Efficient and Privacy-Preserving Decision Tree Inference via Homomorphic Matrix Multiplication and Leaf Node Pruning

Fukui, Satoshi
Wang, Lihua
Ozawa, Seiichi

(Citation)
Applied Sciences,15(10):5560

(Issue Date)
2025-05-15

(Resource Type)
journal article

(Version)
Version of Record

applied sciences

MDPI

Article

Efficient and Privacy-Preserving Decision Tree Inference via Homomorphic Matrix Multiplication and Leaf Node Pruning

Satoshi Fukui [1,†], Lihua Wang [2,*,†] and Seiichi Ozawa [1,*,†]

[1] Graduate School of Engineering, Kobe University, Kobe 657-8501, Japan; kobesatoshi@outlook.jp
[2] Cybersecurity Research Institute, National Institute of Information and Communications Technology, Tokyo 184-8795, Japan
* Correspondences: lh-wang@nict.go.jp (L.W.); ozawasei@kobe-u.ac.jp (S.O.); Tel.: +81-42-327-7251 (L.W.); +81-78-803-5753 (S.O.)
† These authors contributed equally to this work.

Abstract: Cloud computing is widely used by organizations and individuals to outsource computation and data storage. With the growing adoption of machine learning as a service (MLaaS), machine learning models are being increasingly deployed on cloud platforms. However, operating MLaaS on the cloud raises significant privacy concerns, particularly regarding the leakage of sensitive personal data and proprietary machine learning models. This paper proposes a privacy-preserving decision tree (PPDT) framework that enables secure predictions on sensitive inputs through homomorphic matrix multiplication within a three-party setting involving a data holder, a model holder, and an outsourced server. Additionally, we introduce a leaf node pruning (LNP) algorithm designed to identify and retain the most informative leaf nodes during prediction with a decision tree. Experimental results show that our approach reduces prediction computation time by approximately 85% compared to conventional protocols, without compromising prediction accuracy. Furthermore, the LNP algorithm alone achieves up to a 50% reduction in computation time compared to approaches that do not employ pruning.

Keywords: data mining; decision trees; data security; privacy-preserving

Academic Editors: Grzegorz J. Blinowski and George Drosatos

Received: 10 April 2025
Revised: 7 May 2025
Accepted: 13 May 2025
Published: 15 May 2025

Citation: Fukui, S.; Wang, L.; Ozawa, S. Efficient and Privacy-Preserving Decision Tree Inference via Homomorphic Matrix Multiplication and Leaf Node Pruning. *Appl. Sci.* **2025**, *15*, 5560. https://doi.org/10.3390/app15105560

1. Introduction

It has been a long time since data was referred to as "the oil of the 21st century". Today, many of the world's most valuable companies thrive by collecting and monetizing vast amounts of data. However, such data often include sensitive personal information, making it inaccessible for public or collaborative use—not only due to commercial interests but also to mitigate privacy risks. Despite the significant potential of data sharing in addressing pressing societal challenges such as crime prevention, public health, and elder care, privacy concerns continue to severely limit the practical use of sensitive datasets across institutional boundaries. To address this challenge, *privacy-preserving machine learning (PPML)* has emerged as a key paradigm for enabling data-driven insights without compromising confidentiality.

One of the most prominent PPML frameworks is federated learning (FL) [1], which allows multiple parties to collaboratively train machine learning models without sharing their raw data. Each participant trains a local model and shares only model updates with a central server, preserving data privacy. However, FL assumes that participants have sufficient computational capacity for local training. In many real-world settings,

particularly for devices or institutions with limited resources, local computation must be outsourced, raising new privacy concerns–even under semi-honest threat models.

This paper focuses on another type of PPML scenario in which a local entity securely outsources prediction computations. In our setting, both the data and the model remain encrypted, ensuring the confidentiality of sensitive user input and the intellectual property of the model provider. This setup is especially effective in protecting against model inversion attacks [2], which constitute a well-known vulnerability in machine learning-as-a-service (MLaaS) platforms. To implement secure outsourced prediction, we employ *homomorphic encryption (HE)* [3], particularly in its somewhat homomorphic variant, named *somewhat homomorphic encryption (SHE)* [4], which supports a limited number of encrypted additions and multiplications with significantly improved efficiency over fully homomorphic schemes. This enables practical encrypted inference in real-world scenarios. Among the various machine learning models, decision trees are particularly attractive for privacy-preserving inference due to their interpretability and low computational cost. As a result, a growing body of research has focused on designing cryptographic protocols for *privacy-preserving decision tree (PPDT)* inference that minimize both communication and computational overhead while preserving privacy.

1.1. Related Work

Existing PPDT protocols can be broadly classified based on the cryptographic primitives they employ.

HE-based protocols: HE supports computation over encrypted data without requiring decryption. Akavia et al. [5] used low-degree polynomial approximations to support non-interactive inference with communication costs independent of tree depth. Frery et al. [6], Hao et al. [7], and Shin et al. [8] adopted the TFHE, BFV, and CKKS schemes, respectively, for efficient computation with low multiplicative depth. Cong et al. [9] reported ciphertext size comparisons and proposed homomorphic traversal algorithms across various commonly used HE schemes. Most existing HE-based PPDT protocols are designed for two-party settings, similar to the linear-function-based scheme proposed by Tai et al. [10].

Protocols based on other cryptographic primitives: Zheng et al. [11] employed additive secret sharing in a two-server setting, ensuring that no single party gained access to both the model and the data, while maintaining low communication overhead. MPC-based approaches, such as those reported b Wu et al. [12], combine additive HE with oblivious transfer (OT), offering strong privacy guarantees but often suffering from scalability issues, particularly with deep trees. Differential privacy (DP), although more commonly used during training [13], can complement cryptographic inference methods by adding noise to outputs to mask sensitive patterns. However, DP typically compromises utility and operates orthogonally to encrypted inference techniques.

Within this landscape, several notable HE-based PPDT approaches stand out. Tai et al. [10] proposed a prediction protocol using linear functions and DGK-based integer comparison [14], reducing complexity at the cost of requiring bitwise encryption. Lu et al. [15] enhanced efficiency via the XCMP protocol, based on Ring-LWE SHE, although it struggled with large input bit lengths and unstable ciphertexts. Saha and Koshiba [16] addressed this with SK17, encoding integer bits as polynomial coefficients to support larger comparisons. Wang et al. [17] further improved SK17 by introducing a faster comparison protocol and a non-interactive variant with reduced ciphertext functionality.

Despite these advancements, existing protocols still face challenges in simultaneously achieving high efficiency, scalability, low communication overhead, and full tree structure confidentiality—especially under realistic semi-honest adversary models.

1.2. Our Contributions

To overcome the above-mentioned limitations, we propose a novel three-party PPDT inference protocol that achieves secure, structure-hiding, and communication-efficient decision tree predictions over encrypted data and models. The key innovations of our work are as follows:

- Homomorphic matrix multiplication-based inference: Departing from polynomial approximation and linear path evaluation methods [10], we introduce homomorphic matrix multiplication as the primary operation for encrypted path computation. This novel application supports structured and scalable evaluation of encrypted inputs.
- Leaf node pruning during inference: We propose leaf node pruning at inference time, a novel runtime optimization that reduces the number of nodes involved in computation, significantly improving performance. Unlike traditional model pruning during training, this technique operates during encrypted inference.
- Structure-hiding inference protocol: The decision tree structure—including internal nodes and branching conditions—is fully hidden from both the client and the server. By ensuring that all path computations are homomorphically encrypted, the protocol mitigates leakage risks present in prior PPDT methods.
- Semi-interactive three-party architecture: Our protocol requires only one round of interactive communication between the data holder (client) and the outsourced server. No interaction is required from the model holder during inference. This design enables low-latency, real-world deployment scenarios.

To achieve these properties, we adopt the efficient integer comparison protocol by Wang et al. [17] and the homomorphic matrix multiplication techniques from [18,19] for encrypted path evaluation. By integrating homomorphic matrix multiplication, inference-time leaf node pruning, and tree structure hiding into a semi-interactive three-party framework, our method enables efficient, secure, and practical decision tree inference in untrusted environments. We evaluated the proposed PPDT protocol on standard UCI datasets [20] to demonstrate its performance and practicality.

The algorithm design is detailed in Section 3, the experimental results are presented in Section 4, and the conclusions are provided in Section 5.

2. Preliminaries

In this section, we summarize the related approaches used in our proposal. In Section 2.1, we first review the concept of decision tree and the approach of decision tree classification via the linear function that was introduced by Tai et al. in [10], which was concerned with two kinds of secure computations — comparison at each node of tree and linear function calculation for prediction. We focus on how to improve the efficiency of the above two operations:

- Comparison: We use Wang et al.'s protocol [17] instead of the DGK approach [14] to improve the efficiency of secure comparison in each node.
- Prediction: We use secure inner product/matrix multiplication to replace linear function in order to outsource prediction while maintaining the tree model secret.

Therefore, we recall the secure comparison scheme proposed by Wang et al. [17] in Section 2.2 and the secure matrix multiplication introduced by Doung et al. [18] in Section 2.3. The two schemes are both constructed on the ring-LWE-based homomorphic encryption (see Appendix A for details). The notations in this paper are listed in Notations Section.

2.1. Existing Decision Tree Classification

2.1.1. Decision Tree

Figure 1 presents an example of a decision tree. The decision tree $\mathcal{T} : \mathbb{Z}^N \to \mathbb{Z}$ is a function that takes as input the feature vector $X = [X_0, \ldots, X_i, \ldots, X_{N-1}]$ and outputs the class $\mathcal{T}(X) \in \{c_j\}$ to which X belongs. Generally, the feature vector space is $\mathbb{R}^N$; however, in this paper, we denote it as $\mathbb{Z}^N$ because the input is the encrypted attribute data. This paper's decision tree is a binary tree and consists of two types of nodes: decision nodes and leaf nodes. The decision node D_j outputs a Boolean value $b_j = \mathbb{1}\{X_{\lambda_j} > t_j\}$ where $\lambda_j \in [N]$ is the feature vector's index, and t_j is the threshold. A leaf node L_k holds the output value $\mathsf{score}(L_k) \in \mathcal{C} = \{C_0, \ldots, C_{K-1}\}$. In a binary tree, there are m decision nodes and $m+1$ leaf nodes. For example, Figure 1 shows the case of $K = 3, m = 3, \mathsf{score}(L_1) = C_0$, $\mathsf{score}(L_2) = C_1, \mathsf{score}(L_3) = C_0, \mathsf{score}(L_4) = C_2$.

