FSS-DT: Secure Division-free Decision Tree Training and Inference via Function Secret Sharing

Anxiao Song, Shujie Cui, Ke Cheng, Jianli Bai, Qifan Wang, Shangqi Lai, Giovanni Russello, Yulong Shen

***Abstract*—Decision tree (DT), an efficient and interpretable machine learning algorithm, has been widely used in numerous industrial applications. As the demand for higher performance grows, there is increasing interest in cross-domain collaboration for training DTs using vertical datasets, where different owners hold distinct features but share the same set of samples. Due to privacy concerns, leveraging secure multi-party computation (MPC) to train DTs jointly has gained significant attention. Although promising, deploying these approaches across large-scale vertical datasets presents formidable challenges for two bottlenecks: divisions and comparisons. MPC is not friendly to process divisions, whereas DT training algorithms involve division to evaluate each feature split for each tree node. DT training and inference also involve numerous comparisons, yet existing secure comparisons are still not efficient enough.**

We propose FSS-DT, a highly efficient two-party framework for secure DT training and inference on vertically distributed datasets. The efficiency of FSS-DT is achieved by designing a division-free MPC-friendly DT algorithm. Moreover, we present a lightweight secure comparison protocol using our optimized distributed comparison function (DCF), an instance of function secret sharing, via "0/1-encoding" and "early termination" techniques. We have implemented a prototype of FSS-DT. Extensive experiments demonstrate that FSS-DT achieves the same accuracy as models trained in plaintext and outperforms the state-of-the-art Pivot and PriVDT. Specifically, for training on LAN and WAN over the large UCI dataset, FSS-DT is $17.5\times$ and $51.1\times$ faster than Pivot, and about $8.4\times$ and $8.8\times$ faster than Swan, respectively.

***Index Terms*—Secure Division-free Decision Tree, Distributed Comparison Function, Vertical Datasets**

This paper is supported by the National Natural Science Foundation of China (No. 62402358, No. 62220106004, No. 62502363, No. 62572371), the National Key R&D Program of China (No. 2023YFB3107500), the Young Elite Scientist Sponsorship program by CAST (No. YESS20240717), the Young Talent Fund of Association for Science and Technology in Shaanxi, China (No. 20240138), the "Pioneer" and "Leading Goose" R&D Program of Zhejiang (No. 2026SDXT014), the Key Research and Development Program of Shaanxi (Program No. 2025PT-ZCK-52), the Open Topics from the Lion Rock Labs of Cyberspace Security (under the project #LRL24004), the Fundamental Research Funds for the Central Universities (No. ZDRC2202, No. QTZX26022, No. QTZX26041, No. QTZX26050, No. QTZX26118), the Xidian University Specially Funded Project for Interdisciplinary Exploration (No. TZJHF202502), and the China Scholarship Council (CSC). (The corresponding author: Ke Cheng)

Anxiao Song, Ke Cheng, 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, Shaanxi 710071, China. (e-mail: songanxiao@stu.xidian.edu.cn, chengke@xidian.edu.cn, and ylshen@mail.xidian.edu.cn).

Shujie Cui works at the Faculty of Information Technology of Monash University, Australia. (e-mail: shujie.cui@monash.edu); Jianli Bai works at Singapore Management University, Singapore. (e-mail: baijianli0812@ gmail.com); Shangqi Lai works at CSIRO's Data61, Australia.(e-mail:shangqi.lai@csiro.au); Qifan Wang works at Durham University, Durham, United Kingdom. (e-mail: qifan.wang2@durham.ac.uk); Giovanni Russello works in the Faculty of Science of Auckland University, New Zealand. (e-mail: g.russello@auckland.ac.nz).

I. Introduction

Decision tree (DT) algorithms (e.g., CART [1]) are popular machine learning models and have been widely used in many industrial domains for their high performance and strong interpretability [2], [3]. To enhance performance, there is a growing interest in training DT models with cross-domain owners that hold vertical datasets, each contributing to different features of the same samples. For instance, an insurance company can seek a DT model to enhance personalized health insurance product recommendations for its clients through collaboration with a hospital. The insurance company holds detailed financial information, such as insurance coverage, premium payments, and claims history, while the hospital possesses comprehensive patient data, including medical history, treatment plans, and health conditions. An effective recommendation model would integrate these datasets to offer clients personalized health insurance plans tailored to their medical needs and financial situation. However, this ambition faces significant hurdles due to privacy concerns and strict regulations such as GDPR.

This work focuses on training vertical DT models for cross-domain owners. Secure multi-party computation [4] (MPC) is a powerful tool that enables multiple data owners to jointly compute a function in a private fashion. In recent years, a series of works [5]–[9] have used MPC to train privacy-preserving vertical DT models for vertically distributed datasets. In vertical DT, data is column-distributed, and each party holds distinct features such that feature resorting is unnecessary. The split information for a tree node can be revealed to feature owners, allowing each party to perform certain operations (e.g., threshold comparison). In this context, the primary focus of vertical DT is on efficiently determining the best feature split at each tree node. In addition, the training DT protocols [10]–[13] with share-input and share-output can also be considered as a generalization of vertical DT. Existing MPC-based frameworks can effectively protect data privacy, yet they are still impractical due to the following two issues.

MPC-unfriendly plaintext algorithms. Existing DT algorithms must rely on division operations to evaluate which (feature, threshold) pair is the best for a tree node, which are inefficient in MPC setting. As far as we know, no DT algorithm in the plaintext domain is division-free. Existing MPC-based works adopt two main approaches to handle or eliminate division operations. The first employs iterative approximation methods (e.g., Newton's [8], [16] or Goldschmidt's methods [9], [17]) to implement secure divisions. They must in-

TABLE I: Comparison with existing secure DT approaches for training vertical datasets

Approaches	Crypto tools	Comm. round	Computation cost	Comm. cost	Leakage	No accuracy loss
Du *et al.* [5]	ASS	$\mathcal{O}(m\mathcal{T})$	$\mathcal{O}(m\mathcal{T}K)$	$\mathcal{O}(m\mathcal{T}K\ell)$	Label, intermediates	✓
Vaidya *et al.* [7]	AHE	$\mathcal{O}(m\mathcal{T})$	$\mathcal{O}(m)(m\mathcal{T}KC_1)$	$\mathcal{O}(m\mathcal{T}K\ell_1))$	Intermediates	✓
Lindell *et al.* [6]	ASS, GC	$\mathcal{O}(m\mathcal{T}(\log\kappa_i+\ell^2)$	$\mathcal{O}(m\mathcal{T}\log\kappa_i C_g)$	$\mathcal{O}(m\mathcal{T}K\ell^2\log(\kappa_i\ell))$	–	×
Pivot [8]	ASS, AHE	$\mathcal{O}(m\mathcal{T}(\log\ell+\ell^2))$	$\mathcal{O}(m\mathcal{T}K(C_a+C_d))$	$\mathcal{O}(m\mathcal{T}K(\ell^2+\log\ell+\ell))$	–	×
PriVDT [9]	ASS, FSS	$\mathcal{O}(m\mathcal{T}\ell^2)$	$\mathcal{O}(m\mathcal{T}K(C_f+C_d))$	$\mathcal{O}(m\mathcal{T}K(\ell^2+\ell))$	–	×
Swan [14]	improved ASS	$\mathcal{O}(\log m\mathcal{T}+logF)$	$\mathcal{O}(m\mathcal{T}K(C_f+C_d))$	$\mathcal{O}(m\mathcal{T}K\ell+2mK\mathcal{T}(\ell\log^2\ell+\ell^2))$	–	✓
FSS-DT	ASS, FSS	$\mathcal{O}(m\mathcal{T})$	$\mathcal{O}(m\mathcal{T}KC_f)$	$\mathcal{O}(m\mathcal{T}K\ell)$	–	✓

These legal frameworks make it impossible to freely share sensitive information in plaintext across organizations for collaborative DT training. The complexity in the table is for one data sample. ℓ, ℓ_1 are the bit-lengths of secret sharing and homomorphic encryption (HE), respectively; m, $\mathcal{T}$, and K are the number of features, thresholds, and classes, respectively; C_1 is the computation costs of encryption in HE; C_g, C_r, C_D, and C_f are the computation cost of garbled circuits (GC), RAC-based comparison, secure division, and DCF-based comparison, respectively; κ_i is the degree of approximate polynomials for $x\text{ln}(x)$ in [15]; '✓' indicates that the requirement is satisfied; '×' indicates that the requirement is not satisfied; '−' represents no leakage of data privacy.

voke multiple secure bit decompositions, secure comparisons, and secure fixed-point multiplications with secure truncation, which have heavy computation and communication overhead. The second approach [12], [13], [18] transforms division into comparison on polynomials that severely limit sample capacity due to the growth in multiplicative depth and approximation errors. While the former suffers from prohibitive communication overhead, particularly for deep trees, the latter is unsuitable for large-scale datasets due to its combinatorial solution space and accuracy degradation in MPC.

Inefficient secure comparison operations. Secure DT algorithms also invoke many secure comparison operations, which fall into three categories: comparison schemes based on garbled circuit gates [19] (GC), ripple-carry adders [20] (RCA) with secret sharing, and distributed comparison function [21] (DCF). GC-based schemes must generate garbled tables consisting of multiple gate-level circuits and evaluate them, and RCA-based schemes rely on heavy secure bit decomposition and secure bit-wise evaluation. Both methods have high communication overhead. In contrast, the DCF-based comparison requires fewer communication rounds and lower computation costs. Traditional DCF-based comparisons [21]–[23] require long RNG keys and additional operations to solve wrap-around issues. To improve DCF's efficiency, Grotto [24] reduces key length via a parity-segment tree but introduces additional computation during DCF evaluation due to tree traversal.

In light of the above, we present FSS-DT, an FSS-based two-party framework for secure DT training and inference on vertically distributed datasets. FSS-DT leverages ASS and FSS to protect data, which have lower communication and simpler computations than AHE, FHE, and GC. To address the above two challenges, we design a division-free DT training algorithm and a more lightweight secure comparison protocol. In Table I, we compare FSS-DT with existing secure vertical DT frameworks. Among vertical DT works [5]–[9], Pivot [8] and PriVDT [9] stand out for their performance and accuracy. Our FSS-DT achieves less communication and computation costs and better accuracy than Pivot and PriVDT. Du *et al.*'s [5] and Vaidya *et al.*'s [25] works have comparable overhead with ours and also have no accuracy loss compared with the plaintext model. However, they leak intermediates such as data distribution on each node. Our contributions and techniques can be summarized as follows:

- **MPC-friendly DT training without divisions.** We design a division-free DT training algorithm that enables FSS-DT to avoid the expensive secure truncations required for fixed-point multiplication thus improving performance while maintaining the model's accuracy. FSS-DT achieves comparable accuracy with the state-of-the-art DT algorithms (e.g., ID3, C4.5, and CART). Precisely, we propose a division-free splitting criterion to evaluate which (feature, threshold) pair is the best for a tree node.
- **Lightweight secure comparison protocol.** We propose a lightweight secure comparison protocol featuring an optimized distributed comparison function (DCF) based on "0/1-encoding" and "early termination optimization". Compared to the Grotto scheme [24], the best prior DCF approach [21], [26]–[28], our method maintains the same key size while reducing evaluation runtime from $\mathcal{O}((\ell-\log\lambda)\log\lambda)$ to $\mathcal{O}(\ell-\log\lambda)$. Additionally, leveraging ripple-carry adder techniques, our protocol requires only a single DCF call for secure comparison. This approach reduces runtime by approximately 67% compared to previous DCF-based schemes when processing inputs of size 10^6 on the WAN network.
- **Secure efficient DT training and inference.** With the lightweight secure comparison, our MPC-friendly DT algorithm, and ASS, we design secure two-party DT training and inference protocols on vertically distributed datasets. They ensure both data privacy and model accuracy. The parties do not disclose any inputs or intermediate data to each other in FSS-DT.
- **Extensive evaluations.** We implement a prototype of FSS-DT and extensively evaluate its performance on various real-world datasets. The experimental results show that FSS-DT outperforms the state-of-the-art works in terms of accuracy and efficiency. Specifically, for training on LAN and WAN over the large UCI dataset, FSS-DT is $17.5\times$ and $51.1\times$ faster than Pivot, and about $8.4\times$ and $8.8\times$ faster than Swan, respectively.

Organization. The rest of this paper is organized as follows. We introduce the related works and preliminaries in Section II and Section III, respectively. Next, we give an overview of our system in Section IV. Our MPC-friendly DT and secure comparison are presented in Section V and Section VI, respectively. The construction is shown 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. Secure Decision Trees

In recent years, a series of works [5]–[9] have used MPC to train privacy-preserving vertical DT models. In the vertical setting, each party holds the same samples but different features, and it is common when participants are business organizations. The vertical DT is a column-wise algorithm. For a single tree node on vertical DT, the split information can be visible to the party that owns the corresponding feature, while remaining non-visible to the other party. This allows each party to work with their data in unencrypted form for certain operations, such as discretizing features and comparing their own features with split thresholds. In this context, the main challenge is securely determining the best feature split at each node. This involves complex computations and many comparisons, requiring significant effort from all parties to ensure privacy and efficiency during training. Du *et al.* [5] proposed a secure protocol using ASS to train a decision tree model on vertical datasets, aiming to enhance computational efficiency but potentially exposing sensitive data like labels and split information. Vaidya *et al.* [7] employed additive homomorphic encryption (AHE) for secure random decision trees among multiple parties, yet sharing intermediate values allows adversaries to potentially infer sensitive information. To promise perfect security, Lindell *et al.* [6] introduced the first secure DT scheme through oblivious transfer (OT) and garbled circuits (GC), albeit with impractical computational costs. Recent works have focused on new cryptographic primitives to reduce communication costs. Wu *et al.* [8] proposed Pivot, utilizing HE and ASS, while Chen *et al.* [9] developed PriVDT with FSS and ASS. However, Pivot requires time-consuming conversion of HE ciphertexts into ASS values and secure comparison operation of high-frequency communication. To improve the efficiency of Pivot, PriVDT introduces FSS primitives for online communication, but it only invokes the original DCF, ignoring wrap-around issues (in Section VI-C) that result in accuracy loss.

The training DT protocols with share-input and share-output [10]–[13], [29] can also be viewed as a generalization of vertical DT. In these protocols, P_0 and P_1 share their secret-shared inputs and execute the share-input–share-output DT training protocol to obtain a secret-shared DT. Finally, the corresponding feature owner partially reconstructs the secret-shared DT to obtain the output for the vertical DT. Recently, Adams *et al*. [29] successfully extends and implements a secure random forest based on the above two works over horizontal datasets. However, those works all require heavy computation overhead. Besides, Bhardwaj *et al.* [13] and Lin *et al.* [12] introduce the secure radix sort protocols and group permutation protocol to optimize the complexity of feature sort. However, those works require multiple interactions and heavy communication overhead.

In contrast to prior works, FSS-DT optimizes the existing DCF protocol through 0/1-encoding and early termination techniques, achieving a lightweight comparison protocol. Furthermore, we construct a secure and efficient training protocol based on our proposed division-free DT training algorithm, which significantly reduces both communication and computational overhead.

TABLE II: Notations

Notations	Definitions
$\langle x\rangle = (\langle x\rangle_0, \langle x\rangle_1)$	(2,2)-Arithmetic sharing of x
$[\![x]\!] = ([\![x]\!]_0, [\![x]\!]_1)$	(2,2)-Boolean sharing of x
$\mathbf{Y} = \{Y_0, .., Y_{K-1}\}$	Label set
$\mathbf{S}_i = (S_{i,0}, ..., S_{i,m-1}, y_i)$	The i-th sample with m features; y_i is the label
$\mathbf{F}_j = (S_{0,j}, ..., S_{N-1,j})$	The j-th feature vector
$\mathcal{D} = \{\mathbf{S}_0, ..., \mathbf{S}_{N-1}\}$	Complete dataset with N samples
$\mathcal{D}_0$	N samples with the first m_0 features
$\mathcal{D}_1$	N samples with the last $m - m_0$ features and the label
$T.root$	Tree root node
$T[i].l$, $T[i].r$, $T[i].leaf$	Left child, right child, and leaf label of $T[i]$
$T[i].f$, $T[i].\tau$	Best feature and threshold assigned to $T[i]$

B. Secure Comparison Protocols

Secure comparison protocols are a critical component for secure DTs. This significance dates back to Yao's seminal work [30], which introduced the "Two Millionaires' Problem" for comparing integers using garbled circuits. Recent advances have shifted the focus from garbled circuits to more efficient domains, such as secret sharing. Zheng *et al.* [20] transformed the problem of secure comparison into a task of extracting the most significant bit (MSB) of $x - y$. However, it requires $\log \ell$ rounds and $12\ell - 2$ bits of communication. To enhance performance, Rathee *et al.* [31] propose a secure comparison protocol with an OT-based lookup table, which can reduce communication rounds but with additional computation burdens. A significant breakthrough was achieved by Boyle *et al.* [26], who introduced the first FSS that requires one round of communication in the online phase. FSS stands out with its minimal communication overhead, forming the foundation of distributed point function (DPF) and distributed comparison function (DCF) schemes. Subsequent advancements introduce half-tree-based DCF [32], [33] and GGM-tree-based DCF [28], [34]. However, their work requires long RNG keys and additional operations to solve wrap-around issues during the online evaluation. To improve performance, the Grotto [24] scheme integrates the parity-segment tree to generate shorter RNG keys. However, they require traversing the parity-segment tree at each level of DCF, leading to additional computation costs.

III. Preliminaries

In this section, we review additive secret sharing, function secret sharing, and 0/1-encoding. Besides, the key notations utilized throughout our work are listed in Table II.

A. Additive Secret Sharing

FSS-DT employs Arithmetic and Boolean sharing within a 2-out-of-2 additive secret sharing scheme, chosen for its efficiency and security in data sharing.

Arithmetic sharing. For a given ℓ-bit value $x \in \mathbb{Z}_{2^\ell}$, its arithmetic sharing is denoted as $\langle x\rangle = (\langle x\rangle_0, \langle x\rangle_1)$, where $\langle x\rangle_b$ is held by party P_b and $x = (\langle x\rangle_0 + \langle x\rangle_1) \mod 2^\ell$, where $b \in \{0, 1\}$. Without special declaration, we omit (mod 2^ℓ) for

brevity in the rest of the paper. We have the following secure primitives:

- $x = \textbf{Open}(\langle x \rangle)$: Given $\langle x \rangle$, P_b gets $x = \langle x \rangle_0 + \langle x \rangle_1$.
- $\langle x \rangle = \textbf{Share}(x)$: Given x, P_b gets $\langle x \rangle_b$.
- $\langle z \rangle = \langle x \rangle + \langle y \rangle$: Given $\langle x \rangle$ and $\langle y \rangle$, P_b can locally compute the secret sharing of $z = x + y$ as $\langle z \rangle_b = \langle x \rangle_b + \langle y \rangle_b$.
- $\langle z \rangle = \langle x \rangle + c$: Given $\langle x \rangle$ and a constant c, P_b can compute the secret sharing of $z = x + c$ as $\langle z \rangle_b = \langle x \rangle_b + b \cdot c$.
- $\langle z \rangle = \langle x \rangle \cdot \langle y \rangle$: Given $\langle x \rangle$ and $\langle y \rangle$, the secret sharing of $z = x \cdot y$ can be computed using Beaver multiplication triplet technique [20], [35], i.e., $\langle c \rangle = \langle u \rangle \cdot \langle v \rangle$ are pre-generated by P_0 and P_1. First, each party P_b locally computes $\langle e \rangle_b = \langle x \rangle_b - \langle u \rangle_b$ and $\langle f \rangle_b = \langle y \rangle_b - \langle v \rangle_b$. Next, P_0 and P_1 run $e = \textbf{Open}(\langle e \rangle)$ and $f = \textbf{Open}(\langle f \rangle)$. Finally, each party P_b computes $\langle z \rangle_b = b \cdot e \cdot f + f \cdot \langle x \rangle_b + e \cdot \langle y \rangle_b + \langle c \rangle_b$. It requires 1 round and 2ℓ bits of communication to exchange shares of e and f.
- $\langle z \rangle = c \cdot \langle x \rangle$: Given $\langle x \rangle$ and a constant c, P_b gets the secret sharing of $z = c \cdot x$ by locally computing $\langle z \rangle_b = c \cdot \langle x \rangle_b$.

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

Secure B2A gadget. The secure B2A gadget converts a boolean-sharing data $[\![x]\!]$ into its arithmetic-sharing data $\langle x \rangle$, i.e., $\langle x \rangle = \textbf{B2A}([\![x]\!])$. In this process, 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 $\langle v \rangle_1 = [\![x]\!]_1$. Next, P_0 and P_1 compute $\langle x \rangle = \langle u \rangle + \langle v \rangle - 2\langle u \rangle \cdot \langle v \rangle$. Amongst, the multiplication $\langle u \rangle \cdot \langle v \rangle$ requires 1 round of communication and 2ℓ bits of communication cost.

B. Function Secret Sharing

FSS is a lightweight cryptographic primitive, which requires fewer communication rounds and lower computation costs for handling non-linear functions. FSS allows a function $\mathcal{F}$ to be split into two function shares, $\mathcal{F}_0$ and $\mathcal{F}_1$. Each party receives one of these shares, and neither share reveals any private information about the original function $\mathcal{F}$. To test if $x < \alpha$ between two parties, FSS provides a secure scheme, i.e., Distributed Comparison Function (DCF). The formal definition of DCF is as follows:

Definition 1 (DCF)**.** *A comparison function, denoted as $\mathcal{F}^{<}_{\alpha,1}$, with a public input x, outputs 1 if $x < \alpha$ and 0 otherwise. We refer to an FSS scheme for $\mathcal{F}^{<}_{\alpha,1}$ as DCF, which includes a pair of algorithms: **Gen**(λ, α) for generating key pairs of $\mathcal{F}^{<}_{\alpha,1}$, and **Eval**($\mathcal{K}_b$, x) for evaluating the comparison without disclosing α.*

- ***Gen***$(1^\lambda, \alpha) \to (\mathcal{K}_0, \mathcal{K}_1)$*: Given a function description of $\mathcal{F}^{<}_{\alpha,1}$ and a security parameter 1^λ, it generates two secret keys $\mathcal{K}_0$ and $\mathcal{K}_1$.*
- ***Eval***$(\mathcal{K}_b, x) \to \beta_b$*: Given the FSS key $\mathcal{K}_b$ for $b \in \{0, 1\}$ and a public value x, each party P_b can output a boolean bit β_b of the function $\mathcal{F}^{<}_{\alpha,1}(x)$, where $\beta_0 \oplus \beta_1 = 1$ if $x < \alpha$ and 0 otherwise.*

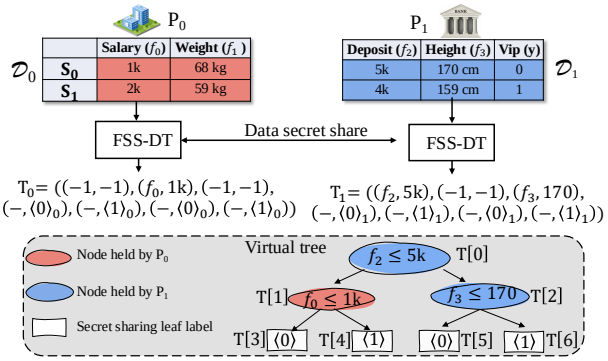


Fig. 1: FSS-DT overview. P_0 and P_1 run FSS-DT on their joint dataset $\mathcal{D} = \mathcal{D}_0 \| \mathcal{D}_1$ and obtain a partial of the trained model that represented as an array. The partial tree T_b only contains the features owned by P_b and the secret share of all leaf labels. Indeed, $|T_b| = |T|$, and non-leaf node $T_b[i] = (f_*, \tau_*)$ if the f_*-th feature belongs to P_b, otherwise, $T_b[i] = (-1, -1)$.

Definition 3 establishes the security and correctness of the FSS scheme, as detailed in Appendix G.

C. 0/1-Encoding

Our scheme uses the 0/1-encoding [36] to solve the comparison problem between two values. **This method can cleverly transform a comparison problem into an equality test problem, allowing us to construct a lightweight DCF.** Let an ℓ-length binary string of x be $x_0 x_1 ... x_{\ell-1}$ where $x_{\ell-1}$ is the least significant bit. The 0-encoding and 1-encoding of x are defined as:

$$Set_0^x = \{x_0 x_1 ... x_{i-1} 1 \mid x_i = 0, 0 \le i \le \ell - 1\}$$
$$Set_1^x = \{x_0 x_1 ... x_{i-1} x_i \mid x_i = 1, 0 \le i \le \ell - 1\}$$

Assume we guess $z > y$. To check if the guess is correct, we encode the hypothetically larger and smaller values with 1-encoding and 0-encoding with the same bit length, i.e., encode z and y into Set_1^z and Set_0^y, respectively, and then check if $|Set_1^z \cap Set_0^y| = 1$. If yes, $z > y$ is true. Otherwise, $|Set_1^z \cap Set_0^y| = 0$ and $z \le y$. Conversely, if we guess $y > z$, we encode y and z into Set_1^y and Set_0^z, respectively. For example, let $z = 10 = 1010_2$ and $y = 3 = 0011_2$ of length $\ell = 4$. To check if $z > y$, we will get $Set_1^z = \{1, 101\}$ and $Set_0^y = \{1, 01\}$. Since $|Set_1^z \cap Set_0^y| = 1$, it is true that $z > y$. Lin *et al.* have proven the correctness of this method in [36].

IV. Overview

A. Overview of FSS-DT

As shown in Fig. 1, our FSS-DT framework consists of two parties P_0 and P_1. In particular, P_0 and P_1 own a different feature space for the same samples. Let a joint dataset with N samples and m features be $\mathcal{D} = \{\mathcal{D}_0, \mathcal{D}_1\}$, where P_b owns $\mathcal{D}_b$ for $b = \{0, 1\}$. Their features are vertically distributed between the two parties. 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 [37], effectively creating a combined dataset $\mathcal{D} = \mathcal{D}_0 \| \mathcal{D}_1$. For clarity, we

designate P_1 as the holder of the whole label vector $\mathbf{y}$. FSS-DT focuses on categorical data. For continuous features, as done in the majority of existing works [8], [9], [17], each party employs the equal-width binning (EWB) method from [29] to discretize them into the numeric bins locally.

P_0 and P_1 run our two-party FSS-DT protocol to train a DT model on their joint dataset. Traditional DT training (shown in Algorithm 5 in Section V-A) mainly involves two processes: calculating the splitting criterion and partitioning $\mathcal{D}$ based on the assigned node, and both of them involve comparison. We design a division-free splitting criterion g' and a lightweight secure comparison protocol for finding the best split. During the training, the best feature and threshold assigned to a node are revealed to the party holding them. Thus, P_b can partition $\mathcal{D}_b$ locally and then share it secretly with P_{1-b} for training the next node. Without a global view, the two parties have to get the best split jointly with our proposed lightweight secure comparison protocol. After training, P_b gets a partial of the DT model T_b, where T_0 and T_1 compose the full tree. T_b only contains the tree part involving P_b's features and all the leaf labels' secret share. For inference, each party P_b inputs his own local data and partial model T_b to produce secret-shared inference results without leaking their data and partial model.

B. Threat Model

Similar to previous works [8], [9], we follow a standard simulation paradigm against a static and semi-honest probabilistic polynomial time (PPT) adversary. In particular, the adversary can only compromise either P_0 or P_1, and it will follow the protocol as specified and try to learn additional information by analyzing the exchanged messages during the execution.

Definition 2 (Security)**.** *Let Π be a two-party protocol for computing a function $\mathcal{F} : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^*$, where $\mathcal{F} = (\mathcal{F}_0, \mathcal{F}_1)$. For input pair $(x, y) \in \{0,1\}^n$, the output pair is $(\mathcal{F}_0(x,y), \mathcal{F}_1(x,y))$. The view of P_b during an execution of Π on (x,y) is denoted by $view_b^{\Pi}(x,y)$. The output of P_b during an execution of Π on (x,y) is denoted by $\mathcal{O}_b^{\Pi}$. 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 $\mathcal{F}(x,y)$ if there exist PPT simulators Sim_0 and Sim_1 such that*

$$\{Sim_0(x, \mathcal{F}_0(x,y)), \mathcal{F}(x,y)\} \stackrel{c}{\equiv} \{view_0^{\Pi}(x,y), \mathcal{O}^{\Pi}(x,y)\} \quad (1)$$

$$\{Sim_1(y, \mathcal{F}_1(x,y)), \mathcal{F}(x,y)\} \stackrel{c}{\equiv} \{view_1^{\Pi}(x,y), \mathcal{O}^{\Pi}(x,y)\} \quad (2)$$

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

V. Our MPC-friendly Decision Tree

This section reviews the DT training procedure and illustrates our MPC-friendly DT (DFDT) algorithm, which will serve as the underlying algorithm for our FSS-DT.

A. Decision Tree

This work focuses on training complete binary DTs for classification tasks. In DTs with the maximum depth $\mathcal{H} \geq 1$, each internal (non-leaf) node signifies a feature with a split threshold, and each leaf node represents a predicted class label for the path taken through the tree. We use an array T to represent a DT and the index of each tree node iterates in $i \in \{0, \ldots, 2^{\mathcal{H}} - 2\}$, where $T[0]$ is the root node (also represented as $T.root$), $T[2i+1]$ and $T[2i+2]$ are the left and right children of $T[i]$, respectively. Each internal node $T[i]$ consists of 2 attributes (f, τ) that represent the feature index and threshold assigned to it, respectively. When $i \in \{2^{\mathcal{H}-1} - 1, \ldots, 2^{\mathcal{H}} - 2\}$, $T[i]$ represents leaf nodes, where it only records a trained leaf label. For clarity, we also use $T[i].r$ and $T[i].l$ to represent the right and left children.

Given a dataset with N samples and m features $\mathcal{D} = \{\mathbf{S}_0, \ldots, \mathbf{S}_{N-1}\}$, the tree T is constructed recursively in a top-down manner, starting from the root. Detailed steps are provided in Algorithm 5 of Appendix A. DT training aims to identify the optimal split (feature and threshold, e.g., $age > 18$) at each internal node based on specific criteria. All samples are used to train the root $T[0]$. Once the optimal split is determined, the data is divided into two subsets, $\mathcal{D}_l$ and $\mathcal{D}_r$, which are used to train the child nodes $T[0].l$ and $T[0].r$ separately. This process continues recursively until the predefined maximum depth $\mathcal{H}$ is reached.

B. Problem Statement

In secure DT training, most existing MPC-based approaches [8], [9], [11] use Gini index [1] as the split criterion due to its training of high-precision DT model. However, the Gini index also requires MPC-unfriendly divisions. Essentially, there are two ways to handle that. Assume the two split candidates to be compared are A and B, and $g(A) = a/b$ and $g(B) = c/d$. The first way [13] is to convert the comparison between a/b and c/d into checking if $a \cdot d - b \cdot c > 0$. This method can avoid divisions but imposes smaller restrictions on the modulus and the dataset size in the schemes using modulo 2^{ℓ}. For instance, if all features are binary, the numerator a and denominator b are upper bounded by N^3 and N^2, respectively, where N is the number of samples. Existing schemes [29] can process at most $2^{13} = 8,192$ samples for the modulus 2^{64} due to $5 \cdot (\log N - 1) = 64$.

Alternatively, some directly compute the MPC-based division of a/b and c/d, truncate them into fixed-point values, convert them into integers by multiplying them with a large value (should be experimentally adjusted), and then compare them. However, this approach introduces significant overhead. Additionally, insufficient precision in fixed-point arithmetic and improper truncation can lead to incorrect comparison results, especially for precision-sensitive metrics like the Gini index. For example, when $Gini(A) = 0.73589$ and $Gini(B) = 0.73511$, low precision or improper truncation might discard critical decimal places, leading to incorrect split selection and accuracy loss. To our knowledge, we are the first to address these issues by designing a division-free splitting criterion.

[1]For a given dataset $\mathcal{D}$ with K classes, the Gini Index on one tree node is calculated as $g(\mathcal{D}) = 1 - \sum_{k \in K} p_k^2$, where p_k is the probability of an object being classified to class k in the node.

C. Division-free Splitting Criterion

The splitting criteria are meant to maximize the purity of the partitioned data subsets. Information gain or Gini index evaluates that with information entropy or Gini index values involving division operations. We measure the purity by computing the differences in the two subsets without divisions. Formally, given a candidate (f, τ), where τ represents one of the possible thresholds for feature f, the input dataset $\mathcal{D}$ is partitioned into $\mathcal{D}_l$ and $\mathcal{D}_r$, containing samples with f-th feature values $< \tau$ and $\geq \tau$, respectively. Let $\mathcal{D}_l^k$ and $\mathcal{D}_r^k$ denote samples with label k in $\mathcal{D}_l$ and $\mathcal{D}_r$, respectively, where $k \in K$ and K is the label set. To maximize purity, the differences of samples within both $\mathcal{D}_l$ and $\mathcal{D}_r$ should be as small as possible. Inspired by the optimization of cross entropy [38], we compute the difference based on the number of samples with distinct labels. Taking $\mathcal{D}_l$ as the example, we compute the difference with $\sum_{k\in K} |\mathcal{D}_l^k|^2$. However, solely relying on it may lead to overfitting issues in the trained DT model, resulting in high variance during inference due to excessive branching. To mitigate this, we define a scaling factor to balance the importance of the difference in the two subsets. For example, the difference of the $\mathcal{D}_l$ is computed by multiplying a scaling factor, $(|\mathcal{D}| - |\mathcal{D}_l|)(|\sum_{k\in K} |\mathcal{D}_l^k|^2)$. Finally, the total difference can be computed as $g'(\tau, f, \mathcal{D})$:

$$\begin{aligned} & g'(\tau, f, \mathcal{D}) \\ &= (|\mathcal{D}| - |\mathcal{D}_l|) \cdot |\sum_{k\in K} |\mathcal{D}_l^k|^2 + (|\mathcal{D}| - |\mathcal{D}_r|) \cdot \sum_{k\in K} |\mathcal{D}_r^k|^2 \\ &= |\mathcal{D}_r| \sum_{k\in K} |\mathcal{D}_l^k|^2 + |\mathcal{D}_l| \sum_{k\in K} |\mathcal{D}_r^k|^2 \end{aligned} \tag{3}$$

Overall, the (f, τ) pair whose $g'(\tau, f, \mathcal{D})$ value is the maximum is the best split. We compare our division-free splitting criterion with other classic DT algorithms, such as ID3, C4.5, etc, over plaintext data. We also theoretically prove the equivalence relation between $g'(\tau, f, \mathcal{D})$ and the original Gini Index $g(\tau, f, \mathcal{D})$ in Appendix B. Our experimental results (see Table V) show that the DT built with our division-free splitting criteria can achieve a comparable accuracy (with a general $\pm 0.2\%$ difference compared to the classic solution) with classic DT algorithms upon a variety of datasets.

The main process for our MPC-friendly DT algorithm is the same as Algorithm 5. We also implement $FindBestSplit(\mathcal{D})$ function with our $g'(\tau, f, \mathcal{D})$ in Algorithm 6 of Appendix C. In MPC, we apply arithmetic-friendly and secure integer multiplication to compute $g'(\tau, f, \mathcal{D})$ over the ring 2^{64}. The maximum of $g'(\tau, f, \mathcal{D})$ is upper bounded by $2 \cdot N^3$, where N is the number of samples. When comparing two different gains over $\mathcal{D}$, we can process at most $N = 2^{21} = 2,097,152$ samples due to $3 \cdot (\log(N + 1) - 1) = 64$, which is much larger than 8,192.

VI. Lightweight Secure Comparison

After calculating g' for all possible (f, τ) pairs, the parties need to compare them to find the maximum secretly. The state-of-the-art comparisons are based on DCF [32] since it only requires a single communication round. However, existing works incur heavy computation overhead during evaluation.

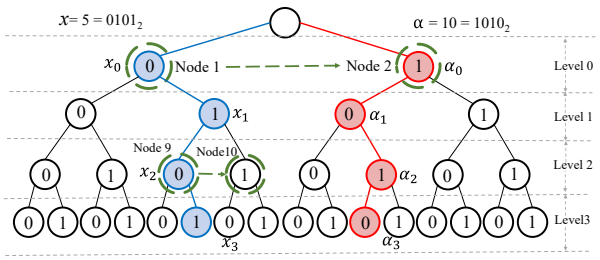


Fig. 2: Our lightweight DCF. A complete binary tree is built for $\alpha = 10$, where each node's left and right children are assigned 0 and 1, respectively. The path that matches α is the special path (the red one). To evaluate if $x < \alpha$, from the root to the bottom, we find the path matching x (the blue path) and check if it has a 0 node whose right sibling is on the special path. If yes, we must have $x < \alpha$; and $x \geq \alpha$ otherwise. In the example, $x = 5 = 0101_2$, on its matching path $Node_1$ and $Node_9$ are 0, and only $Node_1$'s right sibling $Node_2$ is on the special path, indicating $10 > 5$ is true.

This section presents a lightweight DCF with $0/1$-encoding and early termination technique to achieve better performance. Building upon this, we design a lightweight secure comparison protocol in a low-power fashion.

A. Distributed Comparison Function

Distributed Point Function [21] (DPF) is an FSS scheme designed for testing equality $x = \alpha$. Compared to DCF, the key size of DPF is smaller. Thus, our key insight is to replace DCF keys with DPF keys for DCF implementation. For this, **we can convert a comparison problem into a DPF-based equality test via $0/1$-encoding technology to achieve better performance**. Specifically, given α and the security parameter λ, our approach considers the process of generating keys as building a virtual tree. As shown in Fig. 2, the generator builds a complete binary tree with depth ℓ, where ℓ is the bit length of α (typically 32 or 64), and the left and right children of each node represent 0 and 1, respectively. From the top to the bottom of the tree level by level, we can compare node values with each bit of a number from the most significant one to the least. The path that matches α is called *the special path* (the red path in Fig. 2). For each node, a tag is used to indicate if it is on the special path, where it is 1 if yes and 0 otherwise.

In the evaluation, we rely on the 0/1 encoding technology. As mentioned in Section III-C, when $x < \alpha$ is true, we should have $|Set_1^\alpha \cap Set_0^x| = 1$. Recall that $Set_1^\alpha = \{\alpha_0\alpha_1...\alpha_{i-1}\alpha_i \mid \alpha_i = 1, 0 \leq i \leq \ell - 1\}$ and $Set_0^x = \{x_0x_1...x_{i-1}1 \mid x_i = 0, 0 \leq i \leq \ell - 1\}$. That is, if $x < \alpha$, x has only one $x_i = 0$ bit such that '$x_0 \ldots x_{i-1}1 = \alpha_0 \ldots \alpha_i$' Reflecting on the tree, there is a path that matches x (e.g., the blue path in Fig. 2), and '$x_0 \ldots x_{i-1}1 = \alpha_0 \ldots \alpha_i$' means x's first $i-1$ bits are on the special path and $x_i = 0$'s right sibling is on the special path [2]. **So we just need to check which bits of x are 0, get the tags of their right siblings, and check if their sum is 1.** As

[2]Since we always mark the left and right children as 0 and 1, respectively, $x_i = 0$ must match the left child of a node.

Algorithm 1 Lightweight Distributed Comparison Function

Let $G : \{0,1\}^\lambda \to \{0,1\}^{2(\lambda+1)}$ be a pseudorandom generator.

Gen(λ, α):

1: Let $\alpha_0, \dots \alpha_{\ell-1}$ be the bit string of $\alpha \in \{0,1\}^\ell$
2: $s_0^{(0)} \leftarrow \{0,1\}^\lambda$, $t_0^{(0)} \leftarrow 0$, and $s_1^{(0)} \leftarrow \{0,1\}^\lambda$, $t_1^{(0)} \leftarrow 1$
3: **for** $i \in [0, \ell-1]$ **do**
4: $\quad s_0^L||t_0^L||s_0^R||t_0^R \leftarrow G(s_0^{(i)})$ and $s_1^L||t_1^L||s_1^R||t_1^R \leftarrow G(s_1^{(i)})$
5: $\quad$ **if** $\alpha_i = 0$ **then**
6: $\quad\quad Keep \leftarrow L$, $Lose \leftarrow R$
7: $\quad$ **else**
8: $\quad\quad Keep \leftarrow R$, $Lose \leftarrow L$
9: $\quad$ **end if**
10: $\quad t_{cw}^L = t_0^L \oplus t_1^L \oplus \alpha_i \oplus 1$ and $t_{cw}^R = t_0^R \oplus t_1^R \oplus \alpha_i$
11: $\quad s_{cw} \leftarrow s_0^{Lose} \oplus s_1^{Lose}$
12: $\quad CW^{(i)} \leftarrow s_{cw}||t_{cw}^L||t_{cw}^R$
13: $\quad s_0^{(i+1)} \leftarrow s_0^{Keep} \oplus t_0^{(i)} \cdot s_{cw}$, and $s_1^{(i+1)} \leftarrow s_1^{Keep} \oplus t_1^{(i)} \cdot s_{cw}$
14: $\quad t_0^{(i+1)} \leftarrow t_0^{Keep} \oplus t_0^{(i)} \cdot t_{cw}^{Keep}$, and $t_1^{(i+1)} \leftarrow t_1^{Keep} \oplus t_1^{(i)} \cdot t_{cw}^{Keep}$
15: **end for**
16: **return** $\mathcal{K}_b = s_b^{(0)}||CW^{(0)}|| \dots ||CW^{(\ell-1)}$.

Eval($\mathcal{K}_b$,x):

1: Let $t_b^{(0)} \leftarrow b$ and $x = x_0, \dots, x_{\ell-1} \in \{0,1\}^\ell$
2: Parse $s_b^{(0)}||CW^{(0)}|| \dots ||CW^{(\ell-1)} \leftarrow \mathcal{K}_b$.
3: Set $\beta_b = 0$
4: **for** $i \in [0, \ell-1]$ **do**
5: $\quad$ Parse $s_{cw}||t_{cw}^L||t_{cw}^R \leftarrow CW^{(i)}$
6: $\quad \tau_b \leftarrow G(s_b^{(i)}) \oplus (t_b^{(i)} \cdot [s_{cw}||t_{cw}^L||s_{cw}||t_{cw}^R])$
7: $\quad$ Parse $s'^L||t'^L||s'^R||t'^R \leftarrow \tau_b$
8: $\quad$ **if** $x_i = 0$ **then**
9: $\quad\quad s_b^{(i+1)} \leftarrow s'^L$, $t_b^{(i+1)} \leftarrow t'^L$, $\beta_b = \beta_b \oplus t'^R$
10: $\quad$ **else**
11: $\quad\quad s_b^{(i+1)} \leftarrow s'^R$, $t_b^{(i+1)} \leftarrow t'^R$
12: $\quad$ **end if**
13: **end for**
14: **return** β_b

shown in in Fig. 2, let $x = 5 = 0101_2$ and $\alpha = 10 = 1010_2$. We can see $x_0 = x_2 = 0$, yet only x_0's right sibling $Node_2$ is on the special path. Due to the tree structure, for any x, there can be at most one $x_i = 0$ bit whose right sibling lies on the special path. Thus, the sum can only be 0 or 1, and we can replace the sum operation with xor.

Construction of Gen(λ, α)**.** The tree construction process is public, and only the location of the special path, i.e., each bit of α, should be protected. The key generation of DCF secretly shares the location of the special path to the two keys for P_0 and P_1. **Gen**(λ, α) in Algorithm 1 shows the key generation process, and the main idea is to store the location of the special path at each level in a correction word CW.

It is known that on the tree the special path must be either on the left or the right at each level. A λ-bit seed $s^{(i)}$ and a 1-bit tag $t^{(i)}$ are used to mark whether the special path is on the left or right at level i[3]. The main idea is to generate a string with the seed $s^{(i)}$ and a PRG $G : \{0,1\}^\lambda \to \{0,1\}^{2(\lambda+1)}$ for each level, divide the string into two halves, and map the left/right half to the left/right direction of the tree (i.e., 0/1 bit of α_i). Each half string is further divided into $s^@||t^@$, where $@ \in \{L, R\}$, $t^@$ has 1 bit, and $s^@$ has λ bits. If the special path is on the left (i.e., $\alpha_i = 0$), s^L and t^L are used to generate $t^{(i+1)}$; otherwise, s^R and t^R are used. The tag $t^{(i)}$ is generated in a way that $t_0^{(i)} \oplus t_1^{(i)} = 1$. Specifically, if $t^{(i)} = 1$, combining Lines 10 and 14 in Algorithm 1, we can get $t^{(i+1)} = t_0^{(i+1)} \oplus t_1^{(i+1)} = \alpha_i \oplus 1$ and $t^{(i+1)} = \alpha_i$ when $\alpha_i = 0$ and 1, respectively. That is, $t^{(i+1)} = 1$ when $t^{(i)} = 1$ no matter whether $\alpha_i = 0$ or 1. When $t^{(i)} = 0$, $t^{(i+1)} = t^{Keep}$. Since the root node is always on the special path, the generator initializes the tag $t^{(0)} = 1$ by setting $t_b^{(0)} = b$, resulting in $t^{(i)} = 1$ for all $i \in [0, \ell-1]$. The seed $s^{(i+1)} = s^{Keep} \oplus t^{(i)} \cdot s^{Lose}$. Since $t^{(i)} = 1$, so we have $s^{(i+1)} = s^{Keep} \oplus s^{Lose} = s^R \oplus s^L$ for all levels. At each level, s^{Lose}, $t_{cw}^L = t^L \oplus \alpha_i$, $t_{cw}^R = t^R \oplus \alpha_i$ are recorded into a correction word $CW^{(i)}$, composing the keys that be used to derive $s^{(i)}$ and $t^{(i)}$ during the evaluation.

Construction of Eval$(\mathcal{K}_b, x)$**.** According to the 0/1 encoding technique, we need to recover the tags of the right siblings of x's 0 bits and check if their xor sum is 1. Indeed, from the most significant bit to the least, the first different bit between x and α determines the comparison result. Precisely, when $x_0 ... x_{i-1} = \alpha_0 ... \alpha_{i-1}$ bit by bit, we must have $x < \alpha$ if $x_i < \alpha_i$ (i.e., $x_i = 0$ and α_i=1), and $x > \alpha$ if $x_i > \alpha_i$. To reflect on the tree, x_i is the first bit of x that separates from the special path (e.g., x_0 in Fig. 2) and only its right sibling's ($Node_2$ in Fig. 2) tag matters: $x < \alpha$ is true if it is 1. Thus, the main idea of our evaluation is to correctly recover the right sibling's tag of x's first bit that is not on the special path and set that to 0 for the remaining bits. We achieve that by recovering $s^{(i+1)}$ and $t^{(i+1)}$ correctly when $x_i = \alpha_i$ as they are used to generate the tag of the next level and ensuring $s^{(i+1)} = \{0\}^\lambda$ and $t^{(i+1)} = 0$ when $x_i \neq \alpha_i$.

Algorithm 1 shows our DCF evaluation **Eval**$(\mathcal{K}_b, x)$. The evaluation function of DCF for each party takes as inputs its own key $\mathcal{K}_b$ and a public value $x = x_0 \dots x_{\ell-1}$. From the key $\mathcal{K}_b$, each party recovers the initial seed $s_b^{(0)}$ and a common correction word $CW^{(i)}$ for each bit, which contains the location of the special path. With seed $s_b^{(i)}$, the tag $t_b^{(i)}$ and the PRG G, each party recovers the bit string, and gets $\tau_b \leftarrow G(s_b^{(i)}) \oplus (t_b^{(i)} \cdot [s_{cw}||t_{cw}^L||s_{cw}||t_{cw}^R])$. Based on the generation of s_{cw}, t_{cw}^L, and t_{cw}^R in **Gen**(λ, α), we have $\tau = \tau_0 \oplus \tau_1 \leftarrow s^L \oplus t^{(i)} \cdot s^{Loss}||t^L \oplus t^{(i)} \cdot (t^L \oplus \alpha_i \oplus 1)||s^R \oplus t^{(i)} \cdot s^{Loss}||t^R \oplus t^{(i)} \cdot (t^R \oplus \alpha_i)$. Given $t^{(0)} = ... = t^{(i-1)} = 1$, we get $\tau \leftarrow s^L \oplus s^{Loss}||\alpha_i \oplus 1||s^R \oplus s^{Loss}||\alpha_i$ at Line 7 for the i-th level, which depends on α_i and determines the seed $s^{(i+1)}$ and the tag $t^{(i+1)}$ for the next level, and β. The codes between Line 8 and 11 update them in 4 ways:

- When $x_i = \alpha_i = 0$, we have $s^{Loss} = s^R$ and get $s^{(i+1)} = s'^L = s^L \oplus s^R$ and $t^{(i+1)} = t'^L = \alpha_i \oplus 1 = 1$, and $t'^R = \alpha_i = 0$. This case means x_i is still on the special path. The right brother of x_i must not be on the special path, thus we have $\beta = \beta \oplus t'^R = \beta \oplus 0$. We need to continue to compare x_{i+1} and α_{i+1} in the next round with $s^{(i+1)} = s'^L = s^L \oplus s^R$ and $t^{(i+1)} = t'^L = \alpha \oplus 1 = 1$.
- When $x_i = 0$ and $\alpha_i = 1$, we have $s^{Loss} = s^L$ and get $s^{(i+1)} = s'^L = \{0\}^\lambda$, $t^{(i+1)} = t'^L = \alpha_i \oplus 1 = 0$, and $t'^R = \alpha_i = 1$. This case indicates that x_i has been separated from the special path and $t'^R = \alpha_i = 1$ is its right sibling, indicating $x < \alpha$. So we have $\beta = \beta \oplus 1$. The actual comparison can be finished. The following operations should not change the value of β, so we need to ensure $t'^R = 0$ for all the subsequent levels, and that is achieved with $s^{(i+1)} = \{0\}^\lambda$ (i.e., $s_0^{(i+1)} = s_1^{(i+1)}$). Note that once $s^{(i+1)} = \{0\}^\lambda$, based on the codes between Line 6 and 11, the seeds for all subsequent levels will

[3] $t_b^{(i)}$ and $s_b^{(i)}$ are the shares of $t^{(i)}$ and $s^{(i)}$ for P_b, respectively.

also be $\{0\}^\lambda$.

- When $x_i = 1$ and $\alpha_i = 0$, we have $s^{Loss} = s^R$ and get $s^{(i+1)} = s'^R = \{0\}^\lambda$, $t^{(i+1)} = t'^R = \alpha_i = 0$, and $t'^R = 0$. This case means x_i is the right brother of α_i, indicating $x > \alpha$. Similarly, it is unnecessary to do the following comparisons, and we need to ensure $t'^R = 0$ for all the subsequent levels. That is why $s^{(i+1)} = \{0\}^\lambda$.
- When $x_i = \alpha_i = 1$, we have $s^{Loss} = s^L$ and get $s^{(i+1)} = s'^R = s^R \oplus s^L$ and $t^{(i+1)} = t'^R = \alpha_0 = 1$, and $t'^R = 1$. This case means x_i is still on the special path. It is unclear if $x < \alpha$ until this level. The next round needs to check again with x_{i+1}, thus the seed same as the one generated in $\mathbf{Gen}(\lambda, \alpha)$, i.e., $s^{(i+1)} = s^L \oplus s^R$, is required.

In Algorithm 1, $\mathcal{K}_b$ consists of a λ-bit seed $s_b^{(0)}$ and ℓ correction words of $(\lambda + 2)$ bits each. Thus the key size is $\ell(\lambda+2)+\lambda$ bits, which is slightly longer than that of the Grotto scheme with a key size of $(\ell - \log_2 \lambda)(\lambda + 2) + 2\lambda$ bits. To further reduce the DCF key size and speed up evaluation, we apply the early termination technique from Boyle's DPF [21] to reduce the number of correction words.

B. DCF with Early Termination Optimization

With the early termination technique, we can divide the inputs into two halves: $\{\alpha_0, ..., \alpha_v;\ \alpha_{v+1}, ..., \alpha_{\ell-1}\}$ and $\{x_0, ..., x_v;\ x_{v+1}, ..., x_{\ell-1}\}$ and compare them on demand, where $v \in [0, \ell - 1)$. When $x_0 \cdots x_v \neq \alpha_0 \cdots \alpha_v$, we just need to compare the first $(v + 1)$-bit as they are the most significant bits and can determine the comparison result. Otherwise, we also need to check the remaining $(\ell-v-1)$-bit. The *early termination* technique can process the two cases securely and more efficiently. Our DCF with early termination is shown in Algorithm 7 of Appendix D.

To compare the first $v+1$ bits, the two parties still generate and evaluate the first $v + 1$ correction words as described in Algorithm 1. During the evaluation, the two parties terminate the comparison at level v and output the shares β_b, $s_b^{(v+1)}$ and $t_b^{(v+1)}$, where $\beta = \beta_0 \oplus \beta_1 = 1$ if $x_0 \dots x_v < \alpha_0 \dots \alpha_v$ and $\beta = 0$ otherwise. For the remaining $(\ell - v - 1)$-bit, instead of comparing them bit by bit with the $\ell - 1 - v$ correction words, we just need one special correction word, denoted as $\hat{CW}^{(v+1)}$, and operate with it to determine the final output. Specifically, $\hat{CW}^{(v+1)}$ is generated as follows: encode the last $\ell - v - 1$ bits $\hat{\alpha} = \alpha_{v+1} \dots \alpha_{\ell-1}$ into $2^{\ell-v-1}$ bits $\omega = \omega^{(0)} \dots \omega^{(2^{\ell-v-1}-1)}$, where $\omega^{(k)} = 1$ if $k < \hat{\alpha}$ for $k \in \mathbb{Z}_{2^{\ell-v-1}}$, otherwise $\omega^{(k)} = 0$. For example, when $\hat{\alpha} = 011_2 = 3$, we have $\omega^{(k)} = 1$ for $k \in [0, 3)$, resulting in $\omega = 11100000$. After that, generate $\hat{CW}^{(v+1)} = (\omega \oplus s_0^{(v+1)} \oplus s_1^{(v+1)})$. During the evaluation, for the remaining $\ell - 1 - v$ bits, each party just need to recover $\omega_b = s_b^{(v+1)} \oplus (t_b^{(v+1)} \cdot \hat{CW}^{(v+1)})$, choose the $\hat{x}$-th value $\omega_b^{(\hat{x})}$ from ω_b, where $\hat{x} = x_{v+1} \dots x_{\ell-1}$, and update $\beta_b = \omega_b^{(\hat{x})} \oplus \beta_b$. By doing so, the output is correct in any case:

- When $x_0 \cdots x_v \neq \alpha_0 \cdots \alpha_v$, β generated at level v should be the final output, and the remaining operation $\beta = \omega^{\hat{x}} \oplus \beta$ should not change the value of β, i.e., $\omega^{\hat{x}} = 0$. The above design achieves that: Recall that $x_0 \cdots x_v \neq \alpha_0 \cdots \alpha_v$ means x has left the special path at level v, indicating $t^{(v+1)} = 0$ and $s^{(v+1)} = 0$ (i.e., $t_0^{(v+1)} = t_1^{(v+1)}$ and $s_0^{(v+1)} = s_1^{(v+1)}$), thus $\omega = \omega_0 \oplus \omega_1 = s_0^{(v+1)} \oplus s_1^{(v+1)} \oplus (t_0^{(v+1)} \cdot \hat{CW}^{(v+1)}) \oplus (t_1^{(v+1)} \cdot \hat{CW}^{(v+1)}) = 0$, indicating $\omega^{(\hat{x})} = 0$.
- When $x_0 \cdots x_v = \alpha_0 \cdots \alpha_v$, $\beta = 0$ at level v, and the remaining operation $\beta = \omega^{(\hat{x})} \oplus \beta$ should determine the final value of β, i.e., $\omega^{(\hat{x})} = 1$ when $x_{v+1} \dots x_{\ell-1} < \alpha_{v+1} \dots \alpha_{\ell-1}$ and $\omega^{\hat{x}} = 0$ otherwise. The above design also ensures that: Since x is still at the special path at level v, $t^{(v+1)} = t_0^{(v+1)} \oplus t_1^{(v+1)} = 1$ and $s^{(v+1)} = s_0^{(v+1)} \oplus s_1^{(v+1)} \neq 0$, thus the recovered ω is the same as the one generated in $\mathbf{Gen}(\lambda, \alpha)$, where $\omega^{(k)} = 1$ if $k < \hat{\alpha}$ for $k \in \mathbb{Z}_{2^{\ell-v-1}}$, otherwise $\omega^{(k)} = 0$, i.e., $\omega^{(\hat{x})} = 1$ if $\hat{x} < \hat{\alpha}$, and $\omega^{(\hat{x})} = 0$ otherwise.

Since the seed size is λ-bit, the maximum size of ω is λ. A λ-bit ω can encode at most $\lambda = 2^{\ell-1-v}$ possible values for $\alpha_{v+1}, \dots, \alpha_{\ell-1}$. Thus, we set $v = \ell - 1 - \log_2 \lambda$.

Efficiency. Based on the above optimization, our DCF key has only a λ-bits seed $s_b^{(0)}$, $v + 1$ correction words with $(\lambda + 2)$-bits and a λ-bits $\hat{CW}^{(v+1)}$. Thus, the total key size of our optimized DCF are $(\ell - \log_2 \lambda)(\lambda + 2) + 2\lambda$. Compared with Grotto scheme [24] with a parity-segment tree, we maintain the same key size but reduce the running time of $\mathbf{Eval}(\mathcal{K}_b, x)$ from $\mathcal{O}((\ell - \log \lambda) \log \lambda)$ to $\mathcal{O}(\ell - \log \lambda)$. The reason is our DCF only has access to each tag t'_R of $x_i = 0$ in the range $[0, \ell - 1 - \log_2 \lambda]$ and a lookup table with $\mathcal{O}(1)$. However, Grotto traverses a parity-segment tree with $\mathcal{O}(\log \lambda)$ to determine the next path direction at each level in $\mathbf{Eval}(\mathcal{K}_b, x)$. Therefore, our DCF is more efficient in the evaluation.

It is important to underline that in our two-party model, the generation function $\mathbf{Gen}(\lambda, \alpha)$ of DCF must be collaboratively constructed by the two parties in a secret-shared manner. Thus, we also provide secure two-party $\mathbf{Gen}(\lambda, \langle\alpha\rangle)$ protocol (Algorithm 8 in Appendix E) to generate keys.

C. Lightweight Secure Comparison

In our DT algorithm that can operate exclusively on non-negative integers, we need to compare $y \in [0, 2^{\ell-1})$ and $z \in [0, 2^{\ell-1})$ in secret-shared form due to mutual distrust between parties. However, the DCF above cannot be directly used in FSS-DT, as its **Eval** function's input x is public by design. Some prior works hide $x = y - z$ by adding a random α from $\mathbb{Z}_{2^\ell}$ and then use a single DCF to check $x+\alpha < \alpha$. However, they could output the wrong comparison result if $x + \alpha$ wraps around[4]. To avoid that, We propose a lightweight comparison protocol that just calls DCF once. Leveraging the logic rule of ripple-carry adders, our key idea is to transform the comparison of two ℓ-bit secret shared values into a comparison between two $(\ell-1)$-bit secret shared values in the ring $\mathbb{Z}_{2^\ell}$. The detail is shown in Algorithm 2.

Each party P_b inputs the arithmetic shares $\langle y\rangle$ and $\langle z\rangle$ into the protocol and gets the Boolean share of the result $[\![\theta]\!]$, where

[4]For example, assume $x = 10$ and $\alpha = 2^\ell - 1$ (note that ℓ is typically 32 or 64), then $x' = x + \alpha = 10 + 2^\ell - 1 \mod 2^\ell = 9$. As a result, $x' < \alpha \Leftrightarrow x + \alpha < \alpha \Leftrightarrow x < 0$ in $\mathbb{Z}_{2^\ell}$, which is wrong as $x > 0$.

Algorithm 2 Secure Comparison Protocol (Π_{SC})

Input: P_b for $b \in \{0,1\}$ holds $\langle y \rangle_b$ and $\langle z \rangle_b$.
Output: P_b holds $[\![\theta]\!]_b$ where $\theta = 0$ if $y \geq z$ and 1 otherwise.

Offline phase:
1: P_b samples an $(\ell-1)$-bit random value $v_b \in \mathbb{Z}_{2^{\ell-1}}$.
2: $\langle \alpha' \rangle$ = ConvertShare$(v_b, \ell-1, \ell)$ // Invoke the 2PC *ConvertShare* protocol of Ents.
3: P_b samples a random bit $r_b \in \{0,1\}$ and set $[\![\text{MSB}(\alpha)]\!]_b = r_b$.
4: P_0 and P_1 compute $\langle \alpha_{\ell-1} \rangle = \textbf{B2A}([\![\text{MSB}(\alpha)]\!])$
5: P_0 and P_1 compute $\langle \alpha \rangle = \langle \alpha_{\ell-1} \rangle \cdot 2^{\ell-1} + \langle \alpha' \rangle$.
6: P_b receives $\mathcal{K}_b \leftarrow \text{DCF.}\textbf{Gen}(\lambda, \langle \alpha' \rangle)$ on $\mathbb{Z}_{2^\ell}$.

Online phase:
1: P_b computes $\langle x \rangle_b = (\langle y \rangle_b - \langle z \rangle_b) \mod 2^\ell$.
2: P_b sends $\langle x \rangle_b - \langle \alpha \rangle_b$ to P_{1-b} such that they open $x - \alpha$.
3: P_b computes $\hat{x} = 2^{\ell-1} - 1 - x'$ and $x' = x - \alpha \mod 2^{\ell-1}$.
4: P_b computes the carry bit $[\![carry]\!]_b = \text{DCF.}\textbf{Eval}(\mathcal{K}_b, \hat{x})$.
5: P_b outputs $[\![\theta]\!]_b = [\![carry]\!]_b \oplus \text{MSB}(x-\alpha) \oplus [\![\text{MSB}(\alpha)]\!]_b$.

$\theta = 1$ if $y < z$ and $\theta = 0$ otherwise. To accommodate the representation of all values in $\mathbb{Z}_{2^\ell}$, our scheme employs two's complement notation. As done in [31], [39], [40], we can transform the comparison operation $y < z$ into analyzing the most significant bit (MSB) of $x = y - z$, where $y \in [0, 2^{\ell-1})$ and $z \in [0, 2^{\ell-1})$. According to two's complement notation, $\text{MSB}(x) = 1$ if $x < 0$ and $\text{MSB}(x) = 0$ otherwise. Thus, the comparison can be represented as follows:

$$\langle y \rangle < \langle z \rangle \Leftrightarrow \langle \text{MSB}(x = (y - z)) \rangle. \tag{4}$$

As done in Sigma [41] and CrypTFlow2 [31], based on the logic of ripple-carry adders, we have:

$$\text{MSB}(x) = \text{MSB}(x - \alpha) \oplus \text{MSB}(\alpha) \oplus carry \tag{5}$$

where $x' = x - \alpha \mod 2^{\ell-1}$, $\alpha' = \alpha \mod 2^{\ell-1}$ and $carry = \{x' + \alpha' \geq 2^{\ell-1}\}$, i.e., $carry = 1$ iff $x' + \alpha' \geq 2^{\ell-1} \Leftrightarrow 2^{\ell-1} - 1 - x' < \alpha'$ and $carry = 0$ otherwise.

To protect the secret value $\langle x \rangle$, we precompute a random mask $\alpha = (\alpha_{\ell-1} \cdot 2^{\ell-1} + \alpha') \in \mathbb{Z}_{2^\ell}$ on the offline phase, where $\alpha_{\ell-1} = \text{MSB}(\alpha)$ and $\alpha' \in \mathbb{Z}_{2^{\ell-1}}$. In the online phase, we reveal $x' = (x - \alpha) \mod 2^{\ell-1}$. For computing $carry$ in Eq. 5, we use our DCF to evaluate $2^{\ell-1} - 1 - x' < \alpha'$, where $\mathcal{K}_b = \text{DCF}(\langle \alpha' \rangle)$ and $\alpha' = \langle \alpha' \rangle_0 + \langle \alpha' \rangle_1 < 2^{\ell-1}$. Since $\hat{x} = 2^{\ell-1} - 1 - x'$ is guaranteed to be non-negative (as $\hat{x} \in \mathbb{Z}_{2^{\ell-1}}$), this computation avoids modular wraps around in $\mathbb{Z}_{2^\ell}$, thereby requiring only **a single DCF** call to resolve the comparison. Thus, based on Eq. 5 and Eq. 4, the output of secure comparison is $\text{MSB}(x-\alpha) \oplus [\![\text{MSB}(\alpha)]\!] \oplus [\![carry]\!]$.

VII. Construction of FSS-DT

A. Secure Training Protocol

With DFDT and the lightweight secure comparison protocol, we propose an efficient and secure training protocol for DT, as shown in Algorithm 3. This protocol outputs a full tree, as done in other MPC-based DT works [8], [9]. As mentioned, the main process of the training is to partition the data samples securely into two children and find the best split. FSS-DT uses an indicator to partition samples privately. Thus, our secure training protocol for one tree node comprises three parts: construction of the sample indicator, computation of the best splitting threshold, and computation of the leaves.

Construction of the sample indicator. To hide sample distribution, P_0 and P_1 use a secret shared vector $\langle \mathbf{w}^i \rangle = (\langle w_0^i \rangle, \ldots, \langle w_{N-1}^i \rangle)$ of size $N = |\mathcal{D}|$ to indicate which samples are available on the i-th node. For example, $w_q^i = 1$ indicates the sample $\mathbf{S}_q$ is available at the i-th node, and $w_q^i = 0$ otherwise. Clearly, for root node, $\langle w_q^i \rangle = \langle 1 \rangle$ for each $q \in [0, N-1]$ and $i = 0$. Now, we show how to update the indicators for the two children nodes given the indicator $\langle \mathbf{w}^i \rangle$ for the i-th node. Following the previous works

Algorithm 3 FSS-DT Training ($\Pi_{\text{FSS-DT-training}}$)

Secure Building Tree Process

Input: Local samples $\mathcal{D}_b$, Secret sharing indicator $\langle \mathbf{w}^i \rangle$ of i-th node, Label vector of sample $\mathbf{y}$ held by P_1.
Output: Secure distributed tree model T_b for P_b.
Function: SecureBuildTree($\mathcal{D}$,$\langle \mathbf{w}^i \rangle$,$T_b$,$i$):
1: Tree model $T_b = []$,
2: **if** internal nodes $i \in \{0, \ldots, 2^{\mathcal{H}-1} - 2\}$ **then**
3: $(\langle f_*^i \rangle, \langle \tau_*^i \rangle) \leftarrow$ SecureBestSplit($\mathcal{D}_b$, $\langle \mathbf{w}^i \rangle$).
4: **[Open the index f_*^i and the best split τ_*^i.]** $c = 1 \oplus \textbf{Open}(\langle f_*^i \rangle \geq m_0)$; P_c receives $\langle f_*^i \rangle_{1-c}$ and $\langle \tau_*^i \rangle_{1-c}$ from P_{1-c}; P_c opens $f_*^i = \langle f_*^i \rangle_c + \langle f_*^i \rangle_{1-c}$ and $\tau_*^h = \langle \tau_*^i \rangle_c + \langle \tau_*^i \rangle_{1-c}$ $\triangleright$ c indicates that P_c holds m_*^i and τ_*^i; m_0 is feature number held by P_0.
5: **[Record the threshold and feature index of best split.]** P_c records $T_c[i].\tau = \tau_*^i$, $T_c[i].f_*^i$; $T_{1-c}[i].\tau = T_{1-c}[i].f = -1$;
6: **[Update indicator vector.]** P_c computes $\langle \mathbf{w} \rangle_c = \{(S_{0,f_*^i} < \tau_*^i), \ldots, (S_{N-1,f_*^i} < \tau_*^i)\}$ and $\langle \mathbf{w} \rangle_{1-c} = \mathbf{0}$; P_0 and P_1 compute $\langle \mathbf{w}^{2i+1} \rangle = \langle \mathbf{w}^i \rangle \cdot \langle \mathbf{w} \rangle$ and $\langle \mathbf{w}^{2h+1} \rangle = (\mathbf{1} - \langle \mathbf{w} \rangle) \cdot \langle \mathbf{w}^{2i+1} \rangle$. $\triangleright$ $\mathbf{F}_{f_*^i} = (S_{0,f_*^i}, \ldots, S_{N,f_*^i})$ is the f_*^i-th feature vector held by P_c.
7: **[Build sub-trees.]** SecureBuildTree($\mathcal{D}_b$,$\langle \mathbf{w}^{2i+1} \rangle$,$T_b$,$2i+1$); SecureBuildTree($\mathcal{D}_b$ $\langle \mathbf{w}^{2i+2} \rangle$, T_b,$2i+2$).
8: **end if**
9: Initialize $\langle y_* \rangle_b = b$, $\langle N_* \rangle_b = 0$.
10: **if** leaf nodes $i \in \{2^{\mathcal{H}-1} - 1, \ldots, 2^{\mathcal{H}} - 2\}$ **then**
11: **[Compute the class with maximum number on the leaf node.]** $\langle N_k \rangle = \sum \textbf{B2A}((\langle \mathbf{y} \rangle \cdot \langle \mathbf{w} \rangle - Y_k)^2 - 1 \geq 0)$, $\langle \theta \rangle = \textbf{B2A}(\langle N_* \rangle \geq \langle N_k \rangle)$, $\langle y_* \rangle = (\langle 1 - \theta \rangle) \cdot \langle y_* \rangle + \langle \theta \rangle \cdot Y_k$ for each $Y_k \in \mathbf{Y}$. $T_b[i].\tau = \langle y_* \rangle$.
12: **end if**
13: **return** a secure tree T_b

Secure Best Split Process

Input: Local samples $\mathcal{D}_b$, Secret sharing indicator $\langle \mathbf{w}^i \rangle$.
Output: The best split ($\langle f_*^i \rangle$, $\langle \tau_*^i \rangle$)
Function: SecureBestSplit($\mathcal{D}_b$,$\langle \mathbf{w}^i \rangle$):
1: $\langle g_*^{'} \rangle = \langle 2^{\ell-1} - 1 \rangle$, $\langle f_*^i \rangle_b = 0$
2: **[Locally pick all possible threshold.]** P_b computes $\mathcal{T}_b[j] \leftarrow$ Distinct values of $\mathbf{F}_j$ for $\forall j \in [b \cdot m_b, (1-b) \cdot (m_0 - 1) + b \cdot (m_1 - 1)]$.
3: **for** $j \in [b \cdot m_b, (1-b) \cdot (m_0 - 1) + b \cdot (m_1 - 1)]$, $\tau \in \mathcal{T}_b[j]$ **in parallel do**
4: **[Compute left indicator.]** P_c constructs $\langle \mathbf{w} \rangle_c = (\mathbf{F}_j \leq \tau)$ and P_{1-c} constructs $\langle \mathbf{w} \rangle_{1-c} = \mathbf{0}$; $\langle \mathbf{w}^{2i+1} \rangle = \langle \mathbf{w} \rangle \cdot \langle \mathbf{w}^i \rangle$.
5: **[Compute right indicator.]** $\langle \mathbf{w}^{2i+2} \rangle = \langle \mathbf{w}^i \rangle \cdot (\mathbf{1} - \langle \mathbf{w} \rangle)$.
6: **[Compute the number of available samples on children.]** $\langle |\mathcal{D}_l| \rangle = \sum_{q=0}^{N} \langle w_q^{2i+1} \rangle$; $\langle |\mathcal{D}_r| \rangle = \sum_{q=0}^{N-1} \langle w_q^{2i+2} \rangle$.
7: **[Count the number of available samples for each class $Y_k \in \mathbf{Y}$ on children and compute the square sum of their result.]** $\langle |\mathcal{D}_l^k|^2 \rangle = \sum_{k=1}^{K} (\sum_{q=0}^{N-1} 1\{y_q = Y_k\} \cdot \langle w_q^{2i+1} \rangle)^2$ and $\langle |\mathcal{D}_r^k|^2 \rangle = \sum_{k=0}^{K} (\sum_{q=0}^{N-1} 1\{y_q = Y_k\} \cdot \langle w_q^{2i+2} \rangle)^2$. $\triangleright$ $1\{a = b\}$: 1 if $a = b$, and 0 otherwise.
8: **[Compute $g'_{j,\tau}$ with our splitting criterion.]** $\langle g'_{j,\tau} \rangle = \langle |\mathcal{D}_r| \rangle \cdot \langle |\mathcal{D}_l^k|^2 \rangle + \langle |\mathcal{D}_l| \rangle \cdot \langle |\mathcal{D}_r^k|^2 \rangle$.
9: **end for**
10: **[Compute the best τ_*^i, f_*^i]** $\langle \theta \rangle = \textbf{B2A}((\langle g'_{j,\tau} \rangle - \langle g_*^{'} \rangle) < 0)$, $\langle \tau_*^i \rangle = (1 - \langle \theta \rangle) \cdot \tau + \langle \theta \rangle \cdot \langle \tau_*^i \rangle$. $\langle f_*^i \rangle = (1 - \langle \theta \rangle) \cdot j + \langle \theta \rangle \cdot \langle m_*^i \rangle$ for $j \in [0, m-1]$, $\tau \in \mathcal{T}_0[j] \cup \mathcal{T}_1[j]$.
11: **return** $\langle f_*^i \rangle$, $\langle \tau_*^i \rangle$.

[8], [9], [17], an 1-bit indicator $c \in \{0,1\}$ is revealed to determine who is the best split owner, say P_c. Then the best split (f_*^i, τ_*^i) is visible to the corresponding feature owner, who can thus (locally) record the split information. The other party learns nothing more than the indicator bit. Once the best split (f_*^i, τ_*^i) for the i-th node is determined, P_c will find the f_*^i-th feature vector $\mathbf{F}_{f_*^i} = (S_{0,f_*^i}, \ldots, S_{N-1,f_*^i})$ in its dataset. Then P_c will first generate an intermediate indicator $\langle \mathbf{w} \rangle_c = \{(S_{0,f_*^i} < \tau_*^i), \ldots, (S_{N-1,f_*^i} < \tau_*^i)\}$ locally[5]. P_{1-c} who does not have the f_*^i-th feature will simply set $\langle \mathbf{w} \rangle_{1-c} = \mathbf{0}$. Now we have a secret-shared vector $\langle \mathbf{w} \rangle = (\langle \mathbf{w} \rangle_0, \langle \mathbf{w} \rangle_1)$. To obtain the indicators for the two children nodes, P_0 and P_1 can update the indicator for the left child by computing $\langle \mathbf{w}^{2i+1} \rangle = \langle \mathbf{w}^i \rangle \cdot \langle \mathbf{w} \rangle$ and update the indicator for the right child by computing $\langle \mathbf{w}^{2i+2} \rangle = \langle \mathbf{w}^i \rangle \cdot (\mathbf{1} - \langle \mathbf{w} \rangle)$.

Computation of the best splitting. To find the best split (f_*^i, τ_*^i) for $T[i]$, P_b firstly picks all possible thresholds of each feature locally from their dataset $\mathcal{D}_b$. For a possible $\tau \in \mathcal{T}[j]$ of j-th feature $\mathbf{F}_j = (S_{0,j}, \ldots, S_{N-1,j})$, the two parties generate possible intermediate indicators locally, i.e., $\langle \mathbf{w} \rangle_c = (S_{0,j} < \tau, \ldots, S_{N-1,j} < \tau)$, $\langle \mathbf{w} \rangle_{1-c} = \mathbf{0}$ and update the possible left indicator $\langle \mathbf{w}^{2i+1} \rangle = \langle \mathbf{w} \rangle \cdot \langle \mathbf{w}^i \rangle$ on the i-th tree node. For the possible right indicator vector, two parties compute $\langle \mathbf{w}^{2i+2} \rangle = \langle \mathbf{w}^i \rangle \cdot (\mathbf{1} - \langle \mathbf{w} \rangle)$. Next, P_0 and P_1 compute the number of available samples on the left and right subsets, i.e., $\langle |\mathcal{D}_l| \rangle = \sum_{q=0}^{N-1} \langle w_q^{2i+1} \rangle$ and $\langle |\mathcal{D}_r| \rangle = \sum_{q=0}^{N-1} \langle w_q^{2i+2} \rangle$. In the following, two parties count the number of available samples for each class $Y_k \in \mathbf{Y}$ on children and compute the square sum of their result, i.e., $\langle |\mathcal{D}_l^k|^2 \rangle$ and $\langle |\mathcal{D}_l^k|^2 \rangle$. After that, the two parties jointly compute the all splitting scores $\{\langle g'_{\tau,j} \rangle\}_{\forall \tau \in \mathcal{T}, j \in [0,m-1]}$ from the secret shares $\{\langle |\mathcal{D}_l^k|^2 \rangle, \langle |\mathcal{D}_l^k|^2 \rangle, \langle |\mathcal{D}_l| \rangle, \langle |\mathcal{D}_r| \rangle\}$ according to our splitting criterion. Finally, we compute the best splitting threshold $\langle \tau_*^i \rangle$ and the feature index $\langle f_*^i \rangle$ of best split using the same way as the computation of the label on the leaf node.

Computation of the leaf node. For a leaf node, P_0 and P_1 securely select a label y_* that occurs most frequently and assign it to the leaf node. P_b locally initializes $\langle y_* \rangle_b = 0$ and a most occurance times $\langle N_* \rangle_b = 0$. P_0 and P_1 count the total number of samples for each label $Y_k \in \mathbf{Y}$ by computing $\langle N_k \rangle = \sum_{q=0}^{N-1} \mathbf{B2A}((\langle y_q \rangle \cdot \langle w_q^i \rangle - Y_k)^2 - 1 \geq 0)$. Then P_0 and P_1 update $\langle y_* \rangle = (\langle 1 - \theta \rangle) \cdot \langle y_* \rangle + \langle \theta \rangle \cdot Y_k$ where $\langle \theta \rangle = \mathbf{B2A}(\langle N_* \rangle \geq \langle N_k \rangle)$. Finally, the secret-sharing $\langle y_* \rangle_b$ is recorded to the current leaf node held by each party P_b, i.e., $T_b[i].\tau = \langle y \rangle_b$ for $i \in \{2^{\mathcal{H}-1}, \ldots, 2^{\mathcal{H}} - 1\}$.

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_b, all the other information on P_b is hidden from P_{1-b}.

B. Secure Inference Protocol

With T_0 and T_1, P_0 and P_1 can jointly make a secure inference for a sample. Assume that P_0 holds features $\mathcal{S}_0 = \{S_{s,0}, \ldots, S_{s,m_0-1}\}$ and P_1 holds features $\mathcal{S}_1 = \{S_{s,m_0}, \ldots, S_{s,m_0+m_1-1}\}$. Indeed, P_0 and P_1 jointly have all features of the s-th sample $\mathbf{S}_s = \mathcal{S}_0 \| \mathcal{S}_1$. We design a secure inference protocol for distributed samples without leaking any private information in Algorithm 4.

[5]For *local comparison* in plaintext, given two values a and b, we define the output as 1 if $a > b$ and 0 otherwise.

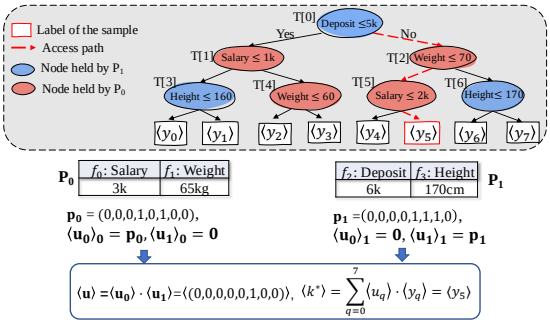


Fig. 3: Toy example for secure inference of FSS-DT. P_b first builds a vector $\mathbf{p}_b$ of size $2^{\mathcal{H}-1}$ based on T_b, where $\mathbf{p}_{b,i} = 0$ and $\mathbf{p}_{b,i} = 1$ indicate the evaluation path of the sample cannot and might reach leaf $T[2^{\mathcal{H}-1} - 1 + i]$, respectively. Second, they secretly set $\langle \mathbf{u}_b \rangle_b \leftarrow \mathbf{p}_b$ and $\langle \mathbf{u}_{1-b} \rangle_b \leftarrow \mathbf{0}$. The inference result $\langle k^* \rangle$ of this sample is computed with element-wise multiplication.

Algorithm 4 FSS-DT Inference ($\Pi_{\text{FSS-DT-inference}}$)

Input: A trained distributed tree list T_b with size of $2^{\mathcal{H}-1}$ held by P_b; The s-th sample $\mathbf{S}_s = \mathcal{S}_1 \| \mathcal{S}_0$, where P_0 holds $\mathcal{S}_0 = \{S_{s,0}, \ldots, S_{s,m_0-1}\}$ with m_0 features and P_1 holds $\mathcal{S}_1 = \{S_{s,m_0}, \ldots, S_{s,m_0+m_1-1}\}$ with m_1 features.
Output: The secret sharing inference result $\langle k^* \rangle$

Let $n_l = 2^{\mathcal{H}-1} - 2$ and $\mathbf{p}_b = \mathbf{0}$
for $i \in [1, n_l]$ **in parallel do**
  if $T_b[i].f \neq -1$ **then**
    P_b computes a local access path $\mathbf{p}_b$ by comparing $T[i].\tau$ with $S_{s,T[i].\tau}$.
  else
    P_b set $\langle y_i \rangle = T[i].\tau$
  end if
end for
[Build a secret sharing access path] Let $\langle \mathbf{u}_b \rangle_b = \mathbf{p}_b$ and $\langle \mathbf{u}_{1-b} \rangle_b = \mathbf{0}$.
[Compute label of sample] $\langle k^* \rangle = \sum_{i=0}^{2^{\mathcal{H}-1}-1} \langle u_{0,i} \rangle \cdot \langle u_{1,i} \rangle \cdot \langle y_i \rangle$, where $\langle y_i \rangle_b = T_b[2^{\mathcal{H}-1} - 1 + i].\tau$ and $\langle u_{b,i} \rangle \in \langle \mathbf{u}_b \rangle$.

The complete tree T with depth $\mathcal{H}$ has $2^{\mathcal{H}-1}$ leaves: $(T[2^{\mathcal{H}-1} - 1], \ldots, T[2^{\mathcal{H}} - 2])$. For simplicity, we denote them as $\mathbf{y}^* = (y_0, \ldots, y_{2^{\mathcal{H}-1}-1})$. Recall that each party holds the tree part involving its features. By comparing its features and corresponding node thresholds, each party P_b can locally prepare a boolean test vector $\mathbf{p}_b = (p_{b,0}, \ldots, p_{b,2^{\mathcal{H}-1}-1})$. Specifically, $p_{b,i} = 0$ for $i \in [0, 2^{\mathcal{H}-1} - 1]$ means the execution path of T_b **cannot** reach into leaf y_i (i.e., $T[2^{\mathcal{H}-1} + i]$) according to the nodes in T_b. In contrast, $p_{b,i} = 1$ means the access path **might** reach into leaf y_i. The matched label can be obtained by computing $\mathbf{p}_0 \cdot \mathbf{p}_1 \cdot \mathbf{y}^* = (p_{0,0} \cdot p_{1,0} \cdot y_0, \ldots, p_{0,2^{\mathcal{H}-1}} \cdot p_{1,2^{\mathcal{H}-1}} \cdot y_{2^{\mathcal{H}-1}}) \cdot \mathbf{y}^*$. To avoid interaction between the two parties, we share $\mathbf{p}_b$ in a special way: $\mathbf{p}_b$'s share for P_b is itself, and its share for P_{1-b} is $\mathbf{0}$. Both parties know they will get $\mathbf{0}$ from each other, so they just build it locally without communication. Precisely, P_b locally sets $\langle \mathbf{u}_b \rangle_b \leftarrow \mathbf{p}_b$ and $\langle \mathbf{u}_{1-b} \rangle_b \leftarrow \mathbf{0}$. Then the inference result is to compute $\langle k^* \rangle = \sum_{i=0}^{2^{\mathcal{H}-1}-1} \langle u_{0,i} \rangle \cdot \langle u_{1,i} \rangle \cdot \langle y_i \rangle$, where $\langle \mathbf{u}_b \rangle = (\langle u_{b,0} \rangle, ..., \langle u_{b,2^{\mathcal{H}-1}-1} \rangle)$.

Fig. 3 shows an example for the inference. The parties first evaluate their features with their nodes locally. For instance,

TABLE III: Communication overhead

Process	Scheme	Comm. rounds	Comm. cost
Multiplication	Pivot [8]	1	ℓ_1
	Swan [14]	1	ℓ
	FSS-DT	1	2ℓ
Comparison	Pivot [8]	$\log \ell$	$12\ell - 12$
	Swan [14]	2	$0.5\log^2 \ell + 2\ell$
	FSS-DT	1	ℓ
Leaf node training	Pivot [8]	$K(\log \ell + 10)$	$NK(42(\ell + f) - 2)$
	Swan [14]	$\log(NK) + 11$	$NK\ell + (C-1)\log^2 \ell + 6\ell$
	FSS-DT	$7K$	$7NK\ell + 7K\ell$
Internal node training	Pivot [8]	$8m\mathcal{T}\log \ell + 16 + m\mathcal{T}\mathcal{O}_d(\ell^2))$	$m\mathcal{T}(158NK + 128)\ell/3 + (167N + 16)\ell/3 + 6m\mathcal{T}K\mathcal{O}_d(f\ell^2)$
	Swan [14]	$mlog(\mathcal{T}K) + 12 + \log(F)\mathcal{O}_d(\ell^2)$	$m\mathcal{T}K(2NK + 32)\ell/3 + (10N + 1)\ell/3 + 2m\mathcal{T}K\mathcal{O}_d(\ell^2)$
	FSS-DT	$4m\mathcal{T} + 9$	$m\mathcal{T}(4N + 4K + 8)\ell + (4N + 2)\ell$
Inference	Pivot [8]	2	$36(2^{\mathcal{H}} - 2)\ell$
	Swan [14]	1	$3(2^{\mathcal{H}} - 1)\ell$
	FSS-DT	2	$(2^{\mathcal{H}} - 2)\ell$

ℓ and $\ell_1 \approx 18\ell$ are the bit-length of the ring in ASS and HE, respectively; f is the bit-length of the fraction in fixed-point values, where $f = 20 \approx \ell/3$ in Pivot [8]; Pivot [8] involves secure multiplication of the fixed-point number and secure division in their secure DT. $\mathcal{O}_d(\ell^2)$ is an approximated communication complexity for their secure division. A secure truncation process incurs a communication cost of f bits; We note that Pivot uses a mixed approach of ASS and HE to optimize communication rounds.

P_0 has weight $65kg > T_0[4].\tau(i.e., 60)$, so y_2 is reachable but y_3 cannot, and it gets $p_{0,2} = 0$ and $p_{0,3} = 1$. P_0 gets $\mathbf{p}_0 = (0,0,0,1,0,1,0,0)$, and P_1 gets $\mathbf{p}_1 = (0,0,0,0,1,1,1,0)$. Secondly, P_b locally sets $\langle \mathbf{u}_b \rangle_b \leftarrow \mathbf{p}_b$ and $\langle \mathbf{u}_{1-b} \rangle_b \leftarrow \mathbf{0}$. The inference result of this sample can be obliviously calculated by element-wise multiplication as follows: $\langle k^* \rangle = \sum_{i=0}^{7} \langle u_{0,i} \rangle \cdot \langle u_{1,i} \rangle \cdot \langle y_i \rangle = \langle y_5 \rangle$, where $\langle y_i \rangle_b = T_b[7+i].\tau$ and $\langle u_{b,i} \rangle \in \langle \mathbf{u}_b \rangle$. This approach ensures that no participant knows which path in the distributed decision list was utilized during the secure inference process, thus maintaining privacy.

C. Theoretical Analysis

Security. Similar to the existing works on vertical DT (e.g., Swan and Pivot), FSS-DT is built on the two-party vertical dataset, where each party contributes to different and independent features of the same samples. In a semi-honest setting, the 1-bit opened indicator reveals split information only to the feature owner, masking it as a dummy node (-1,-1) for the non-owning party. This isolation, combined with feature independence, effectively precludes the possibility of inter-party feature inference. This mechanism leverages feature independence to prevent any cross-party inference, rendering the sensitive attributes of one party invisible to the other. Besides, during training and inference, all data exchanged between parties are secret shares. Linear operations (addition, multiplication, B2A gadgets) are securely performed using ASS, while the nonlinear comparison protocol relies on DCF, ensuring no leakage of inputs or outputs. In Appendix H, we prove the security of all proposed secure protocols (e.g., secure comparison, secure training, and secure inference) against semi-honest adversaries under Definition 2 and provide a detailed security analysis. Additionally, in Appendix H, we explicitly analyze the interaction between discretization (equal-width binning) and the indicator-bit leakage bound, and provide empirical leakage assessment through feature reconstruction and attribute inference attacks, confirming that the discretization-induced leakage remains both theoretically bounded and practically negligible.

TABLE IV: Information of datasets

Dataset	Instances	Features	Classes	Source
Iris	150	13	3	UCI
Breast Cancer	699	9	3	UCI
Covertype	581, 012	54	7	UCI
Poker Hand	1,025,010	10	10	UCI
Smoking	55, 692	26	2	Kaggle
Skin	245,057	3	2	Kaggle

Communication overhead. We analyze the communication overhead of two main primitives (secure multiplication and comparison) and FSS-DT in Table III. The table shows FSS-DT achieves lower overhead than the other two works for both training and inference, where the detailed analysis is given in Appendix I. The table shows that our secure multiplication has less communication overhead. FSS-DT only involves more lightweight integer multiplications for the division-free DT. Our comparison protocol has less communication overhead than the RCA-based comparison protocol of Pivot and iASS-based comparison.

VIII. Experiments

A. Experiment Setup

We implement a prototype of FSS-DT[6] using PyTorch and evaluate it on a workstation equipped with an Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz,128GB RAM, and NVIDIA GeForce RTX 3090 Ti, operating Ubuntu 20. Following prior works [8], [42], we utilize a Linux network tool, "tc", to simulate the local-area network (LAN, RTT: 0.1 ms, 2 Gbps) and the wide-area network (WAN, RTT: 40 ms, 400 Mbps) between the two parties. Similarly to previous work [8], [17], arithmetic sharing length is $\ell = 64$ and the security parameter for FSS is $\lambda = 128$.

Dataset. We use 4 real-world datasets from the UCI Machine Learning Repository[7] and 2 real-world vertical datasets of federated learning from Kaggle Repository [8] to evaluate the accuracy and efficiency of the FSS-DT system. The datasets are summarized in Table IV. For UCI datasets, we distribute features of each dataset randomly and vertically between P_0 and P_1. As in previous work [37], we also assume that samples in all datasets are aligned properly in advance. For datasets from Kaggle, P_0 holds one of the two vertical partitions, and P_1 holds the remaining part and the sample labels.

Baselines. We use different baselines to show the effectiveness of our plaintext DFDT algorithm and secure FSS-DT framework. For the plaintext DFDT, we compare with 3 classical DT algorithms: ID3 [43], C4.5 [44], and CART [1]. For FSS-DT, we compare with the privacy-focused frameworks, Pivot [8] and Swan [14]. We evaluate the training time on the CPU, encompassing communication and computation durations. We

[6]https://github.com/NSS-01/FSS-DT.git

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

[8]https://www.kaggle.com/datasets/wonghoitin/datasets-for-federated-learning

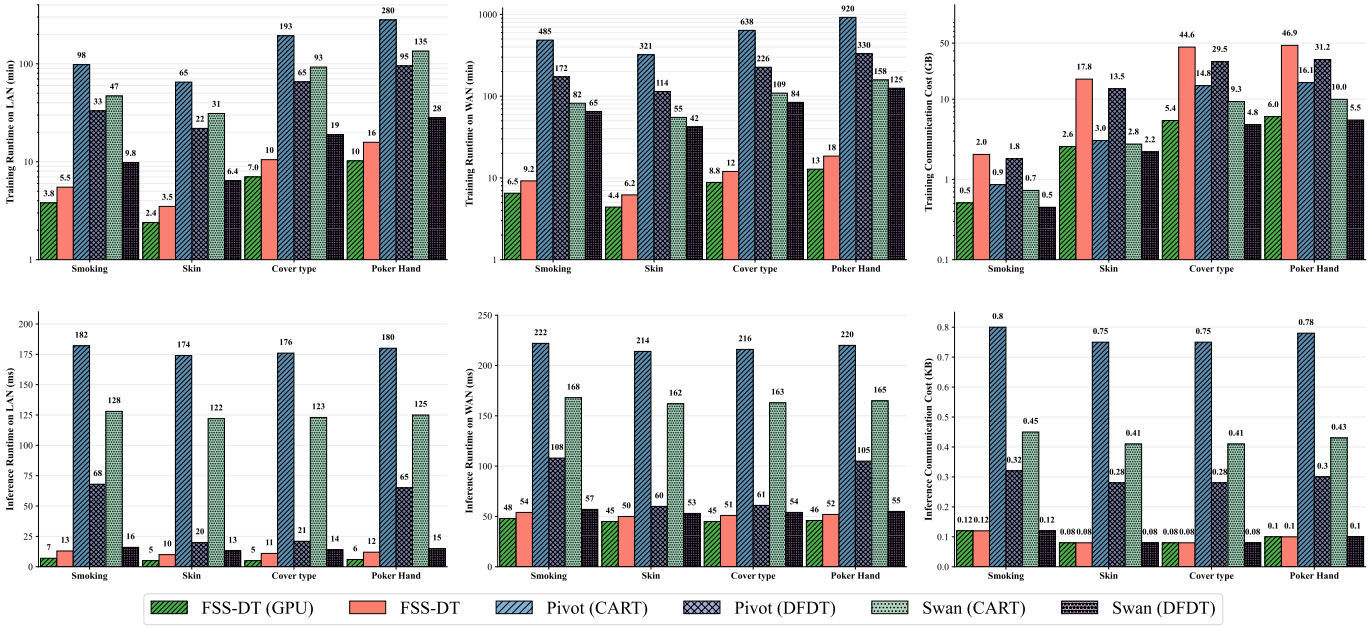


Fig. 4: Performance comparison with baselines

TABLE V: Accuracy under different tree depths

$\mathcal{H}$		Approach	Iris	Breast Cancer	Poker Hand	Smoking	Skin	Cover type
4	Plain-text	ID3	0.95	0.92	0.93	0.90	0.93	0.69
		C4.5	0.98	0.94	0.93	0.88	0.98	0.69
		CART	0.98	0.94	0.96	0.88	0.98	0.70
		DFDT	**0.98**	**0.93**	**0.96**	**0.89**	**0.98**	**0.70**
	Secure	Pivot (CART)	0.97	0.93	0.95	0.85	0.95	0.65
		Pivot (DFDT)	0.98	0.93	0.95	0.87	0.98	0.70
		Swan (CART)	0.97	0.93	0.95	0.85	0.91	0.63
		Swan (DFDT)	0.98	0.93	0.95	0.86	0.98	0.70
		FSS-DT (DFDT)	**0.98**	**0.93**	**0.95**	**0.88**	**0.98**	**0.70**
8	Plain-text	ID3	1.00	0.93	0.99	0.91	0.99	0.70
		C4.5	1.00	0.94	0.99	0.91	0.98	0.71
		CART	1.00	0.93	0.99	0.92	0.99	0.72
		DFDT	**1.00**	**0.93**	**0.99**	**0.92**	**0.99**	**0.73**
	Secure	Pivot (CART)	0.98	0.92	0.99	0.88	0.89	0.53
		Pivot (DFDT)	1.00	0.93	0.99	0.88	0.99	0.73
		Swan (CART)	0.94	0.93	0.99	0.90	0.83	0.61
		Swan (DFDT)	1.00	0.93	0.99	0.91	0.99	0.73
		FSS-DT (DFDT)	**1.00**	**0.93**	**0.99**	**0.91**	**0.99**	**0.73**
16	Plain-text	ID3	1.00	0.94	0.99	0.93	0.99	0.85
		C4.5	1.00	0.94	0.99	0.93	0.99	0.85
		CART	1.00	0.93	0.99	0.94	0.99	0.86
		DFDT	**1.00**	**0.95**	**0.99**	**0.94**	**0.99**	**0.85**
	Secure	Pivot (CART)	0.92	0.89	0.99	0.90	0.86	0.53
		Pivot (DFDT)	1.00	0.95	0.99	0.92	0.99	0.85
		Swan (CART)	0.91	0.89	0.99	0.92	0.83	0.56
		Swan (DFDT)	1.00	0.95	0.99	0.94	0.99	0.85
		FSS-DT (DFDT)	**1.00**	**0.95**	**0.99**	**0.94**	**0.99**	**0.85**

DFDT is our division-free DT. The original DT used in Pivot and Swan is CART.

also evaluate the total communication overhead between the two parties. Besides, we also provide a comparison with the two-party Ents [12] protocol in Appendix J, which represents the best-performing work among the share-input share-output approaches.

B. Accuracy Evaluation

Our division-free DT model is represented as **DFDT**. In Table V, we compare the accuracy of DFDT with ID3, C4.5, and CART in plaintext and compare FSS-DT with Pivot and Swan. For the test, we use multiple datasets with tree depths ($\mathcal{H}$) of 4, 8, and 16. The results indicate that, in the plaintext domain, DFDT achieves comparable or superior accuracy against ID3, C4.5, and CART in most cases, regardless of $\mathcal{H}$ and datasets. We can also see that our FSS-DT has no accuracy loss compared with its plaintext version. Both Pivot and Swan were originally built on CART. Their results show a notable decline in accuracy as the tree depth increases compared to CART in plaintext. That is because CART has division operations, and they process divisions in a manner that results in accuracy loss. To assess how our division-free DT can enhance their accuracy, we replace their underlying DT model with DFDT. The results show that they achieve improved accuracy in all cases, matching values of FSS-DT and the plaintext DFDT.

C. Performance Evaluation

In Fig 4, we compare the training and inference performance of FSS-DT with Pivot and Swan, where the tree depth is set to $\mathcal{H} = 4$. In addition, Fig. 5 presents a comprehensive scalability analysis of FSS-DT during both training and inference phases under WAN and LAN network in Appendix F.

Thanks to our lightweight comparison protocol and the division-free DT algorithm, FSS-DT significantly outperforms Pivot and Swan in both computation and communication on the Poker Hand dataset. For training under LAN and WAN settings, FSS-DT is about $17.5\times$ and $51.1\times$ faster than Pivot, and about $8.4\times$ and $8.8\times$ faster than Swan, respectively. Its training communication cost is also the lowest, requiring only 6.0 GB, which is $2.7\times$ and $1.7\times$ lower than Pivot (16.1 GB) and Swan (10.0 GB). During inference (averaged over 1000 test samples), FSS-DT achieves only 12 ms runtime, making

it approximately $15\times$ and $10.4\times$ faster than Pivot (180 ms) and Swan (125 ms), respectively, while using only 0.1 KB communication, which is about $7.8\times$ and $4.3\times$ lower than Pivot (0.78 KB) and Swan (0.43 KB).

Algorithmic Efficiency vs. Hardware Efficiency. To show the performance gains stem from hardware parallelism or algorithmic innovation, we decouple these factors by analyzing the results across different network environments. As shown in Fig. 4, GPU-based FSS-DT achieves a $2\times$–$3\times$ speedup in LAN settings, largely benefiting from the high-throughput parallel processing of FSS operators. However, in high-latency WAN environments, it maintains a $1.5\times$–$2\times$ performance gain. Since GPU acceleration cannot mitigate inherent network Round-Trip Time (RTT), this sustained improvement in WAN settings directly validates the efficiency of our secure comparison protocol in terms of communication overhead.

Furthermore, we conduct an ablation study by integrating our division-free (DFDT) logic into existing frameworks, namely *Swan* and *Pivot*. While the division-free optimization significantly enhances their performance, our FSS-DT framework continues to outperform both in both training and inference runtimes. This consistent lead is rooted in fundamental communication advantages: compared to *Pivot*'s RAC-based protocol, which incurs an additional cost of $11\ell - 2$ bits per invocation, our FSS-based approach minimizes the transmission volume. Consequently, while GPU hardware increases the computational ceiling, our algorithmic optimizations address the communication bottleneck—the primary constraint in private Decision Tree execution. By reducing bit-complexity, we ensure that GPU resources are utilized effectively rather than idling during network synchronization, representing a true synergy between algorithmic design and hardware capabilities.

D. Microbenchmarks

Since PyTorch supports GPU acceleration, we offer both CPU and GPU versions of FSS-DT's main operations (secure integer multiplication and comparison) and compare their runtimes with secure fixed-point multiplication and existing comparison techniques, as shown in Table VI. Specifically, for an input size of $N = 10^6$ under a LAN network, GPU-based secure integer multiplication takes only 54 ms, achieving a 55% speedup over fixed-point multiplication. In the WAN network with the input size of $N = 10^6$, the secure integer multiplication is 90% faster than the secure fixed-point multiplication. This demonstrates the excellent computation of secure integer multiplication under high-latency communication settings, which is critical for performance at scale. For secure comparison, we reduce key sizes of DCF with the help of "0/1 encoding" and "early termination" techniques. In a WAN with input size $N = 10^6$, our DCF requires only 1654 ms on the GPU and is 67% faster than the original DCF. This performance trend of the operations highlights the fundamental interplay between computational throughput and communication efficiency. The substantial speedup observed in WAN environments underscores that the observed performance improvements are not simply a consequence of raw GPU power. Instead, they are driven by the proposed basic operation's critical role in alleviating high-latency communication bottlenecks, thereby unlocking the GPU's parallel processing capabilities even under adverse network conditions.

TABLE VI: Online performance of basic primitives

Primitive		**Core**	**LAN** (ms)				**WAN** (ms)			
			$N: 10^3$	10^4	10^5	10^6	$N: 10^3$	10^4	10^5	10^6
Integer Mult. (**Ours**)		CPU	1	2	8	82	10	26	125	134
		GPU	1	2	6	54	10	13	103	141
Fixed-point Mult. [8]		CPU	2	3	12.4	182	12	36	185	234
		GPU	2	3	10	84	26	43	163	219
Comp.	RCA-MSB [8]	CPU	191	387	2104	8484	291	1835	6235	12014
		GPU	123	153	863	3389	147	952	4421	9769
	DCF [24]	CPU	221	622	1227	3131	254	1262	2143	4434
		GPU	123	288	623	1806	145	867	1842	2774
	Our DCF	CPU	146	463	926	2547	160	972	1549	3250
		GPU	112	212	418	1103	143	612	1136	1654

IX. Conclusion

This paper presents FSS-DT, a high-performance two-party framework for DT training and inference. To improve efficiency, we designed a new MPC-friendly plaintext DT training algorithm that removes expensive division operations. Besides, we present a lightweight secure comparison protocol based on optimized DCF via the idea of the 0/1-encoding technique. Extensive experiments demonstrate that FSS-DT outperforms the state-of-the-art Pivot and Swan.

References

[1] R. J. Lewis, "An introduction to classification and regression tree (cart) analysis," in *Annual meeting of the society for academic emergency medicine in San Francisco, California*, vol. 14. Citeseer, 2000.

[2] IBM, "Spss:decision trees," https://www.ibm.com/products/spss-statistics/decision-trees.

[3] LLC, "Treeage," https://www.treeage.com/.

[4] R. Cramer, I. Damgård, and U. Maurer, "General secure multi-party computation from any linear secret-sharing scheme," in *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2000, pp. 316–334.

[5] W. Du and Z. Zhan, "Building decision tree classifier on private data," in *Proceedings of the IEEE international conference on Privacy, security and data mining-Volume 14*, 2002, pp. 1–8.

[6] Y. Lindell and B. Pinkas, "Privacy preserving data mining," in *Advances in Cryptology—CRYPTO 2000: 20th Annual International Cryptology Conference Santa Barbara, California, USA, August 20–24, 2000 Proceedings*. Springer, 2000, pp. 36–54.

[7] J. Vaidya, C. Clifton, M. Kantarcioglu, and A. S. Patterson, "Privacy-preserving decision trees over vertically partitioned data," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 2, no. 3, pp. 1–27, 2008.

[8] Y. Wu, S. Cai, X. Xiao, G. Chen, and B. C. Ooi, "Privacy preserving vertical federated learning for tree-based models," *Proceedings of the VLDB Endowment*, vol. 13, no. 12, 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 Transactions on Information Forensics and Security*, vol. 18, pp. 1006–1021, 2022.

[10] A. Akavia, M. Leibovich, Y. S. Resheff, R. Ron, M. Shahar, and M. Vald, "Privacy-preserving decision trees training and prediction," *ACM Transactions on Privacy and Security*, vol. 25, no. 3, pp. 1–30, 2022.

[11] M. Abspoel, D. Escudero, and N. Volgushev, "Secure training of decision trees with continuous attributes," *Proceedings on Privacy Enhancing Technologies*, 2021.

[12] G. Lin, W. Han, W. Ruan, R. Zhou, L. Song, B. Li, and Y. Shao, "Ents: an efficient three-party training framework for decision trees by communication optimization," in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, 2024, pp. 4376–4390.

[13] D. Bhardwaj, S. Saravanan, N. Chandran, and D. Gupta, "Securely training decision trees efficiently," in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, 2024, pp. 4673–4687.

[14] A. Song, K. Cheng, J. Fu, S. Cui, T. Zhang, Z. Chang, and Y. Shen, "Private learning for vertical decision trees: A secure, accurate, and fast realization," *IEEE Transactions on Dependable and Secure Computing*, pp. 1–15, 2025.

[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 Transactions on Information Forensics and Security*, vol. 15, pp. 2914–2929, 2020.

[16] S. Wagh, S. Tople, F. Benhamouda, E. Kushilevitz, P. Mittal, and T. Rabin, "Falcon: Honest-majority maliciously secure framework for private deep learning," *arXiv preprint arXiv:2004.02229*, 2020.

[17] W.-j. Lu, Z. Huang, Q. Zhang, Y. Wang, and C. Hong, "Squirrel: A scalable secure {Two-Party} computation framework for training gradient boosting decision tree," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 6435–6451.

[18] Z. Han, X. Cheng, W. Zhao, J. Fu, Z. He, and S. Su, "Securexgb: A secure and efficient multi-party protocol for vertical federated xgboost," *Proceedings of the ACM on Management of Data*, vol. 3, no. 1, pp. 1–26, 2025.

[19] V. Kolesnikov and T. Schneider, "Improved garbled circuit: Free xor gates and applications," in *Automata, Languages and Programming: 35th International Colloquium, ICALP 2008.* Springer, 2008, pp. 486–498.

[20] Y. Zheng, C. Wang, R. Wang, H. Duan, and S. Nepal, "Optimizing secure decision tree inference outsourcing," *IEEE Transactions on Dependable and Secure Computing*, vol. 20, no. 4, pp. 3079–3092, 2023.

[21] E. Boyle, N. Gilboa, and Y. Ishai, "Function secret sharing: Improvements and extensions," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 1292–1303.

[22] E. Boyle, N. Chandran, N. Gilboa, D. Gupta, Y. Ishai, N. Kumar, and M. Rathee, "Function secret sharing for mixed-mode and fixed-point secure computation," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2021, pp. 871–900.

[23] J. Fu, K. Cheng, A. Song, Y. Xia, Z. Chang, and Y. Shen, "Fss-dbscan: Outsourced private density-based clustering via function secret sharing," *IEEE Transactions on Information Forensics and Security*, 2024.

[24] K. Storrier, A. Vadapalli, A. Lyons, and R. Henry, "Grotto: Screaming fast (2+ 1)-pc or $\mathbb{Z}_{2^n}$ via (2, 2)-dpfs," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, 2023, pp. 2143–2157.

[25] J. Vaidya, B. Shafiq, W. Fan, D. Mehmood, and D. Lorenzi, "A random decision tree framework for privacy-preserving data mining," *IEEE transactions on dependable and secure computing*, vol. 11, no. 5, pp. 399–411, 2013.

[26] E. Boyle, N. Gilboa, and Y. Ishai, "Function secret sharing," in *Annual international conference on the theory and applications of cryptographic techniques*. Springer, 2015, pp. 337–367.

[27] T. Ryffel, P. Tholoniat, D. Pointcheval, and F. Bach, "Ariann: Low-interaction privacy-preserving deep learning via function secret sharing," *Proceedings on Privacy Enhancing Technologies*, 2022.

[28] G. Garimella, B. Goff, and P. Miao, "Computation efficient structure-aware psi from incremental function secret sharing," in *Annual International Cryptology Conference*. Springer, 2024, pp. 309–345.

[29] S. Adams, C. Choudhary, M. De Cock, R. Dowsley, D. Melanson, A. Nascimento, D. Railsback, and J. Shen, "Privacy-preserving training of tree ensembles over continuous data," *Proceedings on Privacy Enhancing Technologies*, 2022.

[30] A. C. Yao, "Protocols for secure computations," in *23rd annual symposium on foundations of computer science*. IEEE, 1982, pp. 160–164.

[31] D. Rathee, M. Rathee, N. Kumar, N. Chandran, D. Gupta, A. Rastogi, and R. Sharma, "Cryptflow2: Practical 2-party secure inference," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 325–342.

[32] X. Guo, K. Yang, X. Wang, W. Zhang, X. Xie, J. Zhang, and Z. Liu, "Half-tree: Halving the cost of tree expansion in cot and dpf," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2023, pp. 330–362.

[33] C. Guo, K. Cheng, J. Fu, R. Fan, Z. Chang, Z. Zhang, and A. Song, "Gfs-cnn: A gpu-friendly secure computation platform for convolutional neural networks," *Journal of Networking and Network Applications*, vol. 3, no. 2, pp. 66–72, 2023.

[34] S. Wang, Y. Zheng, X. Jia, H. Huang, and C. Wang, "Oblivgm: Oblivious attributed subgraph matching as a cloud service," *IEEE Transactions on Information Forensics and Security*, vol. 17, pp. 3582–3596, 2022.

[35] A. Song, J. Fu, X. Mu, X. Zhu, and K. Cheng, "L-secnet: Towards secure and lightweight deep neural network inference," *Journal of Networking and Network Applications*, vol. 3, no. 4, pp. 171–181, 2024.

[36] H.-Y. Lin and W.-G. Tzeng, "An efficient solution to the millionaires' problem based on homomorphic encryption," in *Applied Cryptography and Network Security: Third International Conference, June 7-10, 2005.* Springer, 2005, pp. 456–466.

[37] C. Dong, L. Chen, and Z. Wen, "When private set intersection meets big data: an efficient and scalable protocol," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, 2013, pp. 789–800.

[38] J. Shore and R. Johnson, "Properties of cross-entropy minimization," *IEEE Transactions on Information Theory*, vol. 27, no. 4, pp. 472–482, 1981.

[39] X. Liu, Y. Zheng, X. Yuan, and X. Yi, "Securely outsourcing neural network inference to the cloud with lightweight techniques," *IEEE Transactions on Dependable and Secure Computing*, vol. 20, no. 1, pp. 620–636, 2022.

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

[41] K. Gupta, N. Jawalkar, A. Mukherjee, N. Chandran, D. Gupta, A. Panwar, and R. Sharma, "Sigma: Secure gpt inference with function secret sharing," *Proceedings on Privacy Enhancing Technologies*, 2024.

[42] J. Bai, X. Song, X. Zhang, Q. Wang, S. Cui, E.-C. Chang, and G. Russello, "Mostree: Malicious secure private decision tree evaluation with sublinear communication," in *Proceedings of the 39th Annual Computer Security Applications Conference*, 2023, pp. 799–813.

[43] R. Bhardwaj and S. Vatta, "Implementation of id3 algorithm," *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 3, no. 6, 2013.

[44] J. R. Quinlan *et al.*, "Bagging, boosting, and c4. 5," in *Aaai/Iaai, vol. 1*, 1996, pp. 725–730.

[45] G. Dessouky, F. Koushanfar, A.-R. Sadeghi, T. Schneider, S. Zeitouni, and M. Zohner, "Pushing the communication barrier in secure computation using lookup tables," in *Proceedings of NDSS*, 2017.

[46] P. Yang, Z. L. Jiang, S. Gao, H. Wang, J. Zhou, Y. Jin, S.-M. Yiu, and J. Fang, "Fssnn: communication-efficient secure neural network training via function secret sharing," *Cryptology ePrint Archive*, 2023.

[47] O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game, or a completeness theorem for protocols with honest majority," in *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*, 2019, pp. 307–328.

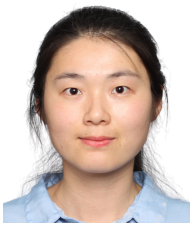
[48] R. Canetti, "Security and composition of multiparty cryptographic protocols," *Journal of CRYPTOLOGY*, vol. 13, pp. 143–202, 2000.

[49] M. Keller, "Mp-spdz: A versatile framework for multi-party computation," in *Proceedings of the 2020 ACM SIGSAC conference on computer and communications security*, 2020, pp. 1575–1590.

Anxiao Song is currently an Assistant Professor at City University of Macau. He received the Ph.D. degree in computer science and technology from Xidian University, Xi'an, Shaanxi, China, in 2025. His research interests include machine learning security and privacy protection.

Shujie Cui is a lecturer at Monash University, Australia, in the Faculty of Information Technology. She received Ph.D. degree in Department of Computer Science, University of Auckland, her M.Sc. degrees in Computer Science from Shandong University, China and the University of Luxembourg, Luxembourg in 2013, and her B.Sc. degree from Shandong University, China in 2011. After obtaining her M.Sc. degree, she worked in the Department of Computer Science at City University of Hong Kong as a research associator. Her research interests include cloud computing, applied cryptography, security and privacy.

Ke Cheng is an Associate Professor in the School of Computer Science and Technology at Xidian University. He received the Ph.D. degree in computer science and technology from Xidian University, Xi'an, Shaanxi, China, in 2022. His research interests include data security and privacy protection.



Jianli Bai received the B.S. degree and the M.S. degree from the School of Computer Science and Technology, Qingdao University, Qingdao, China. She received the Ph.D. degree from University of Auckland, Auckland, New Zealand. She is currently a research fellow at Singapore Management University, Singapore. Her research interests include cloud security, privacy-preserving machine learning, and secure multiparty computation.

Qifan Wang received the B.S. degree from Henan University, Henan, China, the M.S. degree from Central South University, Hunan, China, and the Ph.D. degree from the Department of Computer Science, University of Auckland, New Zealand. He is currently a research fellow on trusted execution at the University of Birmingham, UK. His current research interests include side-channel attacks, privacy-preserving machine learning, applied cryptography, and trusted execution on CPU and GPU.

Shangqi Lai is a Research Scientist at CSIRO's Data61. Before joining CSIRO, he was a Research Fellow at Monash University. Dr Lai obtained his PhD degree at Monash University. His research mainly focuses on the security and privacy issues of cloud systems and aims to systematically adopt privacy-enhancing technologies to guarantee identity/data confidentiality and in critical cloud services. His work has been published in top cyber security venues, such as ACM CCS, NDSS and USENIX Security.

Giovanni Russello is a Professor in the Department of Computer Science at University of Auckland, New Zealand. He received his M.Sc. degree in Computer Science from the University of Catania, Italy in 2000, and his Ph.D. degree from the Eindhoven University of Technology (TU/e) in 2006. After obtaining his Ph.D. degree, he moved to the Policy Group in the Department of Computing at Imperial College London, UK. His research interests include policy-based security systems, privacy and confidentiality in cloud computing, smartphone security, and applied cryptography.

Yulong Shen (Senior Member, IEEE) received the B.S. and M.S. degrees in computer science and the Ph.D. degree in cryptography from Xidian University, Xi'an, China, in 2002, 2005, and 2008, respectively. He is currently a Professor at the School of Computer Science and Technology, Xidian University, where he is also an Associate Director of the Shaanxi Key Laboratory of Network and System Security and a member of the State Key Laboratory of Integrated Services Networks. His research interests include wireless network security and cloud computing security. He has also served on the technical program committees of several international conferences, including ICEBE, INCoS, CIS, and SOWN.