commerce company contributes online transaction details (e.g., purchase history). When inputting consumer features that are distributed across two parties, the trained model in PVDT predicts the types of products that the consumer is likely to purchase. PVDT can assist the e-commerce company in obtaining more accurate models without revealing the private data of the bank, while the bank can make gains from a pay-per-use approach due to its contribution to the training and inference.

Secure multi-party computation (MPC) is a powerful tool that enables multiple parties to jointly compute a function while keeping their inputs private. In particular, the MPC based on secret sharing (SS) has been widely adopted due to its high efficiency. The state-of-the-art works of Pivot [8] and PriVDT [9] have demonstrated the possibility of training and inference of vertical DT using SS-based MPC techniques. However, there are still several issues that need resolving for the practical application of their works. For instance, Pivot employs hybrid cryptographic protocols combining homomorphic encryption (HE) and additive secret sharing (ASS) to hide private information during the training and inference. Nevertheless, expensive operations for the domain conversion between HE and ASS inevitably make participants suffer from a prohibitive computation overhead. Moreover, we found that performing a secure division operation in Pivot requires 32 communication rounds, rendering it unsuitable for real network environments like WAN. To provide improved efficiency, a recent work of PriVDT employs function

TABLE I
COMPARISON WITH EXISTING SECURE DT APPROACHES FOR TRAINING VERTICAL DATASETS

Approaches	Cryptographic primitive	Computation cost	Communication cost	Leakage
Du and Zhan [7]	ASS	$\mathcal{O}(FV)$	$\mathcal{O}(NFVC\ell)$	Label, Gain, Intermediates
Wang et al. [10]	AHE, GC	$\mathcal{O}(FVC)$	$\mathcal{O}(NFVC(\ell_e + \ell_g))$	Gain, Intermediates
SecureBoost [11]	AHE	$\mathcal{O}(FVC)$	$\mathcal{O}(NFVC\ell_e)$	Gain, Intermediates
Lindell et al. [12]	ASS, GC	$\mathcal{O}(FCV\ell \log \ell + FV \log \ell)$	$\mathcal{O}(NFCV\ell + FVC(\ell^2 \log(\kappa_i \ell) + \log \ell + \ell))$	—
PPDT [13]	AHE, FE	$\mathcal{O}(NFCV)$	$\mathcal{O}(NFVC(\ell_e) + FVC(\ell_e + \ell_s))$	—
Privet [14]	ASS	$\mathcal{O}(FCV\epsilon \log \ell)$	$\mathcal{O}(NFVC\ell + FVC(\epsilon \ell^2 + \ell))$	—
Pivot [8]	ASS, AHE	$\mathcal{O}(FCV(\ell \log \ell) + \log \ell)$	$\mathcal{O}(NFVC(\ell_e + \ell) + FVC(\ell^2 \log \ell))$	—
PriVDT [9]	ASS, FSS	$\mathcal{O}(dFCV + FV)$	$\mathcal{O}(NFCV\ell + FVC(12\ell^2 + d\ell))$	—
Swan	iASS	$\mathcal{O}(\log(FV) + \log F)$	$\mathcal{O}(NFVC\ell + 2FCV(\ell \log^2 \ell + \ell^2))$	—

These legal frameworks make it impossible to freely share sensitive information in plaintext across organizations for collaborative DT training. ASS, AHE, GC, FE, FSS, and iASS are Additive Secret Sharing, Homomorphic Encryption, Garbled Circuit, Secure Function Encryption, Function Secret Sharing, and improved Additive Secret Sharing, respectively; ℓ , ℓ_e , ℓ_s , ℓ_g are the ciphertext bits of Secret Sharing, HE, FE, and GC, respectively; N , F , V , C are the total number of sample instances, features, feature splits, and sample classes, respectively; κ_i is the degree of approximate polynomials for $x \ln(x)$ in [15]; ϵ is the degree approximate polynomials for e^x and $1/x$; ‘—’ represents no leakage of data privacy.

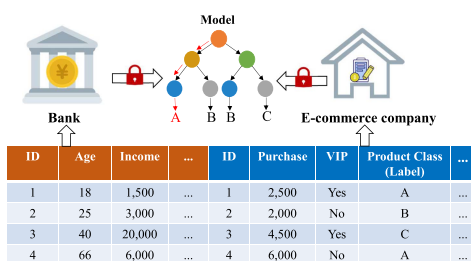


Fig. 1. An illustration of PVDT. In the framework, the independent variables are jointly provided by a bank and an e-commerce company. The bank contributes financial features, including ‘‘Age’’ and ‘‘Income history’’. The e-commerce company supplies online transactions, including ‘‘Purchase history’’ and ‘‘VIP membership status’’ (a binary indicator distinguishing privileged consumers with ‘‘Yes/No’’). These consumer features, which are distributed across two parties, serve as inputs for the model to predict a dependent variable—‘‘Product Class’’—indicating the type of product that the consumer is likely to purchase.

secret sharing (FSS) to construct secure comparison and division protocols, upon which an efficient PVDT scheme is derived. However, PriVDT heavily relies on a trusted third party (TTP) for FSS key generation, and requires many communication rounds similar to Pivot, hindering its deployment in practice.

The challenges in designing a practical PVDT stem from two factors: 1) The training and inference of PVDT consist of time-consuming non-linear functions (e.g., comparison and division) as well as a large number of linear functions (e.g., extensive N -input multiplications), requiring the design of more secure and efficient cryptographic protocols to enhance the scalability of these operations. 2) The state-of-the-art PVDT schemes typically rely on SS-based MPC, but the extensive communication rounds required by SS-based MPC become the primary performance bottleneck for these works. Reducing communication rounds are particularly important in practice, where communication round is often the dominant factor in distributed environments with high-latency such as cross-organizational collaborations or wide-area networks. Thus, these factors pose a rigorous challenge in designing a practical PVDT scheme that is both fast in computation and efficient in communication.

In this paper, we present *Swan*, a two-party PVDT framework for a secure, accurate, and fast realization of vertical DT. For

all components of PVDT, we introduce the improved ASS of ABY2.0 [16] to protect data as it can offer lower communication and simpler computations than the existing lightweight ASS [17]. *Swan* also uses an input-independent offline preprocessing [18] to boost performance for the training and inference of DT in the online phase. As shown in Table I, *Swan* achieves lower communication and computation costs and any sensitive information is not revealed except for the output. Our contributions can be summarized as follows:

- *Communication-efficient build blocks*: We propose a novel ASS-based two-party protocol for N -input multiplication that moves input-independent randomness generation to the offline phase, allowing for just one round of online communication. Building on this protocol, we propose two-party constant-round secure comparison and division protocols using 0/1 encoding techniques [19] and the Goldschmidt algorithm [20], respectively, achieving the state-of-the-art in round complexity. These protocols not only form the foundation of our scheme but also hold potential for other privacy-critical applications, which may be of independent interest.
- *Secure, accurate, and fast realization of vertical decision tree*: Based on the building blocks, we propose a lightweight PVDT framework (*Swan*) for vertical decision trees without revealing either party’s data or model. Our framework provides top-notch accuracy and remarkably improves computational and communication efficiency, which benefits from the use of lightweight cryptographic primitives and the incorporated partial parallel processing.
- *Comprehensive theoretical and experimental analysis*: We provide a formal security proof and a detailed complexity analysis for both the building blocks as well as the training and inference protocols of *Swan*. Moreover, we implement a prototype system and conduct extensive evaluations on real-world datasets under LAN and WAN settings. Experimental results demonstrate that *Swan* outperforms the state-of-the-art works in terms of runtime and communication costs while realizing comparable accuracy to the non-private setting. Especially in WAN settings, the training protocol of *Swan* is $109.3 \sim 188.3\times$ and $3.00 \sim$

10.2 \times faster than Pivot and PriVDT, respectively. For the inference protocol, *Swan* is 3.9 \sim 4.4 \times and 2.2 \sim 2.8 \times faster than them, respectively.

The rest of this paper is organized as follows. We introduce the related works and preliminaries in Sections II and III, respectively. Next, we give an overview of our system in Section IV. The building blocks and the details of *Swan* are presented in Sections V and VI, respectively. We give the theoretical analysis in Section VII. We evaluate the proposed schemes through extensive experiments in Section VIII. Finally, we conclude this paper in Section IX.

II. RELATED WORK

A. Private Learning for Vertical Decision Trees

Private learning for vertical decision trees has garnered significant attention, as shown in Table I, particularly in scenarios where data is vertically partitioned and distributed between two parties. Du and Zhan [7] presented the decision tree training algorithm for vertically partitioned data. However, this approach violates privacy regulations by publicizing sample labels and split information to all parties. Wang et al. [21] enhanced this method by incorporating secure intersection technology and garbled circuits. Nonetheless, the crucial concern of label leakage remains unaddressed. Recently, Chen et al. [11] proposed SecureBoost on privacy-preserving gradient boost decision tree for vertically partitioned data using homomorphic encryption. However, it has limited security due to the exposure of intermediate information during the training. Additionally, the heavy cryptographic operations lead to low training efficiency.

Lindal and Pinkas [12] first introduced a full-private decision tree training scheme utilizing oblivious transfer and garbled circuits. This scheme ensures “perfect” security, guaranteeing that no information is leaked to other parties. However, the computational cost of this approach is prohibitively high, rendering it impractical for practical implementation. Kikuchi et al. [13] proposed privacy-preserving decision tree learning protocol (PPDT) with additive homomorphic encryption. However, this scheme requires computation and communication overhead for a lot of heavy decryptions and encryptions in the training phase. Zheng et al. [14] implemented a framework (Privet) with the ASS technique. However, the framework requires frequent communication interactions when computing non-linear functions. Recently, there have also been some works focusing on vertical decision trees with mixed cryptographic primitives to reduce communication costs. Wu et al. [8] proposed a system (Pivot) using homomorphic encryption and ASS techniques. However, this requires time-consuming conversion of HE ciphertexts into ASS values. Chen et al. [9] built a scheme (PriVDT) by employing FSS and ASS techniques. Nonetheless, this approach relies on a trusted third party (TTP) for FSS key generation.

Different from the recent works, we design a series of secure and communication-efficient computation protocols specifically tailored to less-than comparison and division, which are specific to optimize the training and inference processes of PVDT with full privacy.

B. Secure Comparison and Secure Division Protocols

Secure comparison protocol is considered one of the most fundamental building blocks for PVDT. This significance dates back to Yao’s seminal paper [22], which introduced the “Two Millionaires’ Problem” for comparing integers employing the garbled circuits (GC). Over time, the protocol based on the garbled circuits underwent various enhancements [23], [24], [25]. Nevertheless, GC requires lots of computation overheads. In the protocol, the garbler encrypts each wire of each gate wires using symmetric key [26]. Next, the keys corresponding to the inputs of one user are retrieved by the evaluator by using an Oblivious Transfer protocol. The circuit is evaluated gate-by-gate because the encrypted circuit has been constructed such that the gates have been chained together. However, this process incurs a large computational overhead. In recent years, Liu et al. [27] optimized the comparison protocol based on full adder logic for extracting the most securely significant bit (MSB) of $x - y$ in the secret sharing domain. However, their protocol still requires $\log \ell$ rounds of communication. A significant breakthrough was achieved by Damgard [28] et al. who introduced the first comparison protocol based on secret-sharing that operates in constant rounds. Their protocol can be built upon any linear secret-sharing-based MPC framework that includes a multiplication protocol but requires 44 rounds of multiplication protocols. Inspired by this, we combine 0/1 encoding technology with SS-based MPC to propose a comparison protocol with two communication rounds.

Secure division protocol is the main hurdle for secure computation. Many researchers have explored secure protocols for division. These studies encompass homomorphic encryption-based protocols, garbled circuit-based protocols, and SS-based protocols. Veugen [29] introduces a secure integer division protocol that produces an approximate quotient using homomorphic encryption. Yet, it is only effective in situations where the divisor is either publicly disclosed or known to one of the parties participating in the computation. Bogdanov et al. [30] present a secure integer division protocol that provides an exact quotient based on SS-based protocols. However, their protocol requires a significant amount of communication rounds between the two parties. Payman Mohassel et al. [31] proposed a secure division protocol using garbled circuit-based protocols. However, the protocol requires a lot of communication and computation overhead. In contrast, we focus on a division protocol with constant communication rounds.

III. PRELIMINARIES

In this section, we review the decision tree, the cryptographic primitives of ABY2.0, and the 0/1-encoding technique involved in this paper. For the sake of readability, Table II summarizes the notations used in *Swan*.

A. Decision Trees

This paper focuses on the training and inference of binary-structured decision trees for classification tasks. A DT consists of internal nodes, edges, and leaf nodes: each internal node

TABLE II
NOTATION DESCRIPTIONS

Notations	Definitions
$\langle x \rangle := (\langle x \rangle_0, \langle x \rangle_1)$	2-out-of-2 additive secret sharing of x
$\langle x \rangle_\alpha$	$\langle x \rangle_\alpha$ is held by P_α for $\alpha \in \{0, 1\}$
$\langle x \rangle^B := (\langle x \rangle_0^B, \langle x \rangle_1^B)$	2-out-of-2 boolean secret sharing of x
$\llbracket x \rrbracket := (\llbracket x \rrbracket_0, \llbracket x \rrbracket_1)$	Improved additive secret sharing of x
$\llbracket x \rrbracket_\alpha := (m_x, \langle r_x \rangle_\alpha)$	$\llbracket x \rrbracket_\alpha$ is held by P_α for $\alpha \in \{0, 1\}$
$\llbracket x \rrbracket^B := (\llbracket x \rrbracket_0^B, \llbracket x \rrbracket_1^B)$	Improved boolean secret sharing of x
Set_0^x	0-encoding of x
Set_1^x	1-encoding of x

represents a test (e.g., $Age < 20$) between a feature and a threshold, each edge represents the outcome of the test, and each leaf node represents a predicted class label for the path taken through the tree. Assume that the trained DT is a complete binary tree T with a maximum depth H , as done in PriVDT and Pivot. Given a training dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i) | i \in [0, N-1]\} \in \mathbb{R}^{N \times (F+1)}$, which consists of N samples with F features and a label y_i . The training algorithm constructs DT nodes recursively in a top-down manner, starting from the root.

When a DT node is split, the training algorithm evaluates all potential splits across all features to find the *best-split* (i.e., a pair of the feature index and threshold), using a criterion like information gain or Gini impurity. In this work, we follow the chosen criterion of PriVDT and Pivot and use Gini impurity [32] to determine the best split, as detailed in Algorithm 1. Assume that $D \in \mathbb{Z}_{N-1}$ be the set of sample index for one tree node, where $D = [0, N-1]$ for the root node. Let $\mathcal{C} = [1, C]$ be a class set at one tree node. The Gini impurity of the current node is:

$$g(D) = 1 - \sum_{c \in \mathcal{C}} \left(\frac{|D_c|}{|D|} \right)^2 \quad (1)$$

where $|D_c|$ is the number of samples in D that belong to class c , and $|D|$ is the total number of samples in D . Assume that there are $V \times F$ potential splits $S = \{s_{j,k} | j \in [0, F-1], k \in [0, V-1]\}$, where $s_{j,k}$ is the k th split of the j th feature. For a given split $s_{j,k}$, D is divided into two subsets: $D_l = \{i | (x_{i,j} < s_{j,k}) \wedge (i \in D)\}$ and $D_r = D - D_l$, where $x_{i,j} \in \mathbf{x}_i$ denotes the j th feature of the i th sample. The Gini gain of split $s_{j,k}$ is:

$$g_{j,k} = g(D) - \left(\frac{|D_l|g(D_l)}{|D|} + \frac{|D_r|g(D_r)}{|D|} \right) \quad (2)$$

where $|D_l|$ and $|D_r|$ are the total number of samples in D_l and D_r , separately. To simplify g , we adopt PriVDT's approach as follows:

$$g_{j,k} = \sum_{c \in \mathcal{C}} \frac{|D_{l,c}|^2}{|D_l|} + \sum_{c \in \mathcal{C}} \frac{|D_{r,c}|^2}{|D_r|} \quad (3)$$

where $|D_{l,c}|$ and $|D_{r,c}|$ are the total number of samples labeled with class c in D_l and D_r , respectively. After computing all potential splits, the split with the maximum g is selected as the best feature split $s_{f_*}^*$ and recorded in the current node, where f_* is the corresponding feature index. Based on this best split, D is partitioned into two best subsets: D_l^* and D_r^* . The process is recursively applied to the left and right subsets. Once the

Algorithm 1: DT Training for One Tree Node.

Input: The sample indices at the current node

$D \in [0, N-1]$; The potential splits

$S = \{s_{j,k} | j \in [0, F-1], k \in [0, V-1]\}$; The training dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i) | i \in [0, N-1]\} \in \mathbb{R}^{N \times (F+1)}$; The current depth h .

Hyperparameter: Maximum depth H ; The number of samples N ; The number of features F ; The split number of each feature v ; The label set $\mathcal{C} = [1, C]$.

Output: a well-trained tree node

- 1: **if** $h = H$ **then**
- 2: $c^* = \arg \max_{c \in \mathcal{C}} (|\{i | (y_i = c) \wedge (i \in D)\}|)$
- 3: c^* is recoded into the current node.
- 4: **end if**
- 5: initialize an empty set $\mathbf{gain}_b = \{\}$
- 6: **for** $j \in [0, F-1]$ **do**
- 7: initialize an empty set $\mathbf{gain}_f = \{\}$
- 8: **for** $k \in [0, V-1]$ **do**
- 9: $D_l = \{i | (x_{i,j} < s_{j,k}) \wedge (i \in D)\}$, where $\mathbf{x}_i = \{x_{i,j}\} \in \mathcal{D}$ is the j th feature of the i th sample; $D_r = D - D_l$.
- 10: Compute the split gain of $s_{j,k}$:

$$g_{j,k} = \sum_{c \in \mathcal{C}} \frac{|D_{l,c}|^2}{|D_l|} + \sum_{c \in \mathcal{C}} \frac{|D_{r,c}|^2}{|D_r|}$$

where $D_{l,c} = \{i | (y_i = c) \wedge (i \in D_l)\}$;

$D_{r,c} = \{i | (y_i = c) \wedge (i \in D_r)\}$.

- 11: $\mathbf{gain}_f \leftarrow (g_{j,m}) \triangleright$ Put $g_{j,k}$ into \mathbf{gain}_f .
 - 12: **end for**
 - 13: $k_* = \arg \max_{k \in [0, V-1]} (\mathbf{gain}_f)$
 - 14: $\mathbf{gain}_b \leftarrow (g_{j,k_*}) \triangleright$ Put g_{j,k_*} into \mathbf{gain}_b .
 - 15: **end for**
 - 16: $f_* = \arg \max_{j \in [0, F-1]} (\mathbf{gain}_b)$
 - 17: $s_{f_*}^* = s_{f_*, k_*}$.
 - 18: $D_l^* = \{i | (x_{i,f_*} < s_{f_*}^*) \wedge (i \in D)\}$; $D_r^* = D - D_l^*$.
 - 18: **return** $(f^*, s_{f_*}^*)$ is recoded into the current node.
-

maximum tree depth is reached, the node becomes a leaf node, and the most frequent label in D is chosen as the predicted label:

$$c^* = \arg \max_{c \in \mathcal{C}} |\{i | (y_i = c) \wedge (i \in D)\}| \quad (4)$$

The label c^* is recorded as the prediction for the node.

B. Revisiting ABY2.0

Compared to existing frameworks such as ABY [17], CrypTen [33], SecureML [34], and MP-SPDZ [35], the ABY2.0 framework [16] supports not only generic additive secret sharing ($\langle \cdot \rangle$ -sharing) but also an improved additive secret sharing scheme ($\llbracket \cdot \rrbracket$ -sharing). The latter offers enhanced communication efficiency during the online phase. Now, we revisit the 2PC protocol in ABY2.0, which is executed on the ring \mathbb{Z}_{2^ℓ} between two parties, P_0 and P_1 .

1) $\langle \cdot \rangle$ -Sharing: The $\langle \cdot \rangle$ -sharing of an ℓ -bit secret value $x \in \mathbb{Z}_{2^\ell}$ is a 2-out-of-2 additive secret sharing (ASS) scheme in

two parties, i.e., P_0 holds $\langle x \rangle_0 = r$ and P_1 holds $\langle x \rangle_1 = x - r$, where r is a uniformly random value in \mathbb{Z}_{2^ℓ} . We use the shorthand notation $\langle x \rangle = (\langle x \rangle_0, \langle x \rangle_1)$ to represent the $\langle \cdot \rangle$ -shares of x . The fundamental protocols of ASS include:

- $x = \text{Rec}(\langle x \rangle)$: To reconstruct the secret value x , each party P_α ($\alpha \in \{0, 1\}$) transmits its share $\langle x \rangle_\alpha$ to $P_{1-\alpha}$. The recipient then computes $x = \langle x \rangle_0 + \langle x \rangle_1$.
- $\langle z \rangle = \langle x \rangle + \langle y \rangle$: Given $\langle x \rangle$ and $\langle y \rangle$, P_α computes $\langle z \rangle = \langle x + y \rangle$ by locally setting $\langle x + y \rangle_\alpha = \langle x \rangle_\alpha + \langle y \rangle_\alpha$.
- $\langle z \rangle = \langle x \rangle + c$: Given $\langle x \rangle$ and a public constant c , P_α computes $\langle z \rangle = \langle x + c \rangle$ by locally setting $\langle x + c \rangle_\alpha = \langle x \rangle_\alpha + \alpha \cdot c$.
- $\langle z \rangle = c \cdot \langle x \rangle$: Given $\langle x \rangle$ and a public constant c , P_α computes $\langle z \rangle = \langle cx \rangle$ by locally setting $\langle cx \rangle_\alpha = c \cdot \langle x \rangle_\alpha$.
- $\langle z \rangle = \langle x \rangle \cdot \langle y \rangle$: Given $\langle x \rangle$ and $\langle y \rangle$, P_0 and P_1 first reconstruct $e = \text{Rec}(\langle x \rangle - \langle a \rangle)$ and $f = \text{Rec}(\langle y \rangle - \langle b \rangle)$, where $\{\langle a \rangle, \langle b \rangle, \langle ab \rangle\}$ are precomputed and stored using Beaver multiplication triplet technique during the offline phase. Finally, P_α outputs $\langle z \rangle_\alpha = \alpha \cdot e \cdot f + e \cdot \langle b \rangle_\alpha + f \cdot \langle a \rangle_\alpha + \langle ab \rangle_\alpha$. Note that if x and y are floating point numbers, they are encoded into fixed-point integers with ℓ_f -precision bits. To ensure $z = x \cdot y$ with a lower loss, we use a secure truncation proposed by SecureML [34] to correct the result locally.

Boolean sharing is an XOR-based variant of ASS over \mathbb{Z}_2 , where the addition (+) operation and multiplication operation (\cdot) are replaced with the XOR (\oplus) operation and AND (\wedge) operation, respectively. To simplify the presentation, Boolean sharing of a bit $x \in \mathbb{Z}_2$ is abbreviated as $\langle x \rangle^B = (\langle x \rangle_0^B, \langle x \rangle_1^B)$.

A2B. The A2B protocol inputs $x \in \mathbb{Z}_{2^\ell}$ and outputs ℓ -bit Boolean shares $\langle x_{(\ell-1)} \rangle^B \dots \langle x_{(0)} \rangle^B \in \{0, 1\}^\ell$, i.e., $\langle x_{(\ell-1)} \rangle^B \dots \langle x_{(0)} \rangle^B = \text{A2B}(\langle x \rangle)$, where $x = \langle x \rangle_0 + \langle x \rangle_1 = \sum_{j=0}^{\ell-1} (\langle x_{(j)} \rangle_0^B \oplus \langle x_{(j)} \rangle_1^B) 2^j$. Here, we use a constant-round solution in ABY framework by composing the protocols Y2B (Yao sharing to $\langle \cdot \rangle^B$ -sharing) and A2Y ($\langle \cdot \rangle^B$ -sharing to Yao sharing). That is, $\langle x \rangle^B = \text{Y2B}(\text{A2Y}(\langle x \rangle))$.

2) $\llbracket \cdot \rrbracket$ -Sharing: The $\llbracket \cdot \rrbracket$ -sharing is an improved additive secret sharing [16] (iASS). P_α for $\alpha \in \{0, 1\}$ holds $\llbracket x \rrbracket_\alpha = (m_x, \langle r_x \rangle_\alpha)$, where $r_x \in \mathbb{Z}_{2^\ell}$ is a random values and $m_x = x - r_x \in \mathbb{Z}_{2^\ell}$ such that m_x is publicly known to P_0 and P_1 , and r_x is $\langle \cdot \rangle$ -shared between the two parties. We use the shorthand notation $\llbracket x \rrbracket = (\llbracket x \rrbracket_0, \llbracket x \rrbracket_1)$ to represent the $\llbracket \cdot \rrbracket$ -shares of x . The basic operations in iASS are as follows:

- $x = \text{Rec}(\llbracket x \rrbracket)$: To reconstruct the secret x , P_0 and P_1 mutually exchange their missing $\langle \cdot \rangle$ -share of r , and compute $x = m_x + \text{Rec}(\langle r_x \rangle)$.
- $\llbracket z \rrbracket = \llbracket x \rrbracket + \llbracket y \rrbracket$: Given $\llbracket x \rrbracket$ and $\llbracket y \rrbracket$, P_α ($\alpha \in \{0, 1\}$) compute $\llbracket z \rrbracket = \llbracket x + y \rrbracket$ by locally setting $(m_z = m_x + m_y, \langle r_z \rangle = \langle r_x \rangle_\alpha + \langle r_y \rangle_\alpha)$.
- $\llbracket z \rrbracket = \llbracket x \rrbracket + c$: Given $\llbracket x \rrbracket$ and a public constant c , P_α computes $\llbracket z \rrbracket = \llbracket x + c \rrbracket$ by locally setting $(m_z = m_x + c, \langle r_z \rangle = \langle r_x \rangle_\alpha + \langle r_y \rangle_\alpha)$.
- $\llbracket z \rrbracket = \llbracket x \rrbracket \cdot \llbracket y \rrbracket$: Given $\llbracket x \rrbracket$ and $\llbracket y \rrbracket$, P_α locally generates a random value as $\langle r_z \rangle_\alpha \in \mathbb{Z}_{2^\ell}$ in the offline phase. $\langle r_x r_y \rangle$ is precomputed and stored using multiplication of $\langle \cdot \rangle$ -sharing in the offline phase. In the online phase,

P_α locally computes $\langle z \rangle_\alpha = m_x \cdot \langle r_y \rangle_\alpha + m_y \cdot \langle r_x \rangle_\alpha + \langle r_x r_y \rangle_\alpha - \langle r_z \rangle_\alpha$ and sends it to $P_{1-\alpha}$. P_0 and P_1 locally set $m_z = m_x \cdot m_y + \text{Rec}(\langle z \rangle)$. Finally, P_α outputs $\llbracket z \rrbracket = \llbracket xy \rrbracket_\alpha := (m_z, \langle r_z \rangle_\alpha)$.

- $\llbracket z \rrbracket = c \cdot \llbracket x \rrbracket$: Given $\llbracket x \rrbracket$ and a public constant c , P_α computes $\llbracket z \rrbracket = \llbracket cx \rrbracket$ by locally setting $(m_z = c \cdot m_x, \langle r_z \rangle = c \cdot \langle r_x \rangle_\alpha)$.
- **3MUL protocol.** ABY2.0 implements a 3-input multiplication protocol, i.e., $\llbracket z \rrbracket = \text{3MUL}(\llbracket x_0 \rrbracket, \llbracket x_1 \rrbracket, \llbracket x_2 \rrbracket)$, where $z = x_0 \cdot x_1 \cdot x_2$.

Improved Boolean sharing is an XOR-based variant of $\llbracket \cdot \rrbracket$ -sharing over \mathbb{Z}_2 , where the addition operation (+) and the multiplication operation (\cdot) are replaced with bitwise operations XOR (\oplus) and AND (\wedge), respectively. To simplify the presentation, an improved Boolean sharing of a bit $x \in \mathbb{Z}_2$ is abbreviated as $\llbracket x \rrbracket^B = (\llbracket x \rrbracket_0^B, \llbracket x \rrbracket_1^B)$.

B2A: The B2A protocol inputs $\llbracket x \rrbracket^B \in \mathbb{Z}_2$ and outputs $\llbracket x \rrbracket \in \mathbb{Z}_{2^\ell}$, i.e., $\llbracket y \rrbracket = \text{B2A}(\llbracket x \rrbracket^B)$, where $\llbracket y \rrbracket_0 + \llbracket y \rrbracket_1 = \llbracket x \rrbracket_0^B \oplus \llbracket x \rrbracket_1^B$. In the ABY2.0 framework, P_0 and P_1 employ an oblivious transfer protocol to convert $\langle r_x \rangle^B$ into $\langle r_x \rangle$. Let $\langle r_y \rangle = \langle r_x \rangle$. In the online phase, each party P_α for $\alpha \in \{0, 1\}$ locally computes $\langle y \rangle_i = \alpha m_x + (1 - 2m_x) \cdot \langle r_y \rangle_\alpha$, where $m_x = x \oplus r_x$. After that, two parties jointly compute $m_y = \text{Rec}(\langle x \rangle - \langle r_y \rangle)$. As a result, each party P_α locally has $\llbracket y \rrbracket_\alpha = (m_y, \langle r_y \rangle)$.

C. 0-Encoding and 1-Encoding

Our work uses the 0/1-encoding [19] to solve the comparison problem of two values. This method can cleverly transform a comparison problem into an equality test problem. Let an ℓ -length binary string of x be $x_{(\ell-1)} \dots x_{(0)} \in \{0, 1\}^\ell$. The 0-encoding of x is defined as a set Set_0^x such that:

$$\text{Set}_0^x = \{x_{(\ell-1)}x_{(\ell-2)} \dots x_{(i+1)}1 \mid x_{(i)} = 0, 0 \leq i \leq \ell - 1\} \quad (5)$$

and the 1-encoding of x is defined as a set Set_1^x such that:

$$\text{Set}_1^x = \{x_{(\ell-1)}x_{(\ell-2)} \dots x_{(i)} \mid x_{(i)} = 1, 0 \leq i \leq \ell - 1\} \quad (6)$$

To compare two integers x and y , we encode x and y into Set_1^x and Set_0^y , respectively. If $|\text{Set}_1^x \cap \text{Set}_0^y| = 1$, then $x > y$, where $|\cdot|$ is the size of a set. For example, let $x = 10 = 1010_2$ and $y = 3 = 0011_2$ of length $\ell = 4$. Since $\text{Set}_1^x = \{1, 101\}$ and $\text{Set}_0^y = \{1, 01\}$, we have $|\text{Set}_1^x \cap \text{Set}_0^y| = 1 \rightarrow x > y$.

IV. SYSTEM OVERVIEW

A. System Model

Swan is a secure two-party computation framework for the training and inference of PVDT. In particular, each party P_α owns a vertical dataset \mathcal{D}_α for $\alpha \in \{0, 1\}$. Let $\mathcal{D} = \{\mathcal{D}_0, \mathcal{D}_1\}$ be a joint dataset. The features between \mathcal{D} are vertically distributed between the two parties, i.e., \mathcal{D}_0 and \mathcal{D}_1 have the same sample IDs but contain different features. We assume P_0 's features come before P_1 's features, and the two datasets have been aligned using the common private set intersection technique, effectively creating a combined dataset $\mathcal{D} = \mathcal{D}_0 \parallel \mathcal{D}_1$. For clarity, we designate P_1 as the holder of the whole label set Y . Swan also uses an

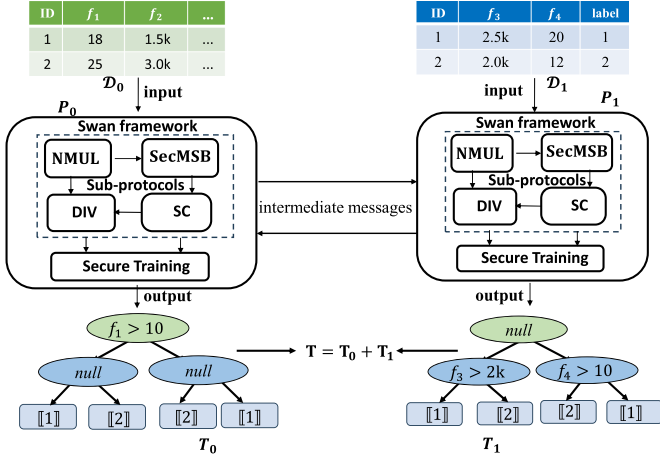


Fig. 2. Swan frameworks for the training of PVDT.

input-independent offline preprocessing to boost performance for the training and inference of DT in the online phase. In all the algorithms of Swan, all of operations (on offline and online) should be deployed on both parties, where the offline operations can be performed in parallel before online computation.

Traditional DT training (shown in Algorithm 1) mainly involves two processes: calculating the splitting criterion and partitioning \mathcal{D} based on the assigned node, which involves many linear and non-linear operations. To reduce the overhead of PVDT in MPC, we design a secure and parallel protocol for N -input multiplication with one communication round. This forms the cornerstone for a series of secure and communication-efficient computation protocols. Based on them, we use these optimized protocols to refine the training and inference processes of PVDT, reducing communication costs.

During the training, the best feature and threshold assigned to a node are revealed to the party who holds the corresponding feature. Each party P_α can partition \mathcal{D}_α locally and then share it securely with $P_{1-\alpha}$ for training the next node. After training, as shown in Fig. 2, each party P_α gets a partial of a DT model, i.e., T_α . The private model T_α only contains the tree part involving P_α 's feature split and all the leaf labels' secret share. T_0 and T_1 compose a full tree model T . During the inference phase, given a well-trained private tree T_α , P_0 and P_1 collaboratively execute the privacy-preserving inference protocol on a sample query whose features are distributed across two parties. Such protocol yields the secret-shared inference result without revealing any private information about the well-trained tree and the query.

B. Threat Model and Security Goals

Swan adopts the semi-honest adversary model [16], which has already been widely used in related work such as PrivDT, and Pivot. In this model, P_0 and P_1 will follow the protocol as specified but try to learn additional information by analyzing the exchanged messages during the protocol's execution. In addition, in line with prior works [16], [36], [37], we assume that P_0 and P_1 do not collude to break the protocols. We aim to design secure training and inference protocols for Vertical

DT under the semi-honest adversary model, such that the two parties, P_0 and P_1 , learn nothing about each other's private dataset. We utilize the following standard simulation paradigm for semi-honest security [38].

Definition 1: Let a two-party protocol Π for computing a functionality $f(x, y) = (f_0(x, y), f_1(x, y))$, where $f_0(x, y)$ with input x and $f_1(x, y)$ with input y are computed by P_0 and P_1 , respectively. The view of P_0 (resp. P_1) during an execution of Π on (x, y) is denoted by $view_0^\Pi(x, y) = (x, r_0, m_1, \dots, m_t)$ (resp. $view_1^\Pi(x, y) = (y, r_1, m_1, \dots, m_t)$), where r_0 (resp. r_1) represents the internal randomness of P_0 (resp. P_1) and m_i represents the i th message passed between the parties. The output of P_0 (resp. P_1) during an execution of Π on (x, y) is denoted by \mathcal{O}_0^Π (\mathcal{O}_1^Π). Let the joint output of two parties be $\mathcal{O}^\Pi = (\mathcal{O}_0^\Pi, \mathcal{O}_1^\Pi)$. We say that Π privately computes $f(x, y)$ if there exist probabilistic polynomial-time (PPT) simulators \mathcal{S}_0 and \mathcal{S}_1 such that

$$(\mathcal{S}_0(x, f_0(x, y)), f(x, y)) \stackrel{c}{\approx} (view_0^\Pi(x, y), \mathcal{O}^\Pi(x, y)) \quad (7)$$

$$(\mathcal{S}_1(y, f_1(x, y)), f(x, y)) \stackrel{c}{\approx} (view_1^\Pi(x, y), \mathcal{O}^\Pi(x, y)) \quad (8)$$

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

Clearly, the security goals essentially require that the proposed secure training and inference protocols between P_0 and P_1 should satisfy the above security definition.

V. BUILDING BLOCKS

A. Secure N -Input Multiplication Protocol

We first introduce a secure N -input multiplication protocol, denoted by $[z] = \text{NMUL}([x_0], [x_1], \dots, [x_{n-1}])$, where $z = x_0 \cdot x_1 \cdot \dots \cdot x_{n-1}$ and $x_j \in \mathbb{Z}_{2^\ell} \setminus \{0\}$ for $0 \leq j \leq n-1$. A naive solution to the problem is to invoke $n-1$ times multiplication protocols, which costs $n-1$ communication rounds. To reduce the online communication complexity, ABY2.0 proposes a secure N -input multiplication protocol. However, the improved performance of their protocol comes at the cost of an expensive pre-processing, and hence to maintain balance, N is usually set to 3 or 4 in the practical applications. To support any size of N , we adopt a **secure chained multiplication computation strategy**, i.e., $z = x_0 \cdot x_1 \cdot \dots \cdot x_{n-1} = x_0 g^{a_0 - a_1} \cdot x_1 g^{a_1 - a_2} \cdot \dots \cdot x_{n-1} g^{a_{n-1} - a_n} \cdot g^{a_n - a_0}$. In offline phase, we pre-compute all random masks $\{g^{a_i - a_{i+1}} \mid i \in [0, n-1]\}$ and $g^{a_n - a_0}$ to use to protect N private inputs. On the online phase, we compute the N -input multiplication computation chain $z = x_0 g^{a_0 - a_1} \cdot x_1 g^{a_1 - a_2} \cdot \dots \cdot x_{n-1} g^{a_{n-1} - a_n} \cdot g^{a_n - a_0}$. Compared with the existing works, our protocol only requires 1 communication round and $(n-1)\ell$ bits of communication overhead (cf. Table VI of Appendix C, available online).

The Algorithm 2 gives specific steps of NMUL protocol. The correctness of our protocol can be demonstrated by checking that the reconstructed result $z = x_0 \cdot x_1 \cdot \dots \cdot x_{n-1}$. As deduced from the protocol, $d_j = x_j b_j = x_j \cdot \text{Rec}(\text{M2A}(g^{(a_j - a_{j+1})_0}, g^{(a_j - a_{j+1})_0})) = x_j g^{a_j - a_{j+1}}$ and $p = g^{a_n - a_0} - r$, then $z = [z]_0 + [z]_1 = p \cdot d_0 \cdot d_1 \cdot \dots \cdot d_{n-1} + r \cdot d_0 \cdot d_1 \cdot \dots \cdot d_{n-1} = x_0 g^{a_0 - a_1} x_1 g^{a_1 - a_2} \cdot \dots \cdot x_{n-1} g^{a_{n-1} - a_n}$

Algorithm 2: Secure N -Input Multiplication Protocol: NMUL($\llbracket x_0 \rrbracket, \llbracket x_1 \rrbracket, \dots, \llbracket x_{n-1} \rrbracket$).

Input: Party P_α owns $\llbracket x_0 \rrbracket_i, \dots, \llbracket x_{n-1} \rrbracket_\alpha$ and a shared PRF key k .

Output: Party P_α outputs $\llbracket \prod_{j=0}^{n-1} x_j \rrbracket_\alpha$

In the offline phase:

- 1: P_α generates a random value as $\langle r \rangle_\alpha \in \mathbb{Z}_{2^\ell}$.
- 2: $P_0 \& P_1$ compute $v_k = \text{PRF}(k)$ and $g = 2v_k + 1 \in \mathbb{Z}_{2^\ell}$, where $v_k \neq 0$.
- 3: P_α locally generates $n + 1$ random values as shares $\{\langle a_j \rangle_\alpha \in \mathbb{Z}_{2^\ell} \mid 0 \leq j \leq n\}$ of $\{a_j \in \mathbb{Z}_{2^\ell} \mid 0 \leq j \leq n\}$.
- 4: P_α computes $\{\langle h_j \rangle_\alpha = g^{\langle a_j - a_\beta \rangle_\alpha} \mid 0 \leq j \leq n, \beta = j + 1 \text{ mod } (n + 1)\}$.
- 5: $P_0 \& P_1$ compute $\{\langle b_j \rangle = \text{M2A}(\langle h_j \rangle_0, \langle h_j \rangle_1) \mid 0 \leq j \leq n\}$.
- 6: $P_0 \& P_1$ compute $p = \text{Rec}(\langle b_n \rangle - \langle r \rangle) = g^{a_n - a_0} - r$.
- 7: $P_0 \& P_1$ compute $\{\langle q_j \rangle = \langle b_j r_{x_j} \rangle = \langle b_j \rangle \cdot \langle r_{x_j} \rangle \mid 0 \leq j \leq n - 1\}$.

In the online phase:

- 8: **for** $j \in [0, n - 1]$ **in parallel do**
 - 9: $P_0 \& P_1$ compute $\langle d_j \rangle = m_{x_j} \cdot \langle b_j \rangle + \langle q_j \rangle$ and reconstruct $d_j = \text{Rec}(\langle d_j \rangle) = x_j g^{a_j - a_\beta}$, where $\beta = (j + 1) \text{ mod } (n + 1)$.
 - 10: **end for**
 - 11: P_α computes $z = d_1 \dots d_{n-1} p$ and $\langle r_z \rangle_\alpha = d_1 \dots d_{n-1} \langle r \rangle_\alpha$.
 - 12: P_α outputs $\llbracket \prod_{j=0}^{n-1} x_j \rrbracket_\alpha := (z, \langle r_z \rangle_\alpha)$.
-

$g^{a_n - a_0} = x_0 \cdot x_1 \cdot \dots \cdot x_{n-1} g^0 = x_0 \cdot x_1 \cdot \dots \cdot x_{n-1}$, where $g = 2v_k + 1$, $v_k \neq 0$ and the M2A¹ protocol is a conversion of multiplicative share to additive share. It is worth emphasizing that g must be odd² in the ring \mathbb{Z}_{2^ℓ} and $g \neq 1$ to ensure data privacy. In the case of $g = 1$, we have $d_j = x_j$ such that x_j will be leaked.

B. Secure Comparison Protocol

The secure comparison protocol (denoted as **SC**) takes $\llbracket a \rrbracket$ and $\llbracket b \rrbracket$ as inputs and outputs $\llbracket \theta \rrbracket$ where $\theta = 0$ if $a \geq b$ and $\theta = 1$ otherwise. To reduce communication rounds in the existing works, we combine 0/1 encoding technology with SS-based MPC to propose a comparison protocol with two communication rounds. As the negative values in our scheme are represented by two's complement, similar to prior works [16], [40], the comparison operation can be transformed into the computation of the

¹M2A protocol [39] converts a pair of multiplicative shares, x_α held by P_α ($\alpha \in \{0, 1\}$) into an additive share pair $\langle y \rangle$, i.e., $\langle y \rangle_0 + \langle y \rangle_1 = x_0 \cdot x_1$. This operation can be concisely represented as $\langle y \rangle = \text{M2A}(x_0, x_1)$. In the protocol, P_0 sets $\langle u \rangle_0 = x_0$ and $\langle v \rangle_0 = 0$, while P_1 sets $\langle u \rangle_1 = 0$ and $\langle v \rangle_1 = x_1$. The output is computed as $\langle y \rangle = \text{M2A}(x_0, x_1) = \langle u \rangle \cdot \langle v \rangle$.

²For the NMUL protocol to be secure and correct, we must ensure that $\forall g^{a_j - a_{j+1}}$ is uniformly random in \mathbb{Z}_{2^ℓ} and has a corresponding multiplicative inverse. When g is odd and $a_j - a_{j+1}$ is uniformly random in \mathbb{Z}_{2^ℓ} , it follows that $\forall g^{a_j - a_{j+1}}$ is also uniformly random in \mathbb{Z}_{2^ℓ} with its corresponding multiplicative inverse given by $g^{a_{j+1} - a_j}$. However, if g is even, there exists some $g^{a_j - a_{j+1}} = 0$ in \mathbb{Z}_{2^ℓ} , rendering the NMUL protocol incorrect and insecure.

Algorithm 3: Secure MSB Protocol: SecMSB($\llbracket x \rrbracket$).

Input: Party P_α holds $\llbracket x \rrbracket_\alpha = (m_x, \langle r \rangle_\alpha)$.

Output: Party P_α outputs $\llbracket \text{MSB}(x) \rrbracket_\alpha^B$.

In offline phase:

// **Conversion from $\langle r \rangle$ to $\llbracket r_{(\ell-1)} \rrbracket^B, \dots, \llbracket r_{(0)} \rrbracket^B$**

- 1: $P_0 \& P_1$ compute $\langle r_{(\ell-1)} \rangle^B, \dots, \langle r_{(0)} \rangle^B = \text{A2B}(\langle r \rangle)$.
- 2: P_α locally choose ℓ -bit values $\langle q_{(\ell-1)} \rangle_\alpha^B, \dots, \langle q_{(0)} \rangle_\alpha^B \in \{0, 1\}^\ell$ at random, where $\{q_{(j)} = \langle q_{(j)} \rangle_0^B \oplus \langle q_{(j)} \rangle_1^B \mid 0 \leq j \leq \ell - 1\}$.
- 3: $P_0 \& P_1$ compute $\{r_{(j)} \oplus q_{(j)} = \text{Rec}(\langle r_{(j)} \oplus q_{(j)} \rangle^B) \mid 0 \leq j \leq \ell - 1\}$ to construct $\{\llbracket r_{(j)} \rrbracket^B = (r_{(j)} \oplus q_{(j)}, \langle q_{(j)} \rangle^B) \mid 0 \leq j \leq \ell - 1\}$.
- 4: $P_0 \& P_1$ set $\llbracket \text{MSB}(r) \rrbracket^B = \llbracket r_{(\ell-1)} \rrbracket^B$.
- 5: P_i locally computes $\llbracket S_1^x \rrbracket_\alpha = \{\llbracket r_{(\ell-2)} \rrbracket^B, \dots, \llbracket r_{(j)} \rrbracket_\alpha^B \mid 0 \leq j \leq \ell - 2\}$ for Boolean shares of $2r$.
- 6: P_i locally encodes 1-encoding of $2r$ $\text{Set}_1^{2r} = \{j + 1 \mid \exists \llbracket u_j \rrbracket^B \text{ s.t. } \llbracket u_j \rrbracket^B \in \llbracket S_1^x \rrbracket_\alpha\}$, where $\llbracket r_{(\ell-2)} \rrbracket^B, \dots, \llbracket r_{(j)} \rrbracket^B \leftarrow \llbracket u_j \rrbracket^B$.

In online phase:

// **Obtain the 0-encoding of the public value**

$$2^\ell - 1 - 2m_x$$

- 7: P_α set locally $y = 2^\ell - 1 - 2m_x$.
 - 8: P_α locally encode y into $S_0^y = \{y_{(\ell-2)}, \dots, y_{(j+1)}, 1 \mid y_{(j)} = 0, 1 \leq j \leq \ell - 2\}$ with 0-encoding rule.
 - 9: P_α locally set $\text{Set}_0^y = \{j \mid \exists v_j \text{ s.t. } v_j \in S_0^y\}$ where $y_{(\ell-1)}, \dots, y_{(j+1)}, 1 \leftarrow v_j$.
 - 10: $P_0 \& P_1$ locally compute $\llbracket \bar{S} \rrbracket = \{\llbracket c_\omega \rrbracket^B = \llbracket u_\omega \rrbracket^B \oplus v_\omega \mid \llbracket u_\omega \rrbracket^B \in \llbracket S_1^x \rrbracket, v_\omega \in S_0^y \text{ s.t. } \omega \in \text{Set}_0^y \cap \text{Set}_1^{2r}\}$, where $P_0 \& P_1$ parse $\llbracket c_{(\ell-1)} \rrbracket^B, \dots, \llbracket c_{(\omega)} \rrbracket^B \leftarrow \llbracket c_\omega \rrbracket^B$.
 - 11: $P_0 \& P_1$ locally encode $\llbracket \hat{S} \rrbracket = \{\llbracket \hat{c}_{(\ell-1)} \rrbracket = \llbracket c_{(\ell-1)} \rrbracket^B \oplus 11_2, \dots, \llbracket \hat{c}_{(\omega)} \rrbracket = \llbracket c_{(\omega)} \rrbracket^B \oplus 11_2 \mid \llbracket c_{(\ell-1)} \rrbracket^B, \dots, \llbracket c_{(\omega)} \rrbracket^B \in \llbracket \bar{S} \rrbracket \text{ s.t. } \omega \in \text{Set}_0^y \cap \text{Set}_1^{2r}\}$, where 11_2 is a binary number to void $\hat{c}_{(\ell-1)} = 0$ as the inputs of NMUL protocol can not be 0.
 - 12: $P_0 \& P_1$ compute $\{\llbracket \hat{c}_{(\omega)} \rrbracket = \text{NMUL}(\llbracket \hat{c}_{(\ell-1)} \rrbracket, \dots, \llbracket \hat{c}_{(\omega)} \rrbracket) \mid \llbracket \hat{c}_{(\ell-1)} \rrbracket, \dots, \llbracket \hat{c}_{(\omega)} \rrbracket \in \llbracket \hat{S} \rrbracket \text{ s.t. } \omega \in \text{Set}_0^y \cap \text{Set}_1^{2r}\}$.
 - 13: $P_0 \& P_1$ compute $\llbracket \text{wrap} \rrbracket^B = \bigoplus_\omega^{\text{Set}_0^y \cap \text{Set}_1^{2r}} \text{LSB}(\llbracket \hat{c}_{(\omega)} \rrbracket)$, where $\text{LSB}(\llbracket \hat{c}_{(\omega)} \rrbracket) = (m_{\hat{c}_{(\omega)}} \wedge 1, \langle r_{\hat{c}_{(\omega)}} \rangle \wedge 1)$.
 - 14: $P_0 \& P_1$ compute $\llbracket \text{MSB}(x) \rrbracket^B = \text{MSB}(m_x) \oplus \llbracket \text{MSB}(r) \rrbracket^B \oplus \llbracket \text{wrap} \rrbracket^B$.
-

most significant (MSB). That is, let $x = a - b$, $\text{MSB}(x) = 0$ if $a - b \geq 0$ and $\text{MSB}(x) = 1$ otherwise. Since $\llbracket x \rrbracket = (m_x, \langle r \rangle)$ is shares of x , i.e., $x = (x - r) + r = m_x + r \text{ mod } 2^\ell$, we have:

$$\text{MSB}(x) = \text{MSB}(m_x) \oplus \text{MSB}(r) \oplus \text{Wrap}(m_x, r) \quad (9)$$

Algorithm 4: Secure Division Protocol $\text{DIV}(\llbracket u \rrbracket, \llbracket x \rrbracket)$.**Input:** Party P_α owns $\llbracket u \rrbracket_\alpha, \llbracket x \rrbracket_\alpha$.**Output:** Party P_α outputs $\llbracket \frac{u}{x} \rrbracket_\alpha$

In offline phase:

// Each party generates a local lookup table LUT.

1: Initialize the 0th element $\text{LUT}[0] = 2^{\ell_f - 1}$ in the lookup table LUT.2: **for** $j \in [1, \ell - 1]$ **do**3: $\text{LUT}[j] = 2^{\ell_f - 1 - j} - 2^{\ell_f - j}$.4: **end for**

In online phase:

// Determine the scaling factor $2^{\ell_f - k}$ of x 5: **for** $j \in [0, \ell - 1]$ **in parallel do**6: $\llbracket k_j \rrbracket = \text{SC}(2^j - \llbracket x \rrbracket)$.7: **end for**8: $\llbracket 2^{\ell_f - k} \rrbracket = \sum_{j=1}^{\ell-1} \llbracket k_j \rrbracket \cdot \text{LUT}[j]$.// Scale x into the range $(2^{\ell_f - 1}, 2^{\ell_f}]$ 9: P_0 & P_1 compute $\llbracket b \rrbracket = \llbracket x \rrbracket \cdot \llbracket 2^{\ell_f - k} \rrbracket$.// Compute $u \cdot \text{AppDIV}(x) \approx u/x$.10: P_0 & P_1 compute $\llbracket w \rrbracket = 2.9142 - 2\llbracket b \rrbracket$.11: P_0 & P_1 compute $\llbracket \epsilon_0 \rrbracket = 1 - (\llbracket b \rrbracket \cdot \llbracket w_0 \rrbracket)$, $\llbracket \epsilon_1 \rrbracket = \llbracket \epsilon_0 \rrbracket \cdot \llbracket \epsilon_0 \rrbracket$.12: P_0 & P_1 compute $\llbracket \frac{u}{x} \rrbracket = \text{NMUL}(\llbracket u \rrbracket, \llbracket w_0 \rrbracket, 1 + \llbracket \epsilon_0 \rrbracket, 1 + \llbracket \epsilon_1 \rrbracket)$.

where $\text{Wrap}(m_x, r)$ determines whether $2m_x + 2r \geq 2^\ell$ [41], which is defined as:

$$\text{Wrap}(m_x, r) = \begin{cases} 1 & (2m_x + 2r) \geq 2^\ell \\ 0 & \text{Otherwise} \end{cases} \quad (10)$$

where $(2m_x + 2r) \geq 2^\ell \Leftrightarrow 2r > 2^\ell - 1 - 2m_x$. Therefore, we propose a secure MSB (SecMSB) protocol based on (9) and (10). The SecMSB($\llbracket x \rrbracket$) protocol takes as input $\llbracket \cdot \rrbracket$ -shares of x and outputs $\llbracket \cdot \rrbracket^B$ -shares of MSB(x).

In consideration of performance, the SecMSB protocol is divided into offline and online phases. Due to the fact that $\langle r \rangle$ is shared between P_0 and P_1 and m_x is a public value, $\llbracket \text{MSB}(r) \rrbracket^B$ can be offloaded to the offline phase, and MSB(m_x) can be computed locally on P_0 and P_1 . Then we focus on how to implement $\text{Wrap}(m_x, r)$ privately. With our key insights in mind, we use the **0/1 encoding technique** to determine whether $2r > 2^\ell - 1 - 2m_x$ in a secure and parallel manner. Algorithm 3 shows the detailed steps of SecMSB protocol. To implement the SC protocol, we have $\llbracket \theta \rrbracket = \text{SC}(\llbracket a \rrbracket - \llbracket b \rrbracket) = \text{B2A}(\text{SecMSB}(\llbracket a \rrbracket - \llbracket b \rrbracket))$, where B2A protocol (as shown in Section III-B) is to convert boolean shares into additive shares.

C. Secure Division Protocol

In DT trianing, the division is an important operation. The secure division protocol in existing works requires a lot of communication and computation overhead. In contrast, we focus on a division protocol with constant communication

rounds. In this work, we propose an efficient division protocol $\text{DIV}(\llbracket u \rrbracket, \llbracket x \rrbracket) = \llbracket u \rrbracket / \llbracket x \rrbracket$ based on the Goldschmidt algorithm [20], where $u \in \mathbb{Z}_{2^\ell}$ and $x \neq 0$. Goldschmidt algorithm is an iterative method that leverages multiplicative corrections to achieve high precision efficiently. That makes sense because multiplication can be faster than division in MPC. To apply the Goldschmidt algorithm, the divisor must be normalized to an initial value in $(0.5, 1]$. This normalization ensures that the iterative process converges efficiently. In MPC, x is represented as an ℓ -bit fixed-point integer, where the last ℓ_f bits denote the fractional part. The integer x has to be constrained in a fixed range, i.e., $2^{k-1} < x \leq 2^k$, where $k \in [0, \ell - 1]$. To satisfy the input requirements, we scale x by multiplying a scaling factor $2^{\ell_f - k}$, resulting in $b = x \cdot 2^{\ell_f - k} \in (2^{\ell_f - 1}, 2^{\ell_f}]$ in MPC.³ After that, we can have an initial approximation for $1/x$, as follows:

$$\frac{1}{x} \approx w_0 \cdot (1 + \epsilon_0) \quad (11)$$

where $\epsilon_0 = 1 - b \cdot w_0$ and $w_0 = 2.9142 - b$ (for choice of constants, cf. [42], [43]). For higher-order approximations, we can compute $\epsilon_i = \epsilon_{i-1}$ and multiply the previous approximate result by $(1 + \epsilon_i)$ to get a better approximation for $1/x$. Each successive iteration increases the computational overhead by 2 in MPC. As done in the existing works [43], we trade off accuracy and computational cost to construct the following approximation with an efficient performance.

$$\frac{1}{x} \approx \text{AppDIV}(x) = w_0 \cdot (1 + \epsilon_0)(1 + \epsilon_1) \quad (12)$$

where its fanal error is 5.4×10^{-3} on the ring $\mathbb{Z}_{2^{64}}$.

Algorithm 4 outlines the details of our DIV protocol. To determine the scaling factor $2^{\ell_f - k}$, we take an optimization trick by introducing a lookup table LUT. In the trick, each party P_α locally generates the lookup table LUT in offline phase, where $\text{LUT}[0] = 2^{\ell_f}$ and $\text{LUT}[j] = 2^{\ell_f - j} - 2^{\ell_f - (j-1)}$ for $j \in [1, \ell - 1]$. During the online phase, we invoke the secure SC protocol, secure scalar multiplication, and secure addition to retrieve the private result $\llbracket 2^{\ell_f - k} \rrbracket$ for the scaling factor, as shown in Lines 5-8. Next, we scale x into b by using secure multiplication, as shown in Lines 9. After that, we apply the secureNMUL protocol and secure multiplication to compute the approximation $u \cdot \text{AppDIV}(x)$ of u/x .

Here, we give a toy example to explain the process. Assume that $u = 000100_2 = 4$ (the real value is 0.5), and $x = 14 = 001110_2$ (the real value is 1.75), where $\ell = 6$ and $\ell_f = 3$. During the offline phase, both P_0 and P_1 own a table $\text{LUT} = \{2^2, 2^1 - 2^2, 2^0 - 2^1, 2^{-1} - 2^0, 2^{-2} - 2^{-1}, 2^{-3} - 2^{-2}\}$. During the online phase, we have $\llbracket k \rrbracket = \{\llbracket 1 \rrbracket, \llbracket 1 \rrbracket, \llbracket 1 \rrbracket, \llbracket 1 \rrbracket, \llbracket 0 \rrbracket, \llbracket 0 \rrbracket\}$ by comparing $\llbracket x \rrbracket$ with $\{2^0, 2^1, 2^2, 2^3, 2^4\}$. Then, we retrieve results for the scaling factor using secure scalar multiplication and addition, i.e., $\llbracket 2^{-1} \rrbracket = \sum_{j=1}^{\ell-1} \llbracket k_j \rrbracket \cdot \text{LUT}[j] = \llbracket 1 \rrbracket \cdot 2^1 + \llbracket 1 \rrbracket \cdot (2^0 - 2^1) + \llbracket 1 \rrbracket \cdot (2^{-1} - 2^0) + \llbracket 0 \rrbracket \cdot (2^{-2} - 2^{-1})$. Thus, we can scale $\llbracket x \rrbracket$ into $\llbracket b \rrbracket = \llbracket x \rrbracket \cdot \llbracket 2^{-k} \rrbracket = \llbracket 000111_2 \rrbracket = \llbracket 7 \rrbracket$ (its real value is 0.875). After that, we follow (12) to securely compute $\llbracket u/x \rrbracket =$

³In the MPC, the real range $(0.5, 1]$ are mapped into $(2^{\ell_f - 1}, 2^{\ell_f}]$.

$\llbracket 0.25 \rrbracket$, which has a loss of 0.035 compared to $u/x \approx 0.285$ in plaintext.

VI. SWAN: A TWO-PARTY PVDT FRAMEWORK

This section provides the details of our Swan framework for both training and inference.

A. Secure Training

To protect sample distribution on tree node during training, a secret-shared indicator $\llbracket \gamma \rrbracket = \{\llbracket \gamma_0 \rrbracket, \llbracket \gamma_1 \rrbracket, \dots, \llbracket \gamma_{N-1} \rrbracket\}$ is used to represent which sample is available for the tree node, where $\gamma_i = 1$ indicates that the i th sample on the current tree node is available for $i \in [0, N-1]$, otherwise $\gamma_i = 0$. At the beginning of the training process, P_0 and P_1 initialize the root node of the decision tree with $\llbracket \gamma \rrbracket = \{\llbracket 1 \rrbracket, \llbracket 1 \rrbracket, \dots, \llbracket 1 \rrbracket\}$. Next, P_0 and P_1 jointly compute the gain of all possible splits and select the best split with the maximum gain. After that, two parties P_0 and P_1 use the best split to divide the node into left and right children nodes with indicators $\llbracket \gamma_l \rrbracket$ and $\llbracket \gamma_r \rrbracket$, respectively. This process is repeated recursively for each child node until the pruning conditions of the tree are satisfied.

During training, Swan establishes input-independent pre-processes to enhance the efficiency of the online computation. In the online phase, each party P_α can train a private DT model T_α with the maximum depth H over its vertical dataset \mathcal{D}_α . At one tree node of T_α , the party P_α that holds f^* th feature can records a pair of the best feature index and best split threshold (i.e., (f^*, s_{f^*})) into the current node. while a tag null is recorded in the corresponding node of $T_{1-\alpha}$, where s_{f^*} represents the best split for the current node. If the current node is the leaf node, a secret-shared best classes $\llbracket c^* \rrbracket_\alpha$ is recoded into T_α . A full DT tree T is composed by T_0 and T_1 .

Algorithm 5 describes the secure training protocol for one DT node on the vertical datas, which consists of three steps: computing the label of the leaf node, selecting the best split, and updating the indicator.

Step 1: Computation of leaf node (Lines 1-10): The step is to assign the most frequent label to the leaf node. In this process, P_1 and P_0 locally construct $\llbracket Y_c \rrbracket$ for each $c \in \mathcal{C}$. Next, P_1 and P_0 compute the available samples for class c at the current node, i.e., $\llbracket \beta_c \rrbracket = \llbracket Y_c \rrbracket \cdot \llbracket \gamma \rrbracket$. Once the current node becomes a leaf node, the sample number for each label c is computed as $\llbracket \mathbf{L} \rrbracket = \{\llbracket L_c \rrbracket = \sum \llbracket \beta_c \rrbracket\}$. After that, the ArgMax protocol with a binary search technique (detailed in Appendix A, available online) is used to determine the class $\llbracket c^* \rrbracket$ with the maximum sample. Finally, the secret share $\llbracket c^* \rrbracket_\alpha$ is recorded into the leaf node of T_α .

Step 2: Selection of the best split (Lines 11-32): If the current node is not a leaf node, the objective of DT training is to determine the best split for the current node. Each party P_α divides its local dataset \mathcal{D}_α into two partitions. That is, $D_l^\alpha = \{i | (x_{i,j} < s_{j,k}) \wedge (i \in D)\}$ and $D_r^\alpha = D - D_l^\alpha$, where $x_i = \{x_{i,j}\} \in \mathcal{D}_\alpha$ is the j th feature of the i th sample. Following D_l^α and D_r^α , each party P_α constructs its local indicator for the two partitions, i.e., $\llbracket t_l^\alpha \rrbracket$ (for the left partition) and $\llbracket t_r^\alpha \rrbracket$ (for the right partition). Next, we invoke secure multiplication protocols to compute the

Algorithm 5: Secure Vertical Tree Training.

Input: Each party P_α owns a local training dataset \mathcal{D}_α , a local potential splits $S_\alpha = \{s_{j,k}^\alpha \mid j \in [0, f_\alpha], k \in [0, V-1]\}$, a secret-shared sample indicator $\llbracket \gamma \rrbracket_\alpha$, the current tree depth h .

Output: A private DT model T_α for each party P_α .

Hyperparameter: Maximum depth H ; Sample number N ; Feature number F ; Split number of each feature V ; Label set $\mathcal{C} = [0, C]$.

// Step 1: Label computation of leaf node

- 1: $P_0 \& P_1$ locally constructs the secret-shared sample labels $\llbracket Y_c \rrbracket$ with class $c \in \mathcal{C}$ and compute the available $\llbracket \beta_c \rrbracket = \llbracket Y_c \rrbracket \cdot \llbracket \gamma \rrbracket$ in parallel for the current node.
- 2: Inite $\llbracket \mathbf{L}_c \rrbracket = \{\}$
- 3: **if** $h = H$ **then**
- 4: **for** $c \in \mathcal{C}$ **in paralle do**
- 5: **[Sample number with class c]:** $\llbracket L_c \rrbracket = \sum \llbracket \beta_c \rrbracket$, where \sum is element-wise addition operation.
- 6: **[Put $\llbracket L_c \rrbracket$ into the vector $\llbracket \mathbf{L} \rrbracket$]:** $\llbracket \mathbf{L} \rrbracket \leftarrow \llbracket L_c \rrbracket$.
- 7: **end for**
- 8: **[Compute class c^* with maximum number]:** $(\llbracket c^* \rrbracket, _) = \text{ArgMax}(\llbracket \mathbf{L} \rrbracket, \llbracket \mathcal{C} \rrbracket)$ where the secret-shared classes $\llbracket \mathcal{C} \rrbracket$ is constructed locally.
- 9: **[Record c^* into T_α]:** The private model T_α records the triple $(\llbracket c^* \rrbracket)$.
- 10: **end if**

// Step 2: Selection of the best split

- 11: Initialize an empty set $\llbracket \text{gain}_b \rrbracket = \{\}$.
- 12: Initialize a feature index vector $\mathbf{f} = \{\llbracket 0 \rrbracket, \dots, \llbracket F-1 \rrbracket\}$.
- 13: **for** $j \in [0, F_\alpha - 1]$ **in paralle do**
- 14: Initialize an empty set $\llbracket \text{gain}_f \rrbracket = \{\}$; Initialize a bin index vector $\mathbf{v} = \{\llbracket 0 \rrbracket, \dots, \llbracket V-1 \rrbracket\}$
- 15: **for** $k \in [0, V-1]$ **in paralle do**
- 16: **[Divide their respective samples into two partitions]:** $D_l^\alpha = \{i | (x_{i,j} < s_{j,k}) \wedge (i \in D)\}$, where $x_{i,j} \in \mathcal{D}_\alpha$ is the j th feature of the i th sample and $s_{j,k} \in S_\alpha$; $D_r^\alpha = D - D_l^\alpha$.
- 17: **[Construct a local indicator for left/right subset]:** $\mathbf{t}_l^\alpha = \{t_i = 1 \mid i \in D_l^\alpha\} \cap \{t_i = 0 \mid i \notin D_l\}$; $\llbracket \mathbf{t}_l^\alpha \rrbracket = \llbracket 0 \rrbracket + \mathbf{t}_l^\alpha$ (resp., $\llbracket \mathbf{t}_r^\alpha \rrbracket$ for right subset).
- 18: **[Sample number on subsets]:** $\llbracket |D_l^\alpha| \rrbracket = \sum (\llbracket \mathbf{t}_l^\alpha \rrbracket \cdot \llbracket \gamma \rrbracket)$; (resp. $\llbracket |D_r^\alpha| \rrbracket$).
- 19: Initialize $\llbracket g_k \rrbracket = \llbracket 0 \rrbracket$.
- 20: **for** $c \in \mathcal{C}$ **in paralle do**
- 21: **[Sample number with class c on subsets]:** $\llbracket |D_{l,c}^\alpha| \rrbracket = \sum (\text{NMUL}(\llbracket \mathbf{t}_l^\alpha \rrbracket, \llbracket \gamma \rrbracket, \llbracket \beta_c \rrbracket))$ (resp., $\llbracket |D_{r,c}^\alpha| \rrbracket$)
- 22: **[Compute gain with (3)]** $\llbracket \rho_l \rrbracket = \text{DIV}(1, \llbracket |D_l^\alpha| \rrbracket)$ (resp. $\llbracket \rho_r \rrbracket$); $\llbracket g_k \rrbracket = \llbracket g_k \rrbracket + \text{NMUL}(\llbracket \rho_r \rrbracket, \llbracket |D_{l,c}^\alpha| \rrbracket, \llbracket |D_{r,c}^\alpha| \rrbracket) + (\llbracket \rho_r \rrbracket, \llbracket |D_{r,c}^\alpha| \rrbracket, \llbracket |D_{r,c}^\alpha| \rrbracket)$.
- 23: **end for**
- 24: **[Put $\llbracket g_k \rrbracket$ into $\llbracket \text{gain}_f \rrbracket$]:** $\llbracket \text{gain}_f \rrbracket \leftarrow \llbracket g_k \rrbracket$.
- 25: **end for**
- 26: **[Best gain of the j th feature]:** $(\llbracket g_j^* \rrbracket, \llbracket v_j^* \rrbracket) = \text{Argmax}(\llbracket \text{gain}_f \rrbracket, \llbracket \mathbf{v} \rrbracket)$.

- 27: **[Open the index v_j^* of the split]:** P_w receives $\llbracket v_j^* \rrbracket_{1-w}$ from P_{1-w} ; P_o opens $v^*_j = \llbracket v_j^* \rrbracket_{1-w} + \llbracket v_j^* \rrbracket_w$, where P_w is the owner of the j th feature.
- 28: **[Put $\llbracket g_j^* \rrbracket$ into $\llbracket \text{gain}_b \rrbracket$]:** $\llbracket \text{gain}_b \rrbracket \leftarrow \llbracket g_j^* \rrbracket$.
- 29: **end for**
- 30: **[Best gain of the current node]:** $(\llbracket g_h^* \rrbracket, \llbracket f^* \rrbracket) = \text{Argmax}(\llbracket \text{gain}_b \rrbracket, \llbracket f \rrbracket)$.
- 31: **[Determine the owner of the best feature]:** $\llbracket o \rrbracket^B = \text{SecMSB}(F_0 - \llbracket f^* \rrbracket)$.
 $o = \text{Rec}(\llbracket o \rrbracket^B)$. \triangleright if $o = 0$, P_0 owns the f^* th feature, otherwise P_1 .
- 32: **[Record the best s_{f^*,k^*} into T_α]:** P_o receives $\llbracket f^* \rrbracket_{1-o}$ from P_{1-o} and opens $f^* = \llbracket f^* \rrbracket_o + \llbracket f^* \rrbracket_{1-o}$; Let $k^* = v_{f^*}^*$, $s_{f^*} = s_{f^*,k^*}$. P_o records $T_o \leftarrow (f^*, s_{f^*})$; P_{1-o} records $T_{1-o} \leftarrow \text{null}$.
- // Step 3: Indicator Update**
- 33: P_o divides D into two best partitions:
 $D_l^* = \{i \mid (x_{i,j} < s_{f^*,k^*}) \wedge (i \in D_o)\}$, $D_r^* = D - D_l^*$.
- 34: P_o constructs a local indicator for best left/right subset:
 $\mathbf{t}_l^* = \{t_i = 1 \mid i \in D_l^*\} \cap \{t_i = 0 \mid i \notin D_l^*\}$;
 $\llbracket \mathbf{t}_l^* \rrbracket = \llbracket \mathbf{0} \rrbracket + \mathbf{t}_l^*$.
- 35: **[Update left/right indicator]:** $\llbracket \gamma_l \rrbracket = \llbracket \mathbf{t}_l^* \rrbracket \cdot \llbracket \gamma \rrbracket$ and $\llbracket \gamma_r \rrbracket = \llbracket \gamma \rrbracket - \llbracket \gamma_l \rrbracket$.
- Repeat **Step 1-3** to construct the left sub-tree and the right sub-tree until the maximum depth H of T_α .
- return** A private DT model T_α .

sample number and each class number in the two partitions, i.e., $\llbracket |D_l^\alpha| \rrbracket$, $\{\llbracket |D_{l,c}^\alpha| \rrbracket \mid \forall c \in \mathcal{C}\}$ (for the left partition), and $\llbracket |D_r^\alpha| \rrbracket$, $\{\llbracket |D_{r,c}^\alpha| \rrbracket \mid \forall c \in \mathcal{C}\}$ (for the right partition). Following (3), we use NMUL protocol and DIV to compute the gains $\{\llbracket g_{f,k} \rrbracket\}$ for all possible splits $S = \{s_{j,k} \mid j \in [0, F-1], k \in [0, V-1]\}$ at the current node. Subsequently, the ArgMax protocol with a binary search technique is used to determine the feature index $\llbracket f^* \rrbracket$ and the split index $\llbracket k^* \rrbracket$ corresponding to the maximum gain $\llbracket g \rrbracket$. After that, (f^*, k^*) is opened for the party P_o who holds the f^* th feature. If P_α holds the f^* th feature, the best split (f^*, s_{f^*}) is recorded into the current node of T_α . And the tag *null* is recoded in the corresponding node of $T_{1-\alpha}$.

Step 3: Indicator update (Lines 33-35): After obtaining the best split (f^*, s_{f^*}) , the feature owner P_o divides the current samples into two partitions, i.e., $D_l^* = \{i \mid (x_{i,j} < s_{f^*,k^*}) \wedge (i \in D_o)\}$ and $D_r^* = D - D_l^*$. Then, the party P_o constructs a local indicator for the left/right partition, i.e., $\mathbf{t}_l^* = \{t_i = 1 \mid i \in D_l^*\} \cap \{t_i = 0 \mid i \notin D_l^*\}$ and $\llbracket \mathbf{t}_l^* \rrbracket = \llbracket \mathbf{0} \rrbracket + \mathbf{t}_l^*$. Finally, the current indicator $\llbracket \gamma \rrbracket$ is updated into the left indicator $\llbracket \gamma_l \rrbracket = \llbracket \mathbf{t}_l^* \rrbracket \cdot \llbracket \gamma \rrbracket$ and the right indicator $\llbracket \gamma_r \rrbracket = \llbracket \gamma \rrbracket - \llbracket \gamma_l \rrbracket$.

Fig. 3 illustrates an example of the secure training process. Suppose that P_0 is a bank that owns the income feature, and P_1 is an e-commerce company that owns the purchase feature along with labels for two classes: class 1 and class 2. The goal of P_0 and P_1 is to split the current tree node with an indicator $\llbracket \gamma \rrbracket = \{\llbracket 1 \rrbracket, \llbracket 1 \rrbracket, \llbracket 0 \rrbracket, \llbracket 1 \rrbracket\}$, meaning that samples 1, 2, and 4 are available in the current tree node.

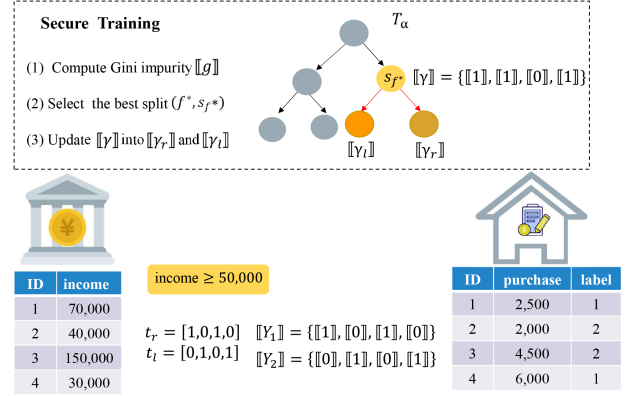


Fig. 3. Secure Training of PVDT in Swan.

All of first, P_0 and P_1 can locally construct a secret-shared $\llbracket Y_1 \rrbracket = \{\llbracket [1] \rrbracket, \llbracket [0] \rrbracket, \llbracket [0] \rrbracket, \llbracket [1] \rrbracket\}$ for class 1 without any communication⁴ (resp. $\llbracket Y_2 \rrbracket$ for class 2). Next, each party P_α performs a secure element-wise multiplication with to compute $\llbracket \beta_1 \rrbracket = \llbracket Y_1 \rrbracket \cdot \llbracket \gamma \rrbracket$, where β_1 represents whether samples with class 1 are available for the node (resp. $\llbracket \beta_2 \rrbracket$). Suppose that P_0 considers a split to divide the current node according to $\text{income} \geq 50,000$. P_0 first locally divides the training samples into two partitions by examining whether the income values of the sample are less than or equal to 50,000, where $\mathbf{t}_l = \{1, 0, 1, 0\}$ for the left partition and $\mathbf{t}_r = \{0, 1, 0, 1\}$ for the right partition. Next P_0 and P_1 perform a secure multiplication between \mathbf{t}_r and β_1 to obtain shared $|D_{l,1}|$, which represents the total number of available samples with class 1 (resp. $|D_{l,2}|$). Using the same way, P_0 and P_1 can compute $|D_l|$ and $|D_r|$. Following (3) and the proposed fundamental protocols, P_0 and P_1 jointly compute the gains $\{\llbracket g_{f,v} \rrbracket\}$ for all possible splits $S = \{s_{j,k} \mid j \in [0, f], k \in [0, v]\}$ at the current node. Subsequently, the ArgMax protocol is employed to determine the feature index $\llbracket f^* \rrbracket$ and the split index $\llbracket k^* \rrbracket$ corresponding to the maximum gain $\llbracket g \rrbracket$. Following the approach in PriVDT, we compute a boolean bit $o = \text{Rec}(\text{SC}(f^* - \llbracket f_0 \rrbracket))$ to identify the owner of the best feature. If $o = 0$, f^* and k^* are opened for P_0 ; otherwise, they are opened to P_1 . Assuming P_α holds the f^* th feature, it records (f^*, s_{f^*,k^*}) as the best split (f^*, s_{f^*}) in the current node of T_α . In contrast, $P_{1-\alpha}$, which does not own the f^* th feature, records a null tag in the corresponding node of $T_{1-\alpha}$. Finally, P_α partitions its dataset using the selected best split, updating the current indicator $\llbracket \gamma \rrbracket$ into $\llbracket \gamma_l \rrbracket$ (for the left partition) and $\llbracket \gamma_r \rrbracket$ (for the right partition) at the corresponding tree nodes.

Both P_0 and P_1 loop through the above training process until reaching the maximum depth. From the process, we can see the two parties do not exchange any data in plaintext. Other than the tree shape of T_α , all the other information on P_α is hidden from $P_{1-\alpha}$.

⁴The label owner P_1 locally sets $\llbracket Y_1 \rrbracket_1 = \{(0, -1), (0, 0), (0, 0), (0, -1)\}$ that indicates samples with class 1 (resp., $\llbracket Y_2 \rrbracket_1 = \{(0, 0), (0, -1), (0, -1), (0, 0)\}$ for class 2). P_0 locally sets $\llbracket Y_1 \rrbracket_0 = \{(0, 0), (0, 0), (0, 0), (0, 0)\}$ (resp., $\llbracket Y_2 \rrbracket_0 = \{(0, 0), (0, 0), (0, 0), (0, 0)\}$).

Algorithm 6: Secure Vertical Tree Inference.

Input: A private tree T_α with size of 2^{H-1} held by P_α ; A new sample $\mathbf{x} = \mathbf{x}^0 \parallel \mathbf{x}^1$, where P_0 holds $\mathbf{x}_0 = \{\mathbf{x}[0], \dots, \mathbf{x}[F_0 - 1]\}$ with F_0 features and P_1 holds $\mathbf{x}_1 = \{\mathbf{x}[F_0 - 1], \dots, \mathbf{x}[F - 1]\}$ with $F - F_0$ features.

Output: The secret sharing inference result $\llbracket c^* \rrbracket$

- 1: $\mathbf{pth}_\alpha = \mathbf{0}$
- 2: Set a empty queue $\mathbf{Q} = \{\}$ and $\llbracket \mathcal{C} \rrbracket = \{\}$
- 3: $\mathbf{Q}.push(T_\alpha.root, h = 0)$
//forlevel – ordertraversalofatree
- 4: **while** $\mathbf{Q}.size \neq null$ **do**
- 5: $node \leftarrow \mathbf{Q}.pop()$;
- 6: **if** $node \neq null$ and $h < H$ **then**
- 7: $(f^*, s_{f^*}) \leftarrow node$; $\alpha = 1 - (f^* < F_0)$.
- 8: P_α updates \mathbf{pth}_α via $\mathbf{x}[f^*] < s_{f^*}$.
- 9: $\mathbf{Q}.push(node.left_node, h + 1)$;
 $\mathbf{Q}.push(node.right_node, h + 1)$.
- 10: **end if**
- 11: **if** $h = H$ **then**
- 12: $\llbracket c \rrbracket \leftarrow node$; Put $\llbracket c \rrbracket$ into the set $\llbracket \mathcal{C} \rrbracket$.
- 13: **end if**
- 14: **end while**
- 15: **[Build access path]:**
 $\llbracket \eta_0 \rrbracket_\alpha = (\mathbf{0}, -(1 - \alpha) \cdot \mathbf{pth}_\alpha)$, $\llbracket \eta_1 \rrbracket_\alpha = (\mathbf{0}, \alpha \cdot \mathbf{pth}_\alpha)$
- 16: **[Compute inference result]:**
 $\llbracket c^* \rrbracket = \sum 3\text{MUL}(\llbracket \eta_0 \rrbracket, \llbracket \eta_1 \rrbracket, \llbracket \mathcal{C} \rrbracket)$.

B. Secure Inference

To accelerate the inference of the private tree T_α with the depth H , we propose a secure parallelized inference protocol via 3-input multiplication protocols, as outlined in Algorithm 6. For a new input sample, each party can locally prepare a boolean vector $\mathbf{pth}_\alpha = \mathbf{0}$ with size of $(2^{(H-1)} - 1)^5$ for each tree. Recall that P_α holds the tree part T_α which includes its features. The elements of \mathbf{pth}_α are updated by comparing his features and the best split of the corresponding node. That is, $\mathbf{pth}_\alpha[j] = 1$ indicates that the sample might be classified to the j th leaf (the leaves are ordered from left to right), while $\mathbf{pth}_\alpha[j] = 0$ means the input will not be classified to the j th leaf based on the split identifiers held by P_α . At the end, there is only one entry of ‘1’ in $\mathbf{pth}_0 \cdot \mathbf{pth}_1$. After that, we define $\llbracket \eta_0 \rrbracket_0 = (\mathbf{0}, -\mathbf{pth}_0)$, $\llbracket \eta_1 \rrbracket_0 = (\mathbf{0}, \mathbf{0})$, held by P_0 and $\llbracket \eta_1 \rrbracket_0 = (\mathbf{0}, \mathbf{0})$, $\llbracket \eta_1 \rrbracket_0 = (\mathbf{0}, -\mathbf{pth}_1)$ held by P_1 . The final prediction on the tree is computed as $\llbracket c^* \rrbracket = \sum 3\text{MUL}(\llbracket \eta_0 \rrbracket, \llbracket \eta_1 \rrbracket, \llbracket \mathcal{C} \rrbracket)$, where $\llbracket \mathcal{C} \rrbracket$ indicates the labels of all leaf nodes.

Fig. 4 illustrates an example of the secure inference process. Assume that we have a well-trained tree $T = \{T_0, T_1\}$ with $H = 3$, where the private tree T_0 is held by P_0 and the private tree T_1 is held by P_1 . The tree T_0 includes the conditions $purchase \geq 3,000$ and $income \geq 50,000$, while the tree T_1 contains the condition $age \geq 40$. Both T_0 and T_1 share a secret-shared label vector $\llbracket \mathcal{C} \rrbracket = \{\llbracket 1 \rrbracket, \llbracket 2 \rrbracket, \llbracket 1 \rrbracket, \llbracket 2 \rrbracket\}$, ensuring privacy-preserving classification. For a new sample, where the features ($purchase = 2,000$, $income = 60,000$) are held by P_0 and the

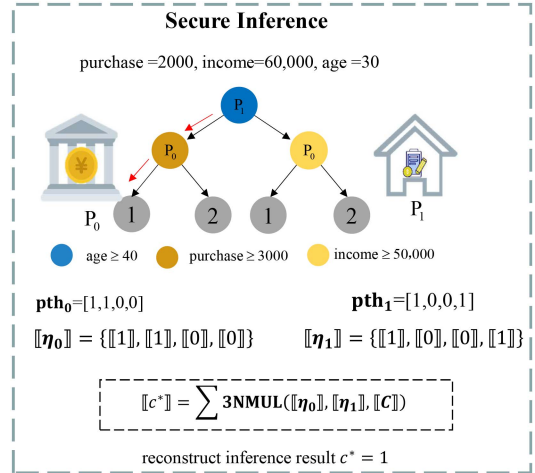


Fig. 4. Secure Inference of Vertical Decision Trees in Swan.

feature $age = 30$ is held by P_1 , the secure inference process proceeds as follows: P_0 initializes a boolean vector $\mathbf{pth}_0 = [0, 0, 0, 0]$ (resp., $\mathbf{pth}_1 = [0, 0, 0, 0]$ for P_1). The elements of \mathbf{pth}_α are updated by comparing his features and the best split of the corresponding node. That is, $\mathbf{pth}_0 = [1, 1, 0, 0]$ and $\mathbf{pth}_1 = [1, 0, 0, 1]$. After that, let $\llbracket \eta_0 \rrbracket_0 = (\mathbf{0}, -\mathbf{pth}_0)$, $\llbracket \eta_1 \rrbracket_0 = (\mathbf{0}, \mathbf{0})$, and $\llbracket \eta_1 \rrbracket_0 = (\mathbf{0}, \mathbf{0})$, $\llbracket \eta_1 \rrbracket_0 = (\mathbf{0}, -\mathbf{pth}_1)$. Thus, we have $\llbracket \eta_0 \rrbracket = \{\llbracket 1 \rrbracket, \llbracket 1 \rrbracket, \llbracket 0 \rrbracket, \llbracket 0 \rrbracket\}$ $\llbracket \eta_1 \rrbracket = \{\llbracket 1 \rrbracket, \llbracket 0 \rrbracket, \llbracket 0 \rrbracket, \llbracket 1 \rrbracket\}$. The final prediction on the tree is computed as $\llbracket c^* \rrbracket = [1] = \sum 3\text{MUL}(\llbracket \eta_0 \rrbracket, \llbracket \eta_1 \rrbracket, \llbracket \mathcal{C} \rrbracket)$.

VII. THEORETICAL ANALYSIS**A. Security Analysis**

We prove the security of all proposed secure protocols (including NMUL, SecMSB, SC, DIV, secure training and secure inference protocols) against semi-honest attackers under the standard security definition [38]. Please refer to Theorem 1–4 in Appendix B, available online for detailed security proofs.

B. Communication Complexity Analysis

We analyze the performance of Swan’s secure protocols regarding communication overhead and rounds. Table VI summarizes the online performance analysis of our secure protocols over the ring \mathbb{Z}_{2^ℓ} , i.e., $\ell = 64$. As a comparison, we also list the corresponding results of PriVDT [9] and Pivot [8]. Please refer to Appendix C, available online for detailed communication complexity. For the sake of simplicity, Table III summarizes the online performance analysis of our secure protocols over the ring $\mathbb{Z}_{2^{64}}$. We set $\ell_e = 1024$, $N = 8$, $F = 2$, $V = 4$, $C = 2$, $H = 4$, and $d = 8$ in the Table VI.

VIII. EXPERIMENTAL EVALUATION**A. Implementation and Experimental Setup**

We implement a prototype system with the PyTorch framework with our NssMPClib.⁶ The evaluations of our secure protocols are performed on two docker servers running Ubuntu 18.04,

⁵A full tree with the depth H has $2^H - 1$ internal nodes.⁶<https://github.com/XidianNss/NssMPClib>

TABLE III
ONLINE COMMUNICATION COMPLEXITY OF SWAN, PRIVDT, AND PIVOT OVER THE RING $\mathbb{Z}_{2^{64}}$

Function	Approach	Comm. Round	Comm. Cost (bits)
NMUL	Pivot	2	$1024n + 128n$
	PriVDT	$n - 1$	$128(n - 1)$
	Ours	1	$64n$
SC	Pivot	7	10,684
	PriVDT	2	192
	Ours	2	143
DIV	Pivot	34	84,290
	PriVDT	92	22,523
	Ours	6	17,792
Label Computation	Pivot	7	43,248
	PriVDT	7	2,560
	Ours	5	2,416
Best Split Selection	Pivot	1,364	1,775,616
	PriVDT	1,136	880,800
	Ours	132	104,449
Indicator Update	Pivot	2	9,216
	PriVDT	3	2,560
	Ours	1	512
Inference	Pivot	1	15,360
	PriVDT	45	6,272
	Ours	1	960

with Intel(R) Core(TM) i7-8700 CPU @ 3.20 GHz and 32 GB of RAM, where each server represents a party. we adopt the TCP protocol to establish point-to-point communication between the two parties. This setup ensures reliable and ordered message delivery, and facilitates the integration of our secure protocol within standard networking environments. All experiments are conducted in LAN and WAN network settings. For the experiments in a LAN network, we host the two servers in the same region with an average network delay of 0.29 ms and a bandwidth of 1 GB/s. For the experiments in a WAN network, we host the two servers in different regions with an average network delay of 72 ms and a bandwidth of 20 MB/s. Consistent with prior work [8], [9], we set all of the secure protocols over the ring $\mathbb{Z}_{2^{64}}$, i.e., $\ell = 64$, and we utilize fixed-point numbers to encode the inputs, with a fractional part of $\ell_f = 20$ bits.

Dataset: We conduct experiments using public datasets from the UCI machine learning repository:⁷ (1) The Iris plants (IS) dataset consists of 150 instances divided into three classes, each with four features. (2) The Breast Cancer Wisconsin (BCW) dataset consists of 569 instances divided into two classes, each with 30 features. (3) The Bank Marketing (BM) dataset consists of 4,521 instances distributed over two classes, each with 17 features. (4) The Credit Card dataset (CC) consists of 30,000 instances divided into three classes, each with 23 features. We follow a ratio of 8:2 for training and testing datasets for every dataset. The features of each training dataset are evenly distributed between P_0 and P_1 , with P_1 holding all of the sample labels. Assume that the samples in each party's database have been properly aligned beforehand.

Baselines: We compare Swan with three baselines: the non-private decision tree (NP-DT), Pivot [8], and PriVDT [9]. For

TABLE IV
ACCURACY COMPARISON BETWEEN SWAN AND THE BASELINES

Dataset	Tree Depth	Swan	PriVDT	Pivot	NP-DT
IS	3	96.61%	96.61%	96.61%	96.66%
CC	3	82.33%	82.01%	82.47%	82.80%
BM	4	90.38%	88.50%	90.05%	90.71%
BCW	5	93.85%	93.01%	93.47%	95.61%

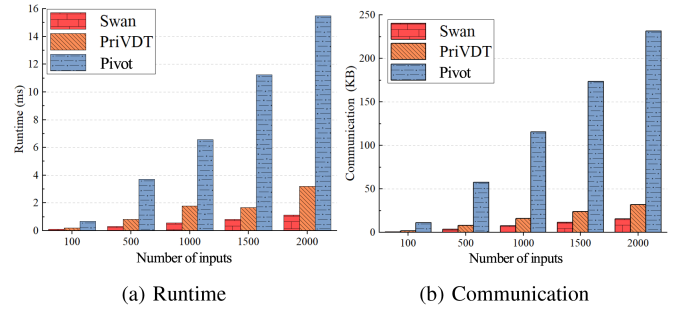


Fig. 5. Performance comparison of N-input multiplication protocol.

Pivot, we run it in the same experimental setup with Swan, with the help of their implementation code.⁸ For PriVDT, its implementation is referred to in their paper. To guarantee a fair comparison, we set consistent hyperparameters for PVDT across all approaches. We measure time (including both communication time and computation time) to evaluate system performance, while quantifying total communication overhead through comprehensive logging of all messages exchanged between two party. To ensure statistical reliability, each experimental data point represents the average of 10 independent trials conducted under identical conditions.

B. Evaluation of Accuracy

To evaluate the accuracy of Swan, we compare its performance with NP-DT, Pivot, and PriVDT. This allows us to demonstrate the effectiveness of Swan in achieving reliable results with the privacy guarantee. As shown in Table IV, Swan achieves comparable accuracy performance to PriVDT and Pivot. Similarly to PriVDT and Pivot, Swan also has a slight accuracy loss compared to NP-DT, which stems from the encoding of fixed-point numbers in all secure protocols and the approximate division protocol. The loss is inherent in a MPC system and can not be eliminated.

C. Microbenchmarks of the Building Blocks

We compare the performance of the building blocks in Swan, PriVDT [9], and Pivot [8] under the LAN network settings. Fig. 5 shows the performance of secure N -input multiplication protocol in PriVDT, Pivot, and Swan. PriVDT employs the naïve solution to implement a secure multiplication protocol. The solution for N -input multiplication requires $n - 1$ communication rounds and considerable communication overhead.

⁷<https://archive.ics.uci.edu/>

⁸<https://github.com/nusdbsystem/pivot>

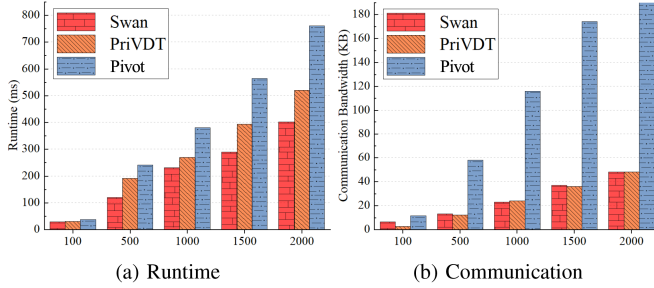


Fig. 6. Performance comparison of SC protocol.

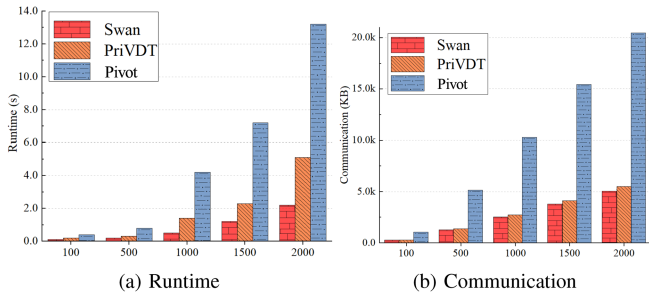


Fig. 7. Performance comparison of DIV protocol.

Pivot employs the threshold Paillier scheme and ASS to build secure multiplication operations. Since the Paillier scheme is an additive homomorphic encryption (AHE) scheme, Pivot must employ an AHE-to-ASS conversion (cf. Algorithm 2 in Pivot) for the N -input multiplication task. However, this process incurs significant communication and computational overhead. In contrast, Swan’s protocol is more efficient than those of Pivot and PriVDT since Swan employs a lightweight $[\cdot]$ -sharing and a secure chained multiplication strategy to reduce computation and communication overheads.

Fig. 6 shows the performance of SC protocol in PriVDT, Pivot, and Swan. PriVDT utilizes the FSS-based SC protocol to reduce communication overhead. However, it depends on a trusted server to generate and distribute FSS keys for both parties. Also, PriVDT incurs significant computational overhead in the online phase due to the time required for generating pseudorandom seeds for FSS. Pivot employs the SC protocol from [42], which necessitates frequent interactions between the two parties during the online phase, resulting in significant runtime overhead. In contrast, Swan leverages 0/1 encoding and parallelization techniques to implement the SC protocol, requiring only two communication rounds. The communication overhead of Swan exhibits dynamic fluctuations due to the varying size of the 0-encoding of the random public integer. Nevertheless, as shown in Fig. 6(a), Swan outperforms both Pivot and PriVDT in runtime.

Fig. 7 shows the performance of DIV protocol in PriVDT, Pivot, and Swan frameworks. PriVDT employs an oblivious transfer protocol and secure bit decomposition protocol to build DIV protocol. Nevertheless, this optimization requires a time-consuming and communication-heavy bit-decomposition

protocol in the online phase. Pivot uses the DIV protocol in [42]. The protocol also requires frequent interaction between two parties such that it invariably results in heavy computation and communication overhead. In contrast, Swan applies a secure lookup table without communication and NMUL protocols to construct secure divisions. To reduce latency, our protocols are combined with multithreading technology. As shown in the Fig. 7, the DIV protocol of Swan outperforms PriVDT and Pivot.

D. Performance Comparison With Baselines

We give a performance evaluation of the secure training and inference protocol of Swan and compare them with two baselines, PriVDT and Pivot, as shown in Table V. In terms of communication overhead, the secure training protocol of Swan saves $13.3\times \sim 27.6\times$ than that of Pivot and $1.2\times \sim 2.0\times$ than that of PriVDT. The secure inference protocol of Swan saves $6.5\times \sim 12.2\times$ and $1.5\times \sim 3.8\times$ than Pivot and PriVDT, respectively. In terms of runtime, the secure training protocol of Swan in the LAN setting achieves $36.9\times \sim 58.6\times$ and $2.0\times \sim 3.0\times$ speedup over Pivot and PriVDT, respectively. The secure inference protocol of Swan in the LAN setting achieves $6.2\times \sim 11.4\times$ and $1.9 \sim 2.6\times$ faster than Pivot and PriVDT, respectively. This improvement is because the secure building blocks of Swan are more lightweight than PriVDT and Pivot, as shown in Section VIII-C. In the WAN setting, we observe that Swan also outperforms Pivot and PriVDT. During the training phase, Swan is $109.3\times \sim 188.3\times$ and $3.0\times \sim 10.2\times$ faster than Pivot and PriVDT, respectively. During the inference phase, Swan demonstrates $3.9\times \sim 4.4\times$ and $2.2\times \sim 2.8\times$ than Pivot and PriVDT, respectively. This is because the communication latency is lower in the LAN setting than in the WAN setting. The cryptographic operations dominate the runtime in the LAN setting, and the communication latency is the main performance bottleneck in the WAN setting. Therefore, Swan reduces communication rounds of our all-secure protocol to minimize communication latency. As shown in Table III, we can see that the communication rounds of Swan are significantly smaller than the method used in Pivot and PriVDT.

E. Scalability Evaluation on Varying Parameters

To evaluate scalability, we compare the performance of Swan with PriVDT [9] and Pivot [8] on the Credit Card dataset with varying parameters. As the default configuration, the dataset consists of $N = 2,000$ samples with $F = 10$ features. The tree depth is set to $H = 3$.

1) *Training Performance With Varying Numbers of Samples:* Figs. 8(a) and 9(a) show the performance comparison for varying numbers of samples in Swan, PriVDT, and Pivot during the training phase. Similarly to PriVDT and Pivot, the running time and communication overhead of Swan increase with the number of samples. Nevertheless, the runtime and communication overhead of Swan still outperform PriVDT and Pivot. Besides, we observe that the runtime of Swan does not double when the sample size doubles. The reason is that we leverage multi-threading parallelization technology to optimize the execution of secure

TABLE V
PERFORMANCE COMPARISON OF SECURE TRAINING AND INFERENCE ON LAN/WAN FOR DIFFERENT DATASETS

Dataset	Work	Training			Inference		
		Time on LAN (s)	Time on WAN (s)	Comm. (MB)	Time on LAN (ms)	Time on WAN (ms)	Comm. (KB)
IS	Pivot	427.06	12260.04	3770.12	29.1	292.12	88.31
	PriVDT	16.83	271.8	482.15	8.2	167.2	25.8
	Swan	7.32	165.12	282.12	4.38	74.45	13.4
CC	Pivot	1413.48	30423.59	27212.87	70.32	268.45	623.12
	PriVDT	48.12	821.28	1121.42	13.42	182.12	135.82
	Swan	23.15	226.23	982.93	7.42	75.42	89.2
BM	Pivot	2012.12	35937.74	28312.91	83.19	287.17	731.32
	PriVDT	92.78	938.70	2091.38	22.21	225.21	349.02
	Swan	42.28	312.74	1025.97	8.21	79.25	97.21
BCW	Pivot	2556.37	48755.81	33212.83	72.12	361.37	1251.78
	PriVDT	138.15	3323.17	2480.90	29.12	221.12	389.21
	Swan	69.12	476.18	1225.27	11.12	82.12	102.37

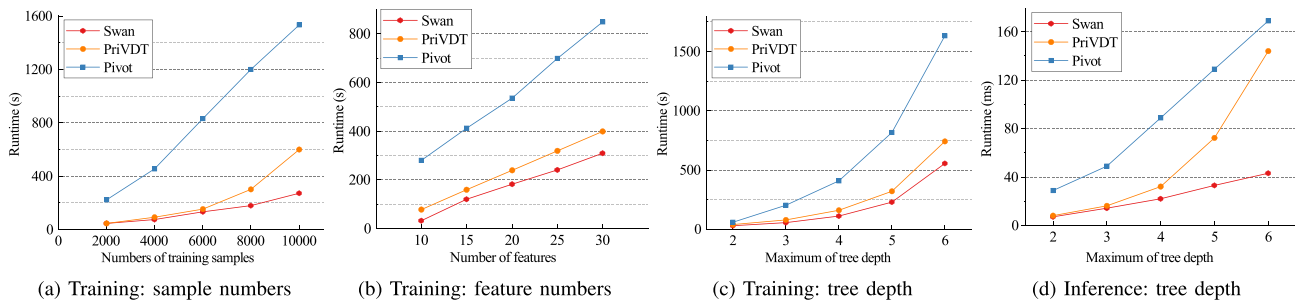


Fig. 8. Runtime of the secure training and inference under various parameters.

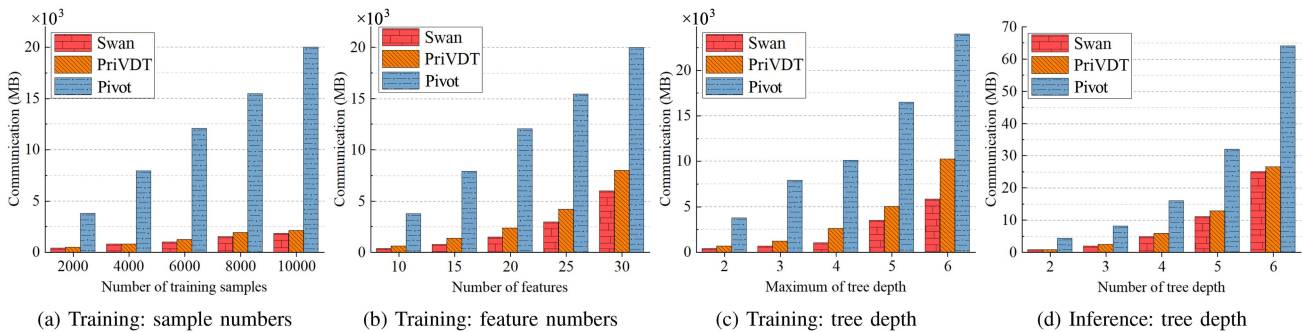


Fig. 9. Communication overhead of the secure training and inference under various parameters.

training protocol. On the other hand, Swan's building blocks also outperform PriVDT and Pivot in terms of computation and communication overhead, such that Swan is faster than PriVDT and Pivot.

2) *Training Performance With Varying Number of Features:* Fig. 8(b) and 9(b) provide the performance for varying numbers of features in Swan, PriVDT, and Pivot during the training phase. The performance trends of the three approaches increase as feature number increases. However, Swan still outperforms PriVDT and Pivot. Besides, we observe that the gap between Swan and Pivot or PriVDT becomes even larger as the number of features increases. This is because the numbers of secure operations increase linearly with feature numbers, such that

the secure training protocols of PriVDT and Pivot need more computation and communication overhead than Swan.

3) *Training Performance With Varying Depth of Trained Tree:* Fig. 8(c) and 9(c) show the performance comparison for varying depth of tree in Swan, PriVDT, and Pivot during the training phase. Obviously, the performance of Swan outperforms PriVDT and Pivot. In addition, the overheads of computation and communication in Swan, PriVDT, and Pivot increase significantly as tree depth increases. The reason for this is that a well-trained model tends to build a full binary tree with $2^{H-1} - 1$ internal nodes, given a maximum tree depth H . Similarly to PriVDT and Pivot, the number of Swan approximately doubles with the depth of the tree.

4) *Inference Performance With Varying Depth of the Tree:* Fig. 8(d) and 9(d) demonstrate the performance comparison for varying depth of tree in Swan, PriVDT, and Pivot during the inference phase. The runtime and communication overhead of Swan increase linearly with the depth of the tree and are lower than those of PriVDT and Pivot. This is because their overhead of secure comparisons and multiplication is higher than that of Swan. Besides, we leverage 3MUL protocol and parallelization technology to optimize secure inference protocol, which reduces large communication rounds.

IX. CONCLUSION AND DISCUSSION

A. Discussion

We explore the possibility of extending Swan to multi-party computation (MPC). Recent works like Meteor [41] and Ruyi [44] have introduced enhanced secret-sharing primitives for three-party and general multi-party settings. By adopting these primitives, we extend Swan's building blocks to multi-party environments. By replacing cryptographic operations in Algorithm 1 (secure training) and 2 (secure inference) with their multi-party building blocks, the protocol seamlessly generalizes to MPC.

Similar to PriVDT and Pivot, Swan currently assumes a semi-honest adversarial model, which may not adequately capture real-world threat scenarios where malicious behavior is possible. Extending Swan to support active security — ensuring correctness and privacy even when parties deviate from the protocol — remains a significant yet essential challenge. Such an extension would require robust zero-knowledge proofs or authenticated secret-sharing mechanisms [45], which introduce non-trivial overhead and design complexity.

B. Conclusion

In this paper, we propose Swan, an efficient framework that incorporates improved secret sharing primitive (iASS) and cryptographic components for secure N -input multiplication, secure comparison, and secure division protocols. Swan's training protocol combines insights from decision trees and lightweight cryptographic building blocks, while its inference protocol is fast and effective. The theoretical analysis shows that Swan has security guarantees and low computational and communication complexity. The results of experiments on real-world datasets demonstrate Swan outperforms the state-of-the-art works in terms of runtime and communication while realizing comparable accuracy to the non-private setting.

REFERENCES

- [1] J. Su and H. Zhang, "A fast decision tree learning algorithm," in *Proc. 21st Nat. Conf. Artif. Intell.*, 2006, pp. 500–505.
- [2] J. Herce-Zelaya, C. Porcel, J. Bernabé-Moreno, A. Tejada-Lorente, and E. Herrera-Viedma, "New technique to alleviate the cold start problem in recommender systems using information from social media and random decision forests," *Inf. Sci.*, vol. 536, pp. 156–170, 2020.
- [3] C. Oswald, S. E. Simon, and A. Bhattacharya, "Spotsam: Intention analysis-driven SMS spam detection using BERT embeddings," *ACM Trans. Web*, vol. 16, no. 3, pp. 1–27, 2022.
- [4] J. Liang, Z. Qin, S. Xiao, L. Ou, and X. Lin, "Efficient and secure decision tree classification for cloud-assisted online diagnosis services," *IEEE Trans. Dependable Secure Comput.*, vol. 18, no. 4, pp. 1632–1644, Jul. 2021.
- [5] J. Basak and R. Kothari, "A classification paradigm for distributed vertically partitioned data," *Neural Comput.*, vol. 16, no. 7, pp. 1525–1544, 2004.
- [6] J. Vaidya, B. Shafiq, W. Fan, D. Mehmood, and D. Lorenzi, "A random decision tree framework for privacy-preserving data mining," *IEEE Trans. Dependable Secure Comput.*, vol. 11, no. 5, pp. 399–411, Sep./Oct. 2014.
- [7] W. Du and Z. Zhan, "Building decision tree classifier on private data," in *Proc. IEEE Int. Conf. Privacy, Secur. Data Mining*, 2002, pp. 1–8.
- [8] Y. Wu, S. Cai, X. Xiao, G. Chen, and B. C. Ooi, "Privacy preserving vertical federated learning for tree-based models," in *Proc. VLDB Endowment*, vol. 13, no. 11, pp. 2090–2103, 2020.
- [9] 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.
- [10] K. Wang, Y. Xu, R. She, and P. S. Yu, "Classification spanning private databases," in *Proc. 21st Nat. Conf. Artif. Intell.*, 2006, pp. 293–298.
- [11] K. Cheng et al., "SecureBoost: A lossless federated learning framework," *IEEE Intell. Syst.*, vol. 36, no. 6, pp. 87–98, Nov./Dec. 2021.
- [12] Y. Lindell and B. Pinkas, "Privacy preserving data mining," in *Proc. 20th Annu. Int. Cryptol. Conf. Santa Adv. Cryptol.*, Barbara, California, USA, 2000, pp. 36–54.
- [13] H. Kikuchi, K. Itoh, M. Ushida, H. Tsuda, and Y. Yamaoka, "Privacy-preserving decision tree learning with boolean target class," *IEICE Trans. Fund. Electron., Commun. Comput. Sci.*, vol. 98, no. 11, pp. 2291–2300, 2015.
- [14] Y. Zheng, S. Xu, S. Wang, Y. Gao, and Z. Hua, "Privet: A privacy-preserving vertical federated learning service for gradient boosted decision tables," *IEEE Trans. Serv. Comput.*, vol. 16, no. 5, pp. 3604–3620, Sep./Oct. 2023.
- [15] L. Liu, R. Chen, X. Liu, J. Su, and L. Qiao, "Towards practical privacy-preserving decision tree training and evaluation in the cloud," *IEEE Trans. Inf. Forensics Secur.*, vol. 15, pp. 2914–2929, 2020.
- [16] A. Patra, T. Schneider, A. Suresh, and H. Yalame, "Aby2.0: Improved mixed-protocol secure two-party computation," in *Proc. USENIX Secur. Symp.*, 2021, pp. 2165–2182.
- [17] D. Demmler, T. Schneider, and M. Zohner, "Aby-a framework for efficient mixed-protocol secure two-party computation," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2015, pp. 1–15.
- [18] F. Kerschbaum, E.-O. Blass, and R. A. Mahdavi, "Faster secure comparisons with offline phase for efficient private set intersection," 2022, *arXiv:2209.13913*.
- [19] H.-Y. Lin and W.-G. Tzeng, "An efficient solution to the millionaires' problem based on homomorphic encryption," in *Appl. Cryptogr. Netw. Secur.: Third Int. Conf.*, New York, NY, USA, 2005, pp. 456–466.
- [20] M. D. Ercegovic, L. Imbert, D. W. Matula, J.-M. Muller, and G. Wei, "Improving goldschmidt division, square root, and square root reciprocal," *IEEE Trans. Comput.*, vol. 49, no. 7, pp. 759–763, Jul. 2000.
- [21] K. Wang, B. C. Fung, and G. Dong, "Integrating private databases for data analysis," in *Proc. Intell. Secur. Informat.: IEEE Int. Conf. Intell. Secur. Inform.*, Atlanta, GA, USA, 2005, pp. 171–182.
- [22] A. C. Yao, "Protocols for secure computations," in *Proc. 23rd IEEE Annu. Symp. Found. Comput. Sci.*, 1982, pp. 160–164.
- [23] V. Kolesnikov, A.-R. Sadeghi, and T. Schneider, "Improved garbled circuit building blocks and applications to auctions and computing minima," in *Proc. Cryptol. Netw. Secur.: 8th Int. Conf.*, Kanazawa, Japan, 2009, pp. 1–20.
- [24] Y. Huang, D. Evans, and J. Katz, "Private set intersection: Are garbled circuits better than custom protocols?," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2012, pp. 1–15.
- [25] E. M. Songhori, S. U. Hussain, A.-R. Sadeghi, T. Schneider, and F. Koushanfar, "Tinygarble: Highly compressed and scalable sequential garbled circuits," in *Proc. IEEE Symp. Secur. Privacy*, 2015, pp. 411–428.
- [26] T. Schneider and M. Zohner, "Gmw vs. yao? efficient secure two-party computation with low depth circuits," in *Proc. Financial Cryptogr. Data Security: 17th Int. Conf.*, Okinawa, Japan, 2013, pp. 275–292.
- [27] X. Liu, Y. Zheng, X. Yuan, and X. Yi, "Securely outsourcing neural network inference to the cloud with lightweight techniques," *IEEE Trans. Dependable Secure Comput.*, vol. 20, no. 1, pp. 620–636, Jan./Feb. 2023.

- [28] I. Damgård, M. Fitz, E. Kiltz, J. B. Nielsen, and T. Toft, “Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation,” in *Proc. Theory Cryptogr. Conf.*, 2006, pp. 285–304.
- [29] T. Veugen, “Encrypted integer division and secure comparison,” *Int. J. Appl. Cryptogr.*, vol. 3, no. 2, pp. 166–180, 2014.
- [30] D. Bogdanov, M. Niitsoo, T. Toft, and J. Willemsen, “High-performance secure multi-party computation for data mining applications,” *Int. J. Inf. Secur.*, vol. 11, pp. 403–418, 2012.
- [31] P. Mohassel, M. Rosulek, and N. Trieu, “Practical privacy-preserving k-means clustering,” in *Proc. Privacy Enhancing Technol.*, vol. 4, pp. 414–433, 2020.
- [32] W.-Y. Loh, “Classification and regression trees,” *Wiley Interdiscipl. Rev.: Data Mining Knowl. Discov.*, vol. 1, no. 1, pp. 14–23, 2011.
- [33] B. Knott, S. Venkataraman, A. Hannun, S. Sengupta, M. Ibrahim, and L. van der Maaten, “Crypten: Secure multi-party computation meets machine learning,” in *Proc. Adv. Neural Inf. Process. Syst.*, 2021, pp. 4961–4973.
- [34] P. Mohassel and Y. Zhang, “Secureml: A system for scalable privacy-preserving machine learning,” in *Proc. IEEE Symp. Secur. Privacy*, 2017, pp. 19–38.
- [35] M. Keller, “MP-SPDZ: A versatile framework for multi-party computation,” in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2020, pp. 1575–1590.
- [36] J. Fu, K. Cheng, Y. Xia, A. Song, Q. Li, and Y. Shen, “Private decision tree evaluation with malicious security via function secret sharing,” in *Proc. Eur. Symp. Res. Comput. Secur.*, 2024, pp. 310–330.
- [37] A. Song, J. Fu, X. Mu, X. Zhu, and K. Cheng, “L-SecNet: Towards secure and lightweight deep neural network inference,” *J. Netw. Netw. Appl.*, vol. 3, no. 4, pp. 171–181, 2024.
- [38] Y. Lindell, “How to simulate it—a tutorial on the simulation proof technique,” *Tut. Found. Cryptogr.: Dedicated Oded Goldreich*, vol. 1, pp. 277–346, 2017.
- [39] Z. Xia, Q. Gu, W. Zhou, L. Xiong, J. Weng, and N. Xiong, “STR: Secure computation on additive shares using the share-transform-reveal strategy,” *IEEE Trans. Comput.*, vol. 73, no. 2, pp. 340–352, Feb. 2024.
- [40] P. Mohassel and P. Rindal, “ABY3: A mixed protocol framework for machine learning,” in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2018, pp. 35–52.
- [41] Y. Dong, C. Xiaojun, W. Jing, L. Kaiyun, and W. Wang, “Meteor: Improved secure 3-party neural network inference with reducing online communication costs,” in *Proc. ACM Web Conf.*, 2023, pp. 2087–2098.
- [42] O. Catrina and A. Saxena, “Secure computation with fixed-point numbers,” in *Proc. Financial Cryptogr. Data Secur.: 14th Int. Conf.*, Tenerife, Canary Islands, 2010, pp. 35–50.
- [43] S. Wagh, S. Tople, F. Benhamouda, E. Kushilevitz, P. Mittal, and T. Rabin, “Falcon: Honest-majority maliciously secure framework for private deep learning,” 2020, *arXiv: 2004.02229*.
- [44] L. Song, Z. Wang, G. Lin, and W. Han, “Ruyi: A configurable and efficient secure multi-party learning framework with privileged parties,” *IEEE Trans. Inf. Forensics Secur.*, vol. 19, pp. 10355–10370, 2024.
- [45] D. Escudero and S. Fehr, “On fully-secure honest majority MPC without round overhead,” in *Proc. Int. Conf. Cryptol. Inf. Secur. Latin Amer.*, 2023, pp. 47–66.
- [46] R. Canetti, “Security and composition of multiparty cryptographic protocols,” *J. Cryptol.*, vol. 13, pp. 143–202, 2000.
- [47] K. Huang, X. Liu, S. Fu, D. Guo, and M. Xu, “A lightweight privacy-preserving CNN feature extraction framework for mobile sensing,” *IEEE Trans. Dependable Secure Comput.*, vol. 18, no. 3, pp. 1441–1455, Mar. 2021.



Anxiao Song received the BE degree in software engineering from the Henan University of Technology, Henan, China, in 2019. He is currently working toward the PhD degree with the School of Computer Science and Technology, Xidian University. His research interests include machine learning security and privacy protection.



Ke Cheng received the PhD degree in computer science and technology from Xidian University, Xi’an, Shaanxi, China, in 2022. He is an associate professor in the School of Computer Science and Technology with Xidian University. He has more than 30 papers published in reputed journals and major international conferences, such as INFOCOM, ESORICS, *IEEE Transactions on Computers*, *IEEE Transactions on Information Forensics and Security*, and *IEEE Transactions on Dependable and Secure Computing*. His research interests include data security and privacy protection.



Jiaxuan Fu received the BS degree from Xidian University, China, in 2019. He is currently working toward the PhD degree with the School of Computer Science and Technology, Xidian University, China. His research interests include IoT data security and machine learning security.



Shujie Cui received the PhD degree from the University of Auckland, in 2019. She is a lecturer with Monash University in the Faculty of Information Technology. Her main research interests include applied cryptography, data security in various systems, trusted execution environments, side-channel attacks, and privacy-preserving machine learning.



Tao Zhang received the MS and PhD degrees in computer science from Xidian University, Xi’an, China, in 2011 and 2015, respectively. He is an associate professor with the School of Computer Science and Technology, Xidian University. His research interests include data security and privacy protection.



Zhao Chang (Member, IEEE) received the BS degree in computer science and technology from Peking University, in 2013, and the PhD degree in computing from University of Utah, in 2021. He currently works as an associate professor in the School of Computer Science and Technology, Xidian University. His research interests focus on security and privacy issues in large-scale data management.



Yulong Shen received the BS and MS degrees in computer science and the PhD degree in cryptography from Xidian University, Xi’an, China, in 2002, 2005, and 2008, respectively. He is currently a professor with the School of Computer Science and Technology, Xidian University. His research interests include wireless network security and cloud computing security. He has served on the technical program committees of several international conferences, including ICEBE, INCoS, CIS, and SOWN.