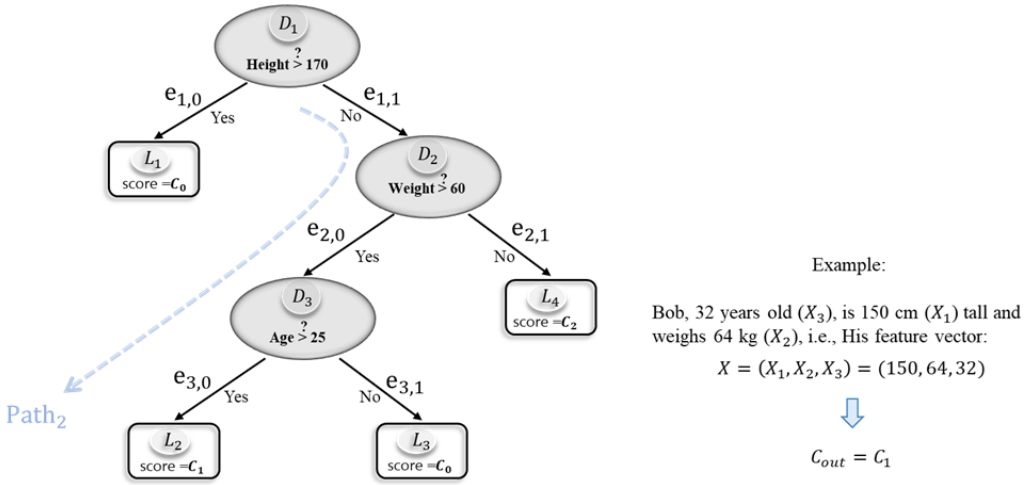

Figure 1. Decision tree example.

2.1.2. Decision Tree Classification via Linear Function [10]

The Boolean output of the decision node D_j, $b_j = 1(0)$ indicates that the next node is the left (right) child. For edges $\mathsf{e}_{j,0}$ and $\mathsf{e}_{j,1}$, respectively connecting to the left child and the right child from the decision node D_j, we define the corresponding edge cost ec as follows:

$$\mathsf{ec}_{j,0} := 1 - b_j, \quad \mathsf{ec}_{j,1} := b_j. \tag{1}$$

From the root node of the decision tree to each leaf node, only one path is determined. We define the path cost pc_k as the sum of the edge costs in Path_k—the path from root node to the leaf node L_k. The edge cost $\mathsf{ec}_{j,0}$ and $\mathsf{ec}_{j,1}$ are determined by b_j, which is the comparison result of decision node D_j. Thereafter, the path cost for each leaf node is defined by a linear function of b_j. For example, the following $\mathsf{pc}_1, \mathsf{pc}_2, \mathsf{pc}_3$, and pc_4 denote the path costs for leaf nodes L_1, L_2, L_3, and L_4 in Figure 1, respectively:

$$\begin{aligned}
\mathsf{pc}_1 &= \mathsf{ec}_{1,0} = 1 - b_1, \\
\mathsf{pc}_2 &= \mathsf{ec}_{1,1} + \mathsf{ec}_{2,0} + \mathsf{ec}_{3,0} = b_1 + (1 - b_2) + (1 - b_3) = 2 + b_1 - b_2 - b_3, \\
\mathsf{pc}_3 &= \mathsf{ec}_{1,1} + \mathsf{ec}_{2,0} + \mathsf{ec}_{3,1} = b_1 + (1 - b_2) + b_3 = 1 + b_1 - b_2 + b_3, \\
\mathsf{pc}_4 &= \mathsf{ec}_{1,1} + \mathsf{ec}_{2,1} = b_1 + b_2.
\end{aligned} \tag{2}$$

The comparison results b_j indicate that the edge cost to the next node is 0, while the edge cost to the other node is 1. As a result, only the path leading to a specific leaf node L_k has a total cost $\mathsf{pc}_k = 0$; all other paths have non-zero costs. The classification result is

returned if and only if $\mathsf{pc}_k = 0$, meaning the output corresponds to the label stored in the leaf node L_k.

This can be illustrated with a concrete example (see Bob in Figure 1): Bob's attributes are height < 170, weight > 60, and age > 25, which yield comparison results $b_1 = 0$, $b_2 = 1$, and $b_3 = 1$, respectively. Substituting these into Equation (2), we compute the path costs as follows: $\mathsf{pc}_1 = \mathsf{pc}_3 = \mathsf{pc}_4 = 1$, and only $\mathsf{pc}_2 = 0$. Therefore, the output for Bob is C_1, the label held by leaf node L_2.

2.2. Secure Comparison Protocol

In this subsection, we describe the protocol proposed by Wang et al. [17], which securely computes the comparison of decision trees.

2.2.1. μ-bit Integer Comparison [14]

Assume that Alice and Bob respectively have two μ-bit integers, a and b, and consider how to compare two integers without revealing the values of a and b between Alice and Bob. Let us define the binary vectors for a and b as follows: $a^{\mathsf{b}} = [a_0, \ldots, a_{\mu-1}]$ and $b^{\mathsf{b}} = [b_0, \ldots, b_{\mu-1}]$. In addition, let us define the following two binary vectors a_i^{b} and b_i^{b} $(1 \le i \le \mu - 1)$ where the first i-bits are the same as those in a^{b} and b^{b} while the remaining upper bits are set to zeros.

$$\begin{aligned} a_i^{\mathsf{b}} &= [a_0, \ldots, a_{i-1}, 0, \ldots, 0], \\ b_i^{\mathsf{b}} &= [b_0, \ldots, b_{i-1}, 0, \ldots, 0]. \end{aligned} \tag{3}$$

To compare the two integers a and b, let us define the following d_i:

$$d_i = w_i + v_i, \tag{4}$$

where

$$\begin{aligned} w_i &= \left\langle a_i^{\mathsf{b}} - b_i^{\mathsf{b}}, a_i^{\mathsf{b}} - b_i^{\mathsf{b}} \right\rangle \ge 0, \\ v_i &= a_i - b_i + 1. \end{aligned} \tag{5}$$

Here, w_i is an inner product for calculating how many bits are different between a_i^{b} and b_i^{b}. Therefore, $w_i = 0$ implies that the first i bits of a and b are the same (i.e., $[a_0, \ldots, a_{i-1}] = [b_0, \ldots, b_{i-1}]$). Next, we look at the $(i+1)$-th bit of a and b when $w_i = 0$ is satisfied. Here, $v_i = a_i - b_i + 1$ could have the three values: 0, 1, or 2. If $(a_i, b_i) = (0, 1)$ (i.e., $a < b$), $v_i = 0$; otherwise, v_i = 1 or 2. That is, if $v_i = 0$, $a < b$ is satisfied under $w_i = 0$; otherwise, $a \ge b$ is satisfied. Therefore, to identify whether $a < b$ is satisfied, we merely need to check if d_i in Equation (4) is 0 for any position of i ($i \in \{0, 1, \ldots, \mu - 1\}$).

2.2.2. Packing Method

For a μ-bit length integer u whose binary vector is denoted as $u^{\mathsf{b}} = [u_0, \ldots, u_{\mu-1}]$, the following packing polynomials are defined:

$$\begin{aligned} \mathsf{poly}_1(u^{\mathsf{b}}) &= \sum_{i=0}^{\mu-1} u_i x^i, \\ \mathsf{poly}_2(u^{\mathsf{b}}) &= \sum_{d=1}^{\mu-1} \sum_{j=0}^{d-1} u_j x^{\mu d - j}. \end{aligned} \tag{6}$$

Using this packing method, we have [17]

$$\mathsf{poly}(d) = (\mathsf{poly}_1(a^{\mathsf{b}}) - \mathsf{poly}_1(b^{\mathsf{b}}))(\mathsf{poly}_2(a^{\mathsf{b}}) - \mathsf{poly}_2(b^{\mathsf{b}}) + \mathsf{poly}_{1\to 3}) + \widetilde{\mathsf{poly}}(\mathbf{1}), \quad (7)$$

where $\mathbf{1} = (1, 1, \ldots, 1)$ denotes the binary vector of the μ-bit integer $2^{\mu} - 1$, and $\mathsf{poly}_{1\to 3} = \sum_{i=1}^{\mu} x^{(\mu-1)(i-1)}$ and $\widetilde{\mathsf{poly}}(\mathbf{1}) = \mathsf{poly}_1(\mathbf{1})\mathsf{poly}_{1\to 3}$ can be computed offline in advance. The coefficient of $x^{i\mu}$ $(i = 0, \ldots, \mu - 1)$ in $\mathsf{poly}(d)$ is d_i.

2.2.3. Secure Comparison Protocol

Wang et al. proposed three enhanced secure comparison protocols in [17]. Here, we recall the most efficient one that uses the packing method defined by Equation (6). There are three participants in this protocol. Alice and Bob who have μ-bit integers a and b, respectively, compare a and b without revealing the data through a server. The server obtains the comparison result, which is the output of the protocol. The protocol is described as follows:

1. Alice generates a secret–public key pair (sk, pk) and sends pk to Bob and the server.
2. Alice and Bob compute
$$[\![a]\!]_i := \mathsf{Enc}(pk, \mathsf{poly}_i(a^{\mathsf{b}})) \quad (8)$$
and
$$[\![b]\!]_i := \mathsf{Enc}(pk, \mathsf{poly}_i(b^{\mathsf{b}})) \quad (9)$$
for $i = 1, 2$, respectively, and send the results to the server.
3. The server computes
$$[\![d]\!] := ([\![a]\!]_1 \ominus [\![b]\!]_1) \otimes ([\![a]\!]_2 \ominus [\![b]\!]_2 \oplus \mathsf{poly}_{1\to 3}) \oplus \widetilde{\mathsf{poly}}(\mathbf{1}). \quad (10)$$
4. The server masks $[\![d]\!]$ in encrypted form using random polynomial $\gamma \leftarrow \mathcal{Z}_p$
$$[\![d']\!] := [\![d]\!] \oplus \gamma \quad (11)$$
and sends $[\![d']\!]$ to Alice.
5. Alice decrypts $[\![d']\!]$
$$d' = \mathsf{Dec}(sk, [\![d']\!]) \quad (12)$$
and sends d' back to the server.
6. The server unmasks d from d' as follows:
$$d = d' - \gamma = \sum_{i=0}^{\mu-1} d_i x^{i\mu} + \text{other terms of degree}, \quad (13)$$
and then verifies whether any $i\mu$-th term for $i = 0, \ldots, \mu - 1$ is 0. If so $a < b$; otherwise, $a \geq b$.

2.3. Ring-LWE-Based Secure Matrix Multiplication

Doung et al. proposed secure matrix multiplication [18] using the packing method proposed by Yasuda et al. [19]. For an ℓ-dimension vector $U = [u_0, u_1, \ldots, u_{\ell-1}]$, the following two polynomials are defined:

$$\begin{aligned} \mathsf{poly}_1(U) &= \sum_{m=0}^{\ell-1} u_m x^m, \\ \mathsf{poly}_t(U) &= -\sum_{m=0}^{\ell-1} u_m x^{n-m}. \end{aligned} \tag{14}$$

Note that for any two vectors $A = [a_0, a_1, \ldots, a_{\ell-1}]$ and $B = [b_0, b_1, \ldots, b_{\ell-1}]$, using the above packing method, we have

$$\mathsf{poly}_1(A) \times \mathsf{poly}_t(B) = \langle A, B \rangle + \text{other terms of degree}, \tag{15}$$

where the constant term of $\mathsf{poly}_1(A) \times \mathsf{poly}_t(B)$ provides the inner product $\langle A, B \rangle$.

Let $\boldsymbol{U}$ be a (k, ℓ) matrix and let $U_1, \ldots, U_k$ denote the row vectors of $\boldsymbol{U}$. For matrix $\boldsymbol{U}$, the packing method is defined as follows:

$$\begin{aligned} \mathsf{poly}_{mat}(\boldsymbol{U}) &= \mathsf{poly}_1(U_1) + \cdots + \mathsf{poly}_1(U_k)x^{(k-1)\ell} \\ &= \sum_{i=1}^{k} \mathsf{poly}_1(U_i)x^{(i-1)\ell}. \end{aligned} \tag{16}$$

Assume that $k\ell \leq n$ (n: dimension of polynomial $x^n + 1$). Letting (k, ℓ) matrix be

$$\boldsymbol{A} = \begin{bmatrix} A_1 \\ \vdots \\ A_k \end{bmatrix} \tag{17}$$

and letting B denote an ℓ-dimensional vector, we have

$$\mathsf{poly}_{mat}(\boldsymbol{A}) \times \mathsf{poly}_t(B) = \sum_{i=1}^{k} \mathsf{poly}_1(A_i) \times \mathsf{poly}_t(B)x^{(i-1)\ell}, \tag{18}$$

where the coefficient of $x^{(i-1)\ell}$ $(i = 1, \ldots, k)$ is $\langle A_i, B \rangle$, the inner product of vectors A_i and B.

3. PPDT Classification Model

In this section, we propose a PPDT classification model that mainly consists of the following two processing parts:

(1) Path cost calculation and (2) secure integer comparison.

The basic idea of the path cost calculation comes from the Tai et al.'s decision tree classification protocol via linear function [10] where a decision tree model is treated as a plaintext under a two-party computation setting (see Section 2.1). In contrast, we extend the Tai et al.'s protocol such that not only input data but also a decision tree model can be encrypted to hide actual contents between a data provider and a model provider. To actualize this, we propose a secure three-party path cost calculation by extending the PPDT protocol proposed by Tai et al. [10] using the integer comparison protocol described in Section 2.2 that was proposed by Wang et al. [17].

3.1. Computation Model

To address the practical considerations of deploying our proposed algorithm in real-world cloud environments, we consider a scenario in which an organization that owns a decision tree classification model outsources the inference task by sending encrypted data to a cloud service provider (e.g., AWS and Google Cloud). Figure 2 illustrates the structure of our computational model, which involves three entities: the *client*, who possesses the

feature vector to be classified; the *model holder*, who holds the trained decision tree; and the *cloud server*, which performs the encrypted computation on behalf of the model holder.

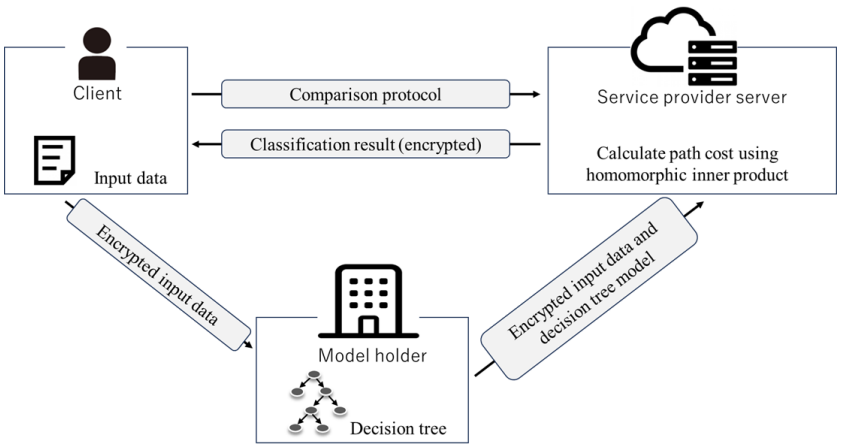

Figure 2. Our computation model.

- The client encrypts the feature vectors and sends them to the model holder, who then encrypts the information needed to calculate the decision tree's path cost and threshold.
- The client's encrypted feature vectors are sent to the server, relaying the model holder, to conceal the information (λ_i) about which elements of the feature vector threshold of the decision node should compare.
- On behalf of the model holder, the server performs the necessary calculation for the decision tree prediction and sends the client's encrypted classification results.
- The client decrypts the information and obtains the classification results.

Even if the organization does not maintain its own servers, decision tree classification with privacy preservation is possible. The concrete process of the protocol is shown in Section 3.3.

A Representative Application of Online Medical Diagnostics

Machine-learned diagnostic models developed by medical institutions represent valuable intellectual property that is central to their competitive advantage. These institutions seek to offer diagnostic services without disclosing proprietary models, and cloud-based computation provides a practical solution to scale such services while reducing local computational costs.

However, concerns related to data confidentiality and model privacy present challenges for real-world deployment. Clients demand privacy-preserving services that do not expose their sensitive health data, while cloud service providers typically prefer not to manage or store sensitive information due to increased compliance and security risks.

Therefore, enabling secure and efficient inference over encrypted data and models aligns with the interests of all stakeholders. Our approach allows computations to be carried out on encrypted inputs and models, thereby mitigating privacy concerns and reducing the operational burdens associated with sensitive data management on cloud platforms.

3.2. Computation of Path Cost by Matrix × Vector Operation

In our method, we encrypt the path costs described in Section 2.1 and send them to the server to keep decision tree model secrets and allow the server to calculate the path costs. Specifically, we transform the path cost $\mathrm{pc}_k = p_{k,0} + p_{k,1}b_1 + \cdots + p_{k,m}b_m$ for a leaf

node L_k. Therefore, we introduce two vectors, P_k and B, that can be used to calculate the path cost:

$$\mathsf{pc}_k = \langle [p_{k,0}, p_{k,1}, \cdots, p_{k,m}], [1, b_1, \cdots, b_m] \rangle = \langle P_k, B \rangle. \tag{19}$$

Here, B is a comparison result vector while P_k is a path vector. We define path matrix $\boldsymbol{P}$ of decision tree $\mathcal{T}$ as follows:

$$\boldsymbol{P} = \begin{bmatrix} P_1 \\ \vdots \\ P_{m+1} \end{bmatrix}. \tag{20}$$

Therefore, it is possible to replace the calculation of the path cost of the decision tree by $\boldsymbol{P} \times B$. To obtain the correct multiplication result for a (k, ℓ) matrix and a vector of length ℓ using Equation (18), the constraint $k\ell \leq n$ must be satisfied. Path matrix $\boldsymbol{P}$ is an $(m+1, m+1)$ matrix; thus $(m+1)^2 \leq n$ must be satisfied. Therefore, we divide the unconstrained path matrix $\boldsymbol{P}$ into several rows and compute each of them. Specifically, $\boldsymbol{P}$ is an $(m+1, m+1)$ matrix; thus, $m' = \lfloor n/(m+1) \rfloor$ rows can be computed. The path matrix is divided into $S = \lceil (m+1)/m' \rceil$ matrices. When $S > (m+1)/m'$, we add vector $I = [1, 0, \ldots, 0]$ for several times so that each partitioned matrix has m' rows, which hides the size of the decision tree. With the above operations, we obtain S $(m', m+1)$ path matrices as follows:

$$\boldsymbol{P}_1 = \begin{bmatrix} P_1 \\ \vdots \\ P_{m'} \end{bmatrix}, \boldsymbol{P}_2 = \begin{bmatrix} P_{m'+1} \\ \vdots \\ P_{2m'} \end{bmatrix}, \ldots, \boldsymbol{P}_S = \begin{bmatrix} P_{(S-1)m'+1} \\ \vdots \\ P_{m+1} \\ I \\ \vdots \\ I \end{bmatrix}. \tag{21}$$

Therefore, the secure matrix $\times$ vector operation in Section 2.3 allows us to securely compute the path cost with the encrypted path matrix.

3.3. Proposed Protocol

The protocol's detailed procedure is illustrated in Figure 3 and Algorithm 1. It comprises eight steps involving data transmission and computation. Let K represent the number of classes in the classification task $\mathcal{C} = \{C_0, C_1, \ldots, C_{K-1}\}$.

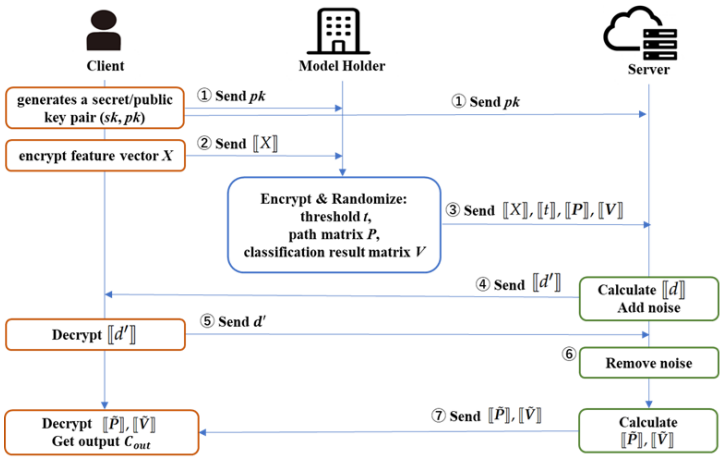

Figure 3. Flowchart of the proposed protocol, where $[\![\cdot]\!]$ denotes the ciphertext of "$\cdot$".

The following provides a step-by-step description of the protocol:

Step 1 (client): The client generates a secret–public key pair (sk, pk) and sends pk to the model holder and server.

Step 2 (client): The client encrypts each element of a feature vector by packing it using Equation (6).

$$\begin{aligned} [\![X_i]\!]_1 &= \mathsf{Enc}(pk, \mathsf{poly}_1(x_i^{\mathsf{b}})), \\ [\![X_i]\!]_2 &= \mathsf{Enc}(pk, \mathsf{poly}_2(x_i^{\mathsf{b}})). \end{aligned} \tag{22}$$

For $i = 1, \ldots, N$, the client sends ciphertexts to the model holder.

Step 3 (model holder): For $j = 1, \ldots, m$, the model holder encrypts threshold t_j by packing it using the Equation (6).

$$\begin{aligned} [\![t_j]\!]_1 &= \mathsf{Enc}(pk, \mathsf{poly}_1(t_j^{\mathsf{b}})), \\ [\![t_j]\!]_2 &= \mathsf{Enc}(pk, \mathsf{poly}_2(t_j^{\mathsf{b}})). \end{aligned} \tag{23}$$

For $k = 1, \ldots, m+1$, the model holder generates path vector P_k multiplied by a random number

$$P'_k = r_k \cdot P_k, \tag{24}$$

and the following classification result vector is generated:

$$V_k := r'_k \cdot P_k + [\mathsf{score}(L_k), 0, \ldots, 0], \tag{25}$$

where $\mathsf{score}(L_k) \in \mathcal{C}, r_k, r'_k \in \mathbb{Z}_p^*$. As described in Section 3.2, the model holder generates S path matrices $P_1, \ldots, P_S$ and S classification result matrices $V_1, \ldots, V_S$. Note that the path vector and classification result vector corresponding to the same path should be in the same matrix row.

For $s = 1, \ldots, S$, path matrix $\boldsymbol{P}_s$ and classification result matrix $\boldsymbol{V}_s$ are encrypted as follows:

$$\begin{aligned} [\![\boldsymbol{P}_s]\!]_{mat} &= \mathsf{Enc}(pk, \mathsf{poly}_{mat}(\boldsymbol{P}_s)), \\ [\![\boldsymbol{V}_s]\!]_{mat} &= \mathsf{Enc}(pk, \mathsf{poly}_{mat}(\boldsymbol{V}_s)), \end{aligned} \tag{26}$$

where Equation (16) is used to compute poly_{mat}. A pair of ciphertexts of the elements of a threshold and its comparative feature vector

$$([\![X_{\lambda_j}]\!]_1, [\![X_{\lambda_j}]\!]_2), ([\![t_j]\!]_1, [\![t_j]\!]_2), \tag{27}$$

and ciphertext pairs of the path matrices and classification result matrices

$$[\![\boldsymbol{P}_s]\!]_{mat}, [\![\boldsymbol{V}_s]\!]_{mat} \tag{28}$$

are sent to the server for $j = 1, \ldots, m$ and $s = 1, \ldots, S$, respectively.

Step 4 (server): The server calculates $[\![d_j]\!]$ in Equation (10) as $a = t_j, b = X_{\lambda_j}$. The server masks $[\![d_j]\!]$ in encrypted form using random polynomial $\gamma_j \leftarrow \mathcal{Z}_p$ and then sends $[\![d'_j]\!]$ to the client.

Step 5 (client): The client decrypts $[\![d'_j]\!]$ and obtains $d'_j = \mathsf{Dec}(sk, [\![d'_j]\!])$, which is then returned to the server.

Step 6 (server): The server obtains d_j from Equation (13) and the comparison result vector $B = [1, b_1, \ldots, b_m]$. If any of the $i\mu$-th $(i = 0, \ldots, \mu - 1)$ coefficients in d_j is zero, then $b_j = 1$; otherwise, $b_j = 0$.

Step 7 (server): The server packs the comparison result vector B using Equation (14) to obtain $\mathsf{poly}_t(B)$. The server calculates

$$\begin{aligned} [\![\tilde{\boldsymbol{P}}_s]\!] &= [\![\boldsymbol{P}'_s]\!]_{mat} \otimes \mathsf{poly}_t(B), \\ [\![\tilde{\boldsymbol{V}}_s]\!] &= [\![\boldsymbol{V}_s]\!]_{mat} \otimes \mathsf{poly}_t(B) \end{aligned} \tag{29}$$

for $s = 1, \ldots, S$ and sends $([\![\tilde{\boldsymbol{P}}_s]\!], [\![\tilde{\boldsymbol{V}}_s]\!])$ to the client.

Step 8 (client): The client obtains the number of path matrix $S = \lceil (m+1)/m' \rceil$ using Equation (21), where $m' = \lfloor n/(m+1) \rfloor$. Then, for $s = 1, \ldots, S$, it decrypts $[\![\tilde{\boldsymbol{P}}_s]\!]$ to obtain a polynomial with randomized non-zero coefficients for the path matrix $\tilde{\boldsymbol{P}}_s$, in which coefficients of $x^{(k-1)(m+1)}$ are k-th path cost $\mathsf{pc}_k \cdot r_k$ according to Equations (19) and (20). Since $\mathsf{pc}_k \cdot r_k = 0 \iff \mathsf{pc}_k = 0$, then it is verified whether any $(k-1)(m+1)$-th term for $k = 1, \ldots, m'$ is 0, which implies the corresponding path cost $\mathsf{pc}_k = 0$. If so, the corresponding leaf of $\mathcal{T}$ is the classification result according to Equation (25), and the client decrypts $[\![\tilde{\boldsymbol{V}}_s]\!]$ and obtains C_{out} from the coefficient of $x^{(k-1)(m+1)}$.

Algorithm 1 Efficient PPDT Inference via Homomorphic Matrix Multiplication

Input: $X = (X_1, \ldots, X_N)$: X_i denotes the i-th feature vector of data X with N features, and $\mathcal{T} = (t_1, \ldots, t_m)$: t_j denotes the j-th threshold of decision tree $\mathcal{T}$ with m decision nodes.
n: degree of polynomial, S: number of path matrices, μ: maximum bit length of X_i and t_j.

Output: Classification result C_{out}

Client generates a secret–public key pair (sk, pk) and sends pk to the model holder and server.
Client encrypts each feature vector of data X (refer to Equation (22))
for $i = 1$ to N **do**
  $[\![X_i]\!]_1 \leftarrow \mathsf{Enc}(pk, \mathsf{poly}_1(x_i^{\mathsf{b}}))$, $[\![X_i]\!]_2 \leftarrow \mathsf{Enc}(pk, \mathsf{poly}_2(x_i^{\mathsf{b}}))$.
end for
return $[\![X]\!] = \{([\![X_i]\!]_1, [\![X_i]\!]_2)\}_{i=1}^{N}$, sends $[\![X]\!]$ to Model holder.
Model holder encrypts threshold t_j (refer to Equation (23))
for $j = 1$ to m **do**
  $[\![t_j]\!]_1 \leftarrow \mathsf{Enc}(pk, \mathsf{poly}_1(t_j^{\mathsf{b}}))$, $[\![t_j]\!]_2 \leftarrow \mathsf{Enc}(pk, \mathsf{poly}_2(t_j^{\mathsf{b}}))$.
end for
return $[\![t]\!] = \{[\![t_j]\!]_1, [\![t_j]\!]_2\}$
Model holder generates path matrix P and classification result matrix V (refer to Equations (24) and (25))
for $k = 1$ to $m+1$ **do**
  generate P_k, select $r_k, r'_k \in \mathbb{Z}_p^*$,
  compute $P'_k \leftarrow r_k \cdot P_k$, $V_k := r'_k \cdot P_k + [\mathsf{score}(L_k), 0, \ldots, 0]$.
end for
return P, V is divided into S matrices $P_1, \ldots, P_S$ and $V_1, \ldots, V_S$, as shown in Equation (21).
The Model Holder encrypts path matrices and classification result matrices (refer to Equations (16) and (26))
for $s = 1$ to S **do**
  $[\![\boldsymbol{P}_s]\!]_{mat} \leftarrow \mathsf{Enc}(pk, \mathsf{poly}_{mat}(\boldsymbol{P}_s))$, $[\![\boldsymbol{V}_s]\!]_{mat} \leftarrow \mathsf{Enc}(pk, \mathsf{poly}_{mat}(\boldsymbol{V}_s))$.
end for
return $[\![\boldsymbol{P}]\!] = \{[\![\boldsymbol{P}_s]\!]\}$, $[\![\boldsymbol{V}]\!] = \{[\![\boldsymbol{V}_s]\!]\}$, and sends $[\![X]\!]$, $[\![t]\!]$, $[\![\boldsymbol{P}]\!]$, $[\![\boldsymbol{V}]\!]$ to Cloud Server.
Server calculates d via one-round communication with the Client (refer to Equation (10))

Algorithm 1 *Cont.*

for $j = 1$ to S **do**
  computes $[\![d_j]\!] \leftarrow ([\![t_j]\!]_1 \ominus [\![X_\lambda]\!]_1) \otimes ([\![t_j]\!]_2 \ominus [\![X_\lambda]\!]_2 \oplus \mathsf{poly}_{1\to 3}) \oplus \widetilde{\mathsf{poly}}(\mathbf{1})$.
  selects $\gamma_j \in \mathcal{Z}_p$, computes $[\![d'_j]\!] \leftarrow [\![d_j]\!] \oplus \gamma_j$, and sends $[\![d'_j]\!]$ to Client.
end for
return $[\![d']\!] = \{[\![d'_j]\!]\}$, and sends $[\![d']\!]$ to Client.
Client decrypts $d' \leftarrow \mathsf{Dec}([\![d']\!], sk)$ and sends it to Server.
Server computes the comparison result vector B (refer to Equation (13))
for $j = 1$ to m **do**
  for $i = 0$ to $\mu - 1$ **do**
    let $b_j = 0$
    if $\exists$ i-th coefficient in $d_j = 0$ **then**
      $b_j \leftarrow b_j + 1$
    end if
  end for
end for
return $B = [1, b_1, \ldots, b_m]$
Server computes encrypted path and classification result matrices (refer to Equations (14) and (29))
for $s = 1$ to S **do**
  $[\![\tilde{\boldsymbol{P}}_s]\!] \leftarrow [\![\boldsymbol{P}'_s]\!]_{mat} \otimes \mathsf{poly}_t(B)$, $[\![\tilde{\boldsymbol{V}}_s]\!] \leftarrow [\![\boldsymbol{V}_s]\!]_{mat} \otimes \mathsf{poly}_t(B)$.
end for
return $([\![\tilde{\boldsymbol{P}}_s]\!], [\![\tilde{\boldsymbol{V}}_s]\!])$ and sends to Client.
Client decrypts and obtains the classification result (refer to Equations (19), (20) and (25))
The default class is $C_{out} = \emptyset$
for $s = 1$ to S **do**
  $\tilde{\boldsymbol{P}}_s \leftarrow \mathsf{Dec}([\![\tilde{\boldsymbol{P}}_s]\!], sk)$
  for $k = 1$ to m' **do**
    if $\exists((k-1)(m+1)$-th coefficients in $P_s = 0)$ **then**
      computes $\tilde{\boldsymbol{V}}_s \leftarrow \mathsf{Dec}([\![\tilde{\boldsymbol{V}}_s]\!], sk)$.
      $C_{out} \leftarrow$ the coefficient of $x^{(k-1)(m+1)}$
    end if
  end for
end for
return C_{out}

3.4. Leaf Node Pruning (LNP)

In this subsection, we describe how to reduce the number of path costs to be calculated. Let the number of leaf nodes of decision tree $\mathcal{T}$ be $m + 1$ and the number of classes to be classified be $K = |\mathcal{C}|$. In this case, we do not calculate the path cost for one class (e.g., C_0) but only the path cost for the other $K - 1$ items. We compute the path cost corresponding to $K - 1$ classes, and if any of them is 0, the output is the class corresponding to the path. If none of them is 0, the output is class C_0 corresponding to the uncalculated path. The detailed procedure is illustrated in Algorithm 2.

Algorithm 2 Efficiency-Enhanced PPDT with LNP (Algorithm 1+LNP)

Input: $X = (X_1, \ldots, X_N)$: X_i denotes the i-th feature vector of data X with N features, and $\mathcal{T} = (t_1, \ldots, t_m)$: t_j denotes the j-th threshold of decision tree $\mathcal{T}$ with m nodes. n: degree of polynomial, S: number of path matrices, μ: maximum bit length of X_i and t_j
$K = |\mathcal{C}|$: number of classes $\mathcal{C} = \{C_0, C_1, \ldots, C_{K-1}\}$

Output: Classification result C_{out}

Given path matrix P and classification result matrix V for $\mathcal{T}$ with $m+1$ leaf nodes.
Model holder prunes leaf nodes that result to the default class
⇒ this process (Lines 2–9 bellow) being inserted between Line 12 and Line 13 of Algorithm 1
for $i = 1$ to $m+1$ **do**
  checks classification value score on leaf node L_i of $\mathcal{T}$,
  if $(\mathsf{score}(L_i) = C_0)$ **then**
    deletes P_i and V_i from P and V, respectively.
    $P \leftarrow P \setminus P_i$, $V \leftarrow V \setminus V_i$
  end if
end for
return P, V
Given encrypted path matrix $[\![\tilde{P}_s]\!]$ and classification result matrix $[\![\tilde{V}_s]\!]$ (Line 36 of Algorithm 1)
Client decrypts and obtains classification result (refer to Equations (19), (20) and (25))
⇒ this process (Lines 11–21 bellow) replaces Lines 37–47 of Algorithm 1
Default class is $C_{out} = C_0$
for $s = 1$ to S **do**
  $\tilde{P}_s \leftarrow \mathsf{Dec}([\![\tilde{P}_s]\!], sk)$
  for $k = 1$ to m' **do**
    if $\exists((k-1)(m+1)$-th coefficients in $P_s = 0)$ **then**
      computes $\tilde{V}_s \leftarrow \mathsf{Dec}([\![\tilde{V}_s]\!], sk)$.
      $C_{out} \leftarrow$ the coefficient of $x^{(k-1)(m+1)}$
    end if
  end for
end for
return C_{out}

Note that the default class C_0 must be securely shared in advance between the client and the model holder. Assigning the majority class as the default is generally more effective. The selection of the class excluded from computation depends on the classification task, as outlined below:

Binary classification tasks (e.g., disease detection, abnormality detection): Only the decision path for class 1 (positive or anomalous) is evaluated. If the result is 0, class 1 is returned; otherwise, class 0 (negative or normal) is output.

Multi-class classification tasks: Only $K-1$ classes are evaluated. If the dataset exhibits class imbalance (e.g., ImageNet), the majority class is assigned as the default. In the absence of such imbalance (e.g., MNIST), one class is randomly selected to serve as the default (i.e., the class not evaluated).

We next use a simple example to illustrate homomorphic matrix multiplication and the leaf node pruning process based on Figure 1 to enhance clarity and aid in reader understanding.

Example 1. *Figure 1 shows a decision tree example that is for three classes* $\mathcal{C} = \{C_0, C_1, C_2\}$, *in which the majority class* C_0 *is assigned as the default. Feature vector* $X := (X_1, X_2, X_3)$, *where* X_1, X_2 *and* X_3 *denote height, weight, and age, respectively. For example, client Bob, whose feature vector* $X = (X_1, X_2, X_3) = (150, 64, 32)$, *tries to use the service to inference his health status. Decision tree* $\mathcal{T}$ *has* $m = 3$ *nodes with threshold* $(t_1, t_2, t_3) = (170, 60, 25)$. *In this case, the comparison result vector can be expressed as* $B = (1, b_1, b_2, b_3)$.

From Figure 1, it can be easily seen that Bob's data should be classified to leaf node L_2 by Path_2. *In the following, we show how to perform classification inference using client Bob's data and the model holder's decision tree model while keeping them encrypted. According to Equations (2) and (19), we have*

$$\begin{aligned} \mathsf{pc}_1 &= \langle P_1, B\rangle = \langle [1, -1, \ 0, \ 0], [1, b_1, b_2, b_3]\rangle, \\ \mathsf{pc}_2 &= \langle P_2, B\rangle = \langle [2, \ 1, -1, -1], [1, b_1, b_2, b_3]\rangle, \\ \mathsf{pc}_3 &= \langle P_3, B\rangle = \langle [1, \ 1, -1, \ 1], [1, b_1, b_2, b_3]\rangle, \\ \mathsf{pc}_4 &= \langle P_4, B\rangle = \langle [0, \ 1, \ 1, \ 0], [1, b_1, b_2, b_3]\rangle. \end{aligned} \tag{30}$$

Therefore,

$$\boldsymbol{P} = \begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \end{bmatrix} = \begin{bmatrix} 1 & -1 & 0 & 0 \\ 2 & 1 & -1 & -1 \\ 1 & 1 & -1 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}. \tag{31}$$

LNP process: *Pruning the leaf nodes with class C_0, i.e., L_1 and L_3, while multiplying P_k by a random non-zero numbers $r_k, r'_k \in \mathbb{Z}_p^*(k = 2, 4)$, we obtain the path matrix*

$$\boldsymbol{P'} = \begin{bmatrix} r_2 P_2 \\ r_4 P_4 \end{bmatrix} = \begin{bmatrix} P'_1 \\ P'_2 \end{bmatrix} = \begin{bmatrix} 2r_2 & r_2 & -r_2 & -r_2 \\ 0 & r_4 & r_4 & 0 \end{bmatrix}, \tag{32}$$

and the corresponding classification result matrix

$$\boldsymbol{V'} = \begin{bmatrix} r'_2 P_2 + [C_1, 0, 0, 0] \\ r'_4 P_4 + [C_2, 0, 0, 0] \end{bmatrix} = \begin{bmatrix} 2r'_2 + C_1 & r'_2 & -r'_2 & -r'_2 \\ C_2 & r'_4 & r'_4 & 0 \end{bmatrix}. \tag{33}$$

Instead of using the original matrices P and V, we only need to use the pruned matrices P' and V' for calculation, thus saving computational cost, as shown in Figure 4.

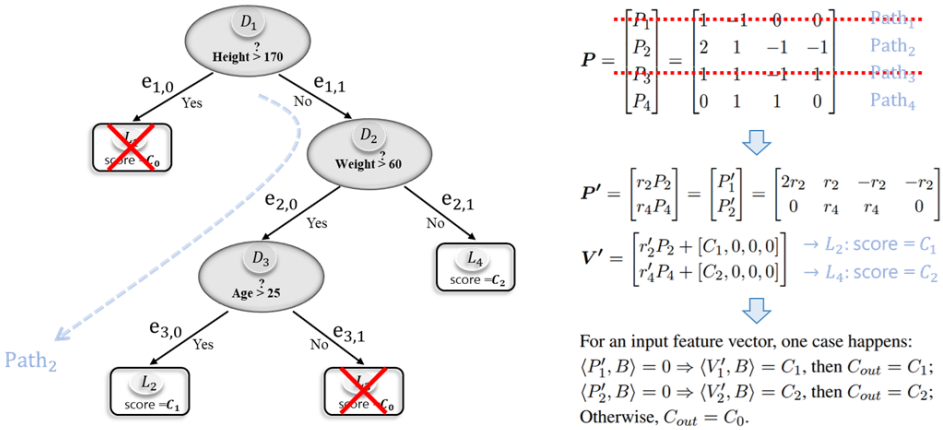

Figure 4. Image for LNP.

The following two processes are run on encrypted statuses.

Comparison on each node: *To check whether $X_i > t_i$ at node D_i, let μ denote the bit length of the two integers X_i and t_i, where $i = 1, 2, 3$. For ease of explanation, we use X_3, t_3 as an example, with a bit length of $\mu_3 = 6$.*

- *Bob encrypts each feature of his own data using Equations (6) and (9).*

$$[\![X]\!] := \{[\![X_i]\!]_1, [\![X_i]\!]_2\}_{i=1}^3 = \{\mathsf{Enc}(pk, \mathsf{poly}_1(X_i)), \mathsf{Enc}(pk, \mathsf{poly}_2(X_i))\}_{i=1}^3, \tag{34}$$

where $X_3 = 32 = (100000)_2$ *in binary,* $\mathsf{poly}_1(32) = 1, \mathsf{poly}_2(32) = x^6 + x^{12} + x^{18} + x^{24} + x^{30}$.

- *Model holder encrypts each threshold of* $\mathcal{T}$ *using Equations (6) and (8).*

$$[\![t]\!] := \{[\![t_i]\!]_1, [\![t_i]\!]_2\}_{i=1}^3 = \{\mathsf{Enc}(pk, \mathsf{poly}_1(t_i)), \mathsf{Enc}(pk, \mathsf{poly}_2(t_i))\}_{i=1}^3, \tag{35}$$

where $t_3 = 25 = (11001)_2$ *in binary,* $\mathsf{poly}_1(25) = x + x^2 + x^5, \mathsf{poly}_2(25) = x^{11} + x^{16} + x^{17} + x^{22} + x^{23} + x^{28} + x^{29}$.

- *Server runs homomorphic operation to obtain polynomials of the comparison result and then adds a random polynomial using Equation (10) over a polynomial ring* $\mathcal{Z}_q$*, which implies* $x^n = -1 \mod q$*, as follows.*

$$\begin{aligned}
[\![d]\!] :=& \{[\![d_i]\!]\}_{i=1}^3 \\
=& \{\mathsf{Enc}\Big(pk, (\mathsf{poly}_1(t_i) - \mathsf{poly}_1(X_i))(\mathsf{poly}_2(t_i) - \mathsf{poly}_2(X_i) + \mathsf{poly}_{1\to 3}) + \widetilde{\mathsf{poly}(\mathbf{1})}\Big)\}_{i=1}^3, \\
[\![d']\!] :=& \{[\![d_i + \gamma]\!]\}_{i=1}^3,
\end{aligned} \tag{36}$$

where $\gamma \in \mathcal{Z}_q$.

- *Bob decrypts* $[\![d']\!]$ *and sends* d' *back to the server. After removing the random polynomial* γ *from* d'*, the server can obtain comparison results from* $d = \{d_i\}_{i=1}^3$ *by checking whether* $\exists$ *any coefficient of* $x^{(i-1)\mu} = 0$*. If so* $X_i > t_i$*, set* $b_i = 1$*; otherwise,* $b_i = 0$.
  For example, $d_3 = 3x^6 + x^{12} + 4x^{18} + 4x^{24} + 4x^{30} + \mathsf{otherterms}$*, which is the coefficient of* $x^0 = 0$*, so* $b_3 = 1$*. Similarly,* $b_1 = 0, b_2 = 0$*; therefore,* $B = [1, b_1, b_2, b_3] = [1, 0, 0, 1]$.

Classification inference:

- *The server runs a multiplication homomorphic operation for Bob using vector B and the following encrypted path matrix* $[\![\boldsymbol{P'}]\!]$ *and classification result matrix* $[\![\boldsymbol{V'}]\!]$ *based on Equations (14), (16) and (18)*

$$\begin{aligned}
[\![\boldsymbol{P'}]\!] &= \mathsf{Enc}(\mathsf{poly}_{mat}(\boldsymbol{P'})) \\
&= \mathsf{Enc}\Big(2r_2 + r_2 x - r_2 x^2 - r_2 x^3 + 0 \cdot x^4 + r_4 x^5 + r_4 x^6 + 0 \cdot x^7\Big), \\
[\![\boldsymbol{V'}]\!] &= \mathsf{Enc}(\mathsf{poly}_{mat}(\boldsymbol{V'})) \\
&= \mathsf{Enc}\Big((2r'_2 + C_1) + r'_2 x - r'_2 x^2 - r'_2 x^3 + C_2 x^4 + r'_4 x^5 + r'_4 x^6\Big)
\end{aligned} \tag{37}$$

to obtain

$$\begin{aligned}
&\mathsf{Enc}(\mathsf{poly}_{mat}(\boldsymbol{P'})) \otimes \mathsf{poly}_t(B) \\
&= \mathsf{Enc}\Big((2r_2 x^0 + r_2 x - r_2 x^2 - r_2 x^3 + r_4 x^5 + r_4 x^6)(1 - x^{n-3} - x^{n-2})\Big) \\
&= \mathsf{Enc}\Big(0 \cdot x^0 + r_4 x^4 + \mathsf{otherterms}\Big) \\
&= \mathsf{Enc}(F_{\mathsf{path}}(x)), \\
&\mathsf{Enc}(\mathsf{poly}_{mat}(\boldsymbol{V'})) \otimes \mathsf{poly}_t(B) \\
&= \mathsf{Enc}\Big(((2r'_2 + C_1) + r'_2 x - r'_2 x^2 - r'_2 x^3 + C_2 x^4 + r'_4 x^5 + r'_4 x^6)(1 - x^{n-3} - x^{n-2})\Big) \\
&= \mathsf{Enc}\Big(C_1 x^0 + (r_4 + C_2) x^4 + \mathsf{otherterms}\Big) \\
&= \mathsf{Enc}(F_{\mathsf{class}}(x)).
\end{aligned} \tag{38}$$

- *Bob decrypts and obtains the polynomial corresponding to a* (k, ℓ)*-dimension path matrix* $F_{\mathsf{path}}(x)$ *and then checks its coefficients of* $x^{(i-1)\ell} (i = 1, \ldots, k)$ *terms. In this example,* $k = 2, \ell = 4$*, so whether the coefficient of* x^0 *or* x^4 *is zero is checked.*
  In fact,

$$F_{\mathsf{path}}(x) = \langle P'_1, B \rangle + \langle P'_2, B \rangle \cdot x^4 + \mathsf{otherterms},$$

$$F_{\mathsf{class}}(x) = (\langle P_1', B\rangle + C_1) + (\langle P_2', B\rangle + C_2)\cdot x^4 + \mathsf{otherterms},$$

and since $\langle P_1', B\rangle = r_2\langle P_2, B\rangle, \langle P_2', B\rangle = r_4\langle P_4, B\rangle$,

the coefficient of x^0 *is zero* $\iff \mathsf{pc}_2 = \langle P_2, B\rangle = 0$, *and*
the coefficient of x^4 *is zero* $\iff \mathsf{pc}_4 = \langle P_4, B\rangle = 0$,

which links to leaf nodes L_2 *and* L_4, *respectively.*
As shown in Equation (38), since the coefficient of x^0 *in the first polynomial is zero, if and only if* $\mathsf{pc}_2 = 0$, *which means X is classified to leaf node* L_2. *Then, the coefficient of* x^0 *in the second polynomial is the classification result; that is,* $C_{out} = C_1$.

3.5. Complexity

The client encrypts each element of the feature vectors with two different packing methods for comparison protocols and sends them to the model holder. The model holder encrypts the threshold for each decision node for $2m$ times. In addition, the path and classification result matrices are encrypted $S = \lceil (m+1)/m' \rceil$ $(m' = \lfloor n/(m+1) \rfloor)$ times, respectively. The model holder sends the pairs of threshold and feature vector elements and the pairs of path and classification result matrices to the server. Next, the server and client cooperate to perform the comparison protocol for m times. In this case, the client performs decryption m times. The server performs homomorphic multiplication of the path and classification result matrices $2S$ times and sends the client results. The client obtains the classification result by decrypting the received path and classification result matrices $2S$ times.

3.6. Security Analysis

For a homomorphic encryption scheme, indistinguishability under chosen-plaintext attack (IND-CPA) is the basic security requirement. The SHE scheme used in the experiments in Section 4 is proven IND-CPA secure under the Ring-LWE assumption. In our proposed approach, as the input of the homomorphic computation consists of ciphertexts, the corresponding IND-CPA-secure SHE schemes will not be weakened.

Consider the entities involved in the protocol one by one (see Figure 3) and assume that they are all honest but curious and that they do not collude with one another.

- Client. The client can decrypt the ciphertext sent from the cloud server in Steps 5 and 7 using the secret key and obtain d' related to the comparison result and the decision tree structure. However, in Step 4, the noise from the cloud server is added for randomization. Therefore, the client receives a polynomial with completely random coefficients in d'. Similarly, $\tilde{\boldsymbol{P}}_s$ is also a polynomial with random coefficients except zero, which leads to the classification $C_{out} \in \mathcal{C}$; that is, the coefficient corresponding to $\tilde{\boldsymbol{V}}_s$. Thus, the client can obtain the classification C_{out} without knowing the decision tree model $\mathcal{T}$
- Model holder. The model holder only obtains the ciphertext of data X in Step 2, and IND-CPA security guarantees the privacy and security of the data.
- Cloud server. The server honestly performs homomorphic operations for the system but is curious about the data provided by the client and the model holder. In Step 3, it receives only encrypted inputs using an SHE scheme that satisfies IND-CPA security, ensuring the confidentiality of both user data and model parameters. Even if the server obtains the comparison result vector $B = [1, b_1, \ldots, b_m]$ from d sent in Step 5, it cannot recover the original data X nor infer meaningful information about the decision tree $\mathcal{T}$ (threshold t, path matrix P, and classification result V). This is due not only to encryption but also to randomization of P using noise terms r_k and r_k' in Step 3, which prevents linkage $b_i (i = 1, \ldots, m)$ to specific nodes of $\mathcal{T}$.

Beyond technical guarantees, our approach also addresses ethical challenges inherent to PPML in cloud environments. Specifically, it supports secure inference while protecting sensitive user data, ensuring model confidentiality and the reducing risks of unauthorized access or data misuse. These safeguards are especially crucial in sensitive domains such as healthcare and finance, where balancing performance with privacy and intellectual property protection is key to responsible AI deployment.

4. Experiments

4.1. Experimental Setup

We implemented the proposed PPDT protocol in C++ where the BFV method [21] in SEAL 3.3 [22] was used as the encryption library. In our implementation, the two comparison protocols were adopted: Wang et al.'s efficiency-enhanced scheme (Protocol-1 in [17]) and Lu et al.'s non-interactive private comparison (XCMP) [15]. The path cost calculation in Section 3.2 was implemented.

Parameter Choice. First, to ensure that the above comparison methods can accurately decrypt the comparison results, the parameters n, q, p, and σ of the two methods must respectively satisfy the following two inequality requirements. Second, they must be selected to meet the 128-bit security level. The following parameters were used to ensure that the proposed model worked accurately and had 128-bit security, which is the currently recommended security level in [23]:

- Efficiency-enhanced scheme [17]
  $n = 2048,\ \log_2 q = 54,\ p = 40{,}961,\ \sigma = 3.2.$
  These satisfy the following inequality: $q > 8p^2\sigma^4n^2 + 4p\sigma n^{2/3}$.
- XCMP [15]
  $n = 8192, \log_2 q = 110,\ p = 8191, \sigma = 3.2.$
  These satisfy the following inequality: $q > p^3\sigma^2 n(\sigma n + 1)^2 + 2p^2\sigma n(\sigma n + 1)$.

Note that $\sigma = 3.2$ is the default noise standard deviation used in Microsoft SEAL. The inequalities involving n, q, p, and σ ensure the correctness of the corresponding comparison schemes. To support a maximum input bit length of ℓ, we chose n to satisfy the conditions from Wang et al. [17] (i.e., $\ell^2 < n$) and Lu et al. [15] (i.e., $\ell < \log_2 n$). Based on these constraints, the comparison protocols could handle 32-bit and 13-bit integers for Wang et al.'s and Lu et al.'s schemes, respectively. This implies that the efficiency-enhanced scheme [17] supports comparisons of larger integers than does XCMP [15], even with a relatively smaller n. Given the selected n, we chose q to ensure a 128-bit security level as recommended in [23] and then determined p such that it satisfied the necessary inequalities while remaining efficient to compute.

The experiments were conducted on a standard PC with the Core i7-7700K (4.20 GHz) processor using a single thread mode. We used scikit-learn, a standard open-source machine learning library in Python 3.7.2, to train the decision trees. In both the BFV method and the comparison protocol by Lu et al. [15], an input value must be represented as an integer $a \in \mathbb{Z}_n$. At every an appropriate number of times when the multiplication is performed, the fractional part of a value must be truncated, and it should be normalized within the range of $[0, 2^{13})$.

In the performance evaluations, the computational time was measured by averaging over 10 trials for the following 8 data sets from the UCI machine learning repository [20]: *Heart Disease* (HD), *Spambase* (SP) (which were used for the benchmark in [15]), *Breast Cancer Wisconsin* (BC), *Nursery* (NS), *Credit-Screening* (CS), *Shuttle* (ST), *EGG EYE State* (EGG), and *Bank Marketing* (BK). The data set information and the parameters of the decision tree used are listed in Table 1.

Table 1. Data set information and the size of trained decision tree.

Dataset	Dataset			Model
	# Data	# Attributes N	# Classes C	# Nodes m
BC	569	30	2	8
ST	43,500	9	7	24
CS	653	15	2	26
HD	720	13	5	35
NS	12,960	8	5	49
SP	4601	57	2	110
EGG	14,980	14	2	724
BK	45,211	16	2	1027

# denotes the number of data, attributes, classes, and decision nodes in the table for brevity.

4.2. Performance Comparisons

4.2.1. Path Cost Calculation

We compared the computational time to obtain the path costs under the three-party protocol when the proposed matrix multiplication in Section 3.2 and the Yasuda et al.'s homomorphic inner product calculation [19] were used. Table 2 shows the computational time for the path costs. We also show the computation time in parentheses when LNP was introduced into the proposed path cost calculation.

In calculating path costs by using the naive inner product, $m+1$ homomorphic multiplications are required. Therefore, as the number of nodes in a tree m increases, the computation time for path costs also increases. On the other hand, when calculating the path cost by matrix multiplication, for the decision tree of $m \leq 35$ of Table 2, the path cost computation time was the same. The reason is that for parameter n, the entire path cost can be calculated with one matrix in the case of $m \leq 35$. Therefore, for datasets other than BK, more than one path cost can be computed with one matrix. The computation time for the path cost was reduced by more than 90% compared to the inner product case. However, in the case of BK, only $\lfloor n/(m+1) \rfloor = \lfloor 2048/1028 \rfloor = 1$ path cost could be calculated with a single matrix multiplication. Hence, there is no difference in the computation time for matrix multiplication and inner product when $\lfloor n/(m+1) \rfloor = 1$.

Table 2. Computation time for path cost calculation in two methods: matrix multiplication of Section 3.2 and the homomorphic inner product calculation of Yasuda et al. [19] that we proposed in [24]. The computation time for the matrix multiplication with the addition of LNP is provided in parentheses. The computation time for LNP was measured as an average of 10 trials for each dataset class.

Dataset	Accuracy	Path Cost (ms)		Total Time (ms)	
		Matrix (+LNP)	Inner Product	Matrix (+LNP)	Inner Product
BC	96.4%	0.25 (0.25)	3.04	44.19 (43.92)	63.58
ST	99.8%	0.25 (0.25)	6.74	66.85 (66.83)	117.51
CS	90.8%	0.25 (0.25)	7.47	77.57 (76.12)	134.64
HD	61.1%	0.25 (0.25)	9.73	98.82 (98.11)	174.34
NS	98.6%	0.50 (0.37)	11.72	130.87 (129.07)	235.04
SP	90.4%	1.72 (0.86)	13.13	321.89 (302.72)	490.04
EGG	69.8%	88.57 (46.20)	179.31	3425.44 (2516.35)	4442.00
BK	88.6%	247.63 (131.14)	252.87	6485.30 (4394.58)	6455.17

With LNP being applied, the path cost computation time remained unchanged for decision trees (e.g., BC, ST, CS, HD) with $m \leq 35$, as all path costs could be computed within a single matrix. However, for larger trees (e.g., NS, SP, EGG, BK), LNP significantly

reduced the computation time—by 26% for NS, 50% for SP, 48% for EGG, and 47% for BK. The effect correlated with the number of classes: datasets with fewer classes (SP, EGG, BK each with 2) benefited more than did NS (5 classes), as fewer classes enhance pruning efficiency. See Table 1 and Figure 5 for dataset-specific details.

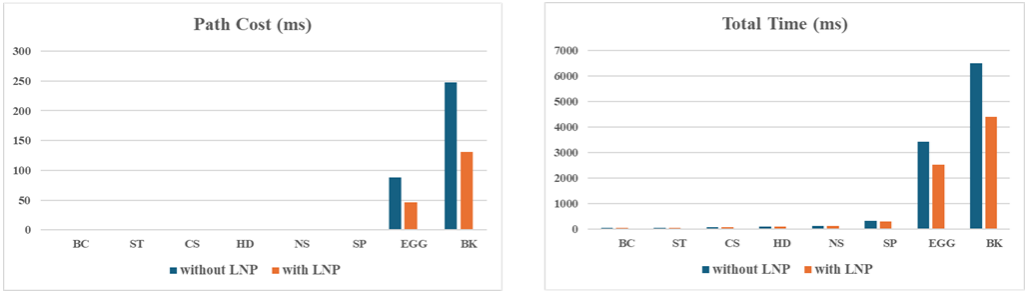

Figure 5. Comparison of computation times with and without LNP across all datasets. For each dataset, the bar chart shows the path cost and total computation times, highlighting the reduction achieved by applying LNP. Datasets with fewer classes (e.g., SP, EGG, BK) showed more significant improvements due to more efficient pruning.

4.2.2. Protocol Efficiency

In this experiment, we used the XCMP of Lu et al. [15] for comparisons. We implemented two protocols and measured their computation times.

1. To evaluate the efficiency of different homomorphic encryption approaches, we also integrated the XCMP scheme into the comparison component of the three-party protocol described in Section 3.3. The path cost computation was performed following the procedure outlined in Steps 6–8 of Section 3.3.
2. A naive two-party protocol using XCMP by Lu et al. computed the result of two-party comparison (see Appendix B). Then, we computed the path cost under encrypt conditions by additive homomorphism.

Table 3 provides the measurement results.

Table 3. Computation time with XCMP [15] in the following scenarios: three-party with matrix multiplication and LNP; two-party with homomorphic addition of XCMP results for computing the path cost.

Dataset	Path Cost (ms)		Total Time (ms)	
	Three-Party	Two-Party	Three-Party	Two-Party
BC	2.17	1577.51	302.19	1958.13
ST	2.17	11,745.70	647.51	12,724.20
CS	2.18	14,905.70	736.95	16,037.90
HD	2.18	27,041.40	961.19	28,298.90
NS	2.18	51,772.50	1321.47	54,268.10
SP	2.19	255,659.00	2996.92	263,008.00
EGG	74.01	10,947,300.00	19,432.90	11,013,300.00
BK	166.74	22,027,200.00	28,821.60	22,130,500.00

In the naive two-party protocol, the path cost calculation is performed only by homomorphic addition, so $O(m^2)$ addition is required. Therefore, the path cost calculation accounted more than 80% of the total time. In contrast, when applied to the proposed protocol, the path cost calculation time was reduced to 0.5% of the total time. As a result, the time reduction is achieved by applying XCMP to the proposed protocol on the experiments' dataset. In particular, the reduction rate was higher for the EGG and BK datasets with a large number of decision nodes.

4.3. Time Complexity

To verify the computational complexity in Section 3.5, we measured the computation time of the proposed protocol for a decision tree constructed with the number of m decision nodes fixed between [10, 1000] using the BK dataset.

Figure 6 provides the experimental results that demonstrated that the computation time was linear with the number of decision nodes. There were several points at which there was a rapid increase in the number of times the model holder's encryption was performed, the number of times the client was compounded, and the number of times the server's path cost was computed. As illustrated in Section 3.5, these computation times depend on $S = \lceil (m+1)/m' \rceil$, where $m' = \lfloor n/(m+1) \rfloor$. Thus, as m increases, the computation time increases rapidly.

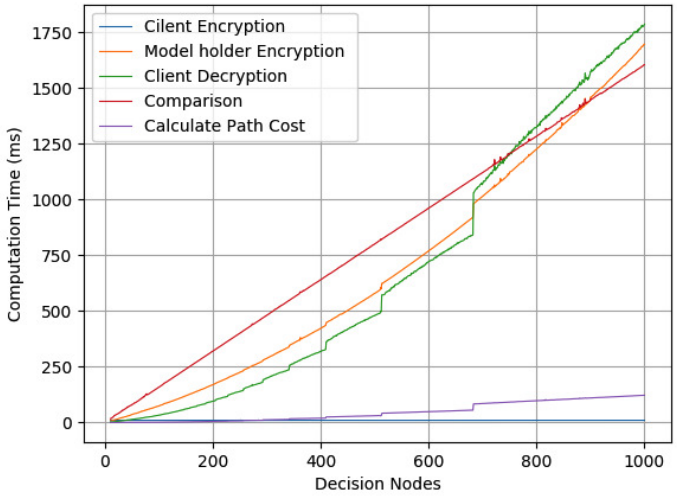

Figure 6. Computation time for decision nodes.

5. Conclusions

In this paper, we proposed a privacy-preserving decision tree prediction protocol that enables secure outsourcing using homomorphic encryption. The proposed method uses the comparison protocol proposed by Wang et al. [17] or Lu et al. [15] to encrypt both feature vectors of a data holder and decision tree parameters of a model holder, and the structure of a decision tree model is hidden against both data holder and outsourced servers by calculating the path cost using the homomorphic matrix multiplication proposed by Dung et al. [18].

Our experimental results demonstrate that the computation time for three-party protocol using XCMP [15] is drastically reduced by more than 85% compared to the naive two-party protocol without a reduction in the prediction accuracy. In addition, we compared the method using homomorphic matrix multiplication [18] with homomorphic inner product [19]. As a result, we confirmed that there were cases where the computation time was reduced by more than 90%. We also proposed a leaf node pruning (LNP) algorithm to accelerate the three-party prediction protocol. The computation time was reduced by up to 50% when the proposed LNP was applied.

However, the proposed method has certain limitations. In particular, while the comparison results at each decision node are revealed to the outsourced server, the server cannot determine which specific feature each node corresponds to. This contrasts with the protocol by Lu et al. [15], which prevents such leakage but incurs significantly higher computational costs. This highlights a fundamental trade-off between security and computational efficiency: our approach enhances scalability and performance at the cost of

limited information leakage. In many practical scenarios, this trade-off may be acceptable; however, minimizing data exposure remains a critical objective. In future work, we will aim to design a more efficient protocol that preserves performance while eliminating any information leakage, thereby strengthening both privacy and practicality.

Author Contributions: Conceptualization, S.F., L.W. and S.O.; methodology, S.F., L.W. and S.O.; software, S.F. and S.O.; validation, S.F. and S.O.; formal analysis, S.F. and L.W.; investigation, S.F. and L.W.; resources, S.O.; data curation, S.F.; writing—original draft preparation, S.F., L.W. and S.O.; writing—review and editing, L.W.; supervision, S.O.; project administration, S.O. and L.W.; funding acquisition, S.O. and L.W. All authors have read and agreed to the published version of the manuscript.

Funding: This work was partially supported by JST CREST (grant number JPMJCR21M1) and JST AIP accelerated program (grant number JPMJCR22U5), Japan.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The original contributions presented in the study are included in the article; further inquiries can be directed to the corresponding author.

Acknowledgments: We would like to thank Takuya Hayashi for the useful discussion.

Conflicts of Interest: The authors declare no conflicts of interest.

Notations

The following notations are used in this manuscript:

$\mathbb{1}\{\cdot\}$	Functions that return 1 if $\cdot$ is true and 0 if false
μ	Maximum bit length of an integer
$A = [a_0, \ldots a_{\ell-1}]$	Vector of length ℓ
$A_d = [a_0, \ldots, a_{d-1}, 0, \ldots, 0]$	d-bit subvector of A
$a^{\mathsf{b}} = [a_0, \ldots, a_{\mu-1}]$	μ-bit binary vector of integer a, where $a_{\mu-1}$ is the least significant bit of integer a.
N	Dimension of the feature vector
$X = [X_0, \ldots, X_{N-1}]$	Feature vector
D_i	Decision node
L_j	Leaf node
λ_i	Index of feature vectors to be compared by decision node D_i
t_i	Threshold of decision node D_i
c_j	Class to be output by leaf node L_j
m	Number of decision nodes in the decision tree
$m+1$	Number of leaf nodes in the decision tree

Parameters being used in the ring-LWE-based encryption schemes:

n	An integer of power of 2 that denotes the degree of polynomial $x^n + 1$. Define a polynomial ring $\mathcal{Z} := \mathbb{Z}[x]/(x^n + 1)$,
q	An integer composed of $q = q_1 \times \cdots \times q_k$ (q_i is a prime number). Define a polynomial ring representing a ciphertext space $\mathcal{Z}_q := \mathbb{Z}_q[x]/(x^n + 1)$,
p	Define a plaintext space $\mathcal{Z}_p := \mathbb{Z}_p[x]/(x^n + 1)$, where p and q are mutually prime natural numbers with the relation $p < q$,
σ	Standard deviation of the discrete Gaussian distribution defining the secret key space $\mathcal{Z}_{(0,\sigma^2)}$. The elements of $\mathcal{Z}_{(0,\sigma^2)}$ are polynomials on the ring $\mathcal{Z}$. Each coefficient is independently sampled from the discrete Gaussian distribution of the variance σ^2.

Appendix A. Ring-LWE-Based Homomorphic Encryption

This study used the ring-LWE-based public key homomorphic encryption library called the Simple Encrypted Arithmetic Library (SEAL) v.3.3 [22] to implement our pro-

tocols. SEAL implements the somewhat homomorphic encryption scheme proposed by Fan et al. [21]. Due to the additive and multiplicative homomorphism, packing plaintexts provides an efficient homomorphic inner product and matrix multiplication calculations. See [21,22] for details on the encryption scheme.

The somewhat homomorphic encryption scheme consists of the following four basic algorithms:

- $\mathsf{ParamGen}(1^\lambda)$: input security parameter 1^λ and output system parameter $pp = (n, q, p, \sigma)$.
- KeyGen: input system parameter pp and output public key pk and secret key sk.
- $\mathsf{Enc}(pk, \cdot)$: input plaintext m and output ciphertext c.
- $\mathsf{Dec}(sk, \cdot)$: input ciphertext c and output plaintext m.

Homomorphic addition and multiplication algorithms are defined by Add and Mul, and the corresponding decryption algorithms are represented by DecA and DecM, respectively. Let $c = \mathsf{Enc}(pk, m)$ and $c' = \mathsf{Enc}(pk, m')$ denote the ciphertexts of the two plaintexts m and m', respectively. Then, the sum and product of m and m' can be calculated as follows.

$$\begin{aligned} \mathsf{Add}(c, c') &= c_{add} && \in \mathcal{Z}_q, \\ \mathsf{DecA}(sk, c_{add}) &= m + m' && \in \mathcal{Z}_p; \\ \mathsf{Mul}(c, c') &= c_{mul} && \in \mathcal{Z}_q, \\ \mathsf{DecM}(sk, c_{mul}) &= mm' && \in \mathcal{Z}_p. \end{aligned} \tag{A1}$$

Hereafter, we write $\mathsf{Enc}(pk, \cdot) := [\![\cdot]\!]$, $\mathsf{Add}(c, c') := c \oplus c'$, and $\mathsf{Mul}(c, c') = c \otimes c'$. The difference between c and c' can be obtained using homomorphic addition, and we define $\mathsf{Sub}(\mathsf{c}, \mathsf{c}') := \mathsf{c} \ominus \mathsf{c} = \mathsf{Add}(\mathsf{c}, -\mathsf{c}')$.

Appendix B. XCMP [15]

Assume that client and server have two ℓ-bit integer values such as a and b, respectively. According to the algorithm in Figure A1, the server computes $[\![C]\!]$ to obtain the comparison result in encrypted form. In this scheme, the comparison result is a constant term in the polynomial C, as shown in $C = \mathbb{1}\{a > b\} + \text{other terms of degree}$. It is possible to perform scalar multiples, additions, and subtractions on the resulting ciphertext $[\![C]\!]$.

Client generates key.
1. (Client) sends $[\![x^a]\!]$ to server.
2. (Server) calculates $\bar{\mu} = 2^{-1} \mod p$, $\mu = -\bar{\mu} \mod p$.
3. (Server) calculates $T = \mu + \mu X + \cdots + \mu X^{n-1}$.
4. (Server) $SampleR \leftarrow \mathcal{Z}_p$ where $R[0] = \bar{\mu}$.
5. (Server) calculates x^{-b}.
6. (Server) calculates $[\![C]\!] = [\![x^a]\!] \otimes x^{-b} \otimes T \oplus R$.

Figure A1. XCMP [15].

References

1. Konečný, J.; McMahan, H.B.; Ramage, D.; Richtárik, P. Federated Optimization: Distributed Machine Learning for On-Device Intelligence. *arXiv* **2016**, arXiv:1610.02527. [CrossRef]
2. Fredrikson, M.; Jha, S.; Ristenpart, T. Model Inversion Attacks that Exploit Confidence Information and Basic Countermeasures. In Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, 12–16 October 2015; Ray, I., Li, N., Kruegel, C., Eds.; ACM: New York, NY, USA, 2015; pp. 1322–1333. [CrossRef]
3. Rivest, R.L.; Dertouzos, M.L. On Data Banks and Privacy Homomorphisms. In *Foundations of Secure Computation*; DeMillo, R., Ed.; Academic Press: Cambridge, MA, USA, 1978; Volume 4, pp. 169–180.
4. Gentry, C. A Fully Homomorphic Encryption Scheme. Ph.D. Thesis, Stanford University, Stanford, CA, USA, 2009.

5. Akavia, A.; Leibovich, M.; Resheff, Y.S.; Ron, R.; Shahar, M.; Vald, M. Privacy-Preserving Decision Trees Training and Prediction. In Proceedings of the Machine Learning and Knowledge Discovery in Databases—European Conference, ECML PKDD 2020, Ghent, Belgium, 14–18 September 2020; Proceedings, Part I; Hutter, F., Kersting, K., Lijffijt, J., Valera, I., Eds.; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2020; Volume 12457, pp. 145–161. [CrossRef]
6. Fréry, J.; Stoian, A.; Bredehoft, R.; Montero, L.; Kherfallah, C.; Chevallier-Mames, B.; Meyre, A. Privacy-Preserving Tree-Based Inference with TFHE. In Proceedings of the Mobile, Secure, and Programmable Networking—9th International Conference, MSPN 2023, Paris, France, 26–27 October 2023; Revised Selected Papers; Bouzefrane, S., Banerjee, S., Mourlin, F., Boumerdassi, S., Renault, É., Eds.; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2023; Volume 14482, pp. 139–156. [CrossRef]
7. Hao, Y.; Qin, B.; Sun, Y. Privacy-Preserving Decision-Tree Evaluation with Low Complexity for Communication. *Sensors* **2023**, *23*, 2624. [CrossRef] [PubMed]
8. Shin, H.; Choi, J.; Lee, D.; Kim, K.; Lee, Y. Fully Homomorphic Training and Inference on Binary Decision Tree and Random Forest. In Proceedings of the Computer Security—ESORICS 2024—29th European Symposium on Research in Computer Security, Proceedings, Part III, Bydgoszcz, Poland, 16–20 September 2024; García-Alfaro, J., Kozik, R., Choras, M., Katsikas, S.K., Eds.; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2024; Volume 14984, pp. 217–237. [CrossRef]
9. Cong, K.; Das, D.; Park, J.; Pereira, H.V.L. SortingHat: Efficient Private Decision Tree Evaluation via Homomorphic Encryption and Transciphering. In Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, 7–11 November 2022; Yin, H., Stavrou, A., Cremers, C., Shi, E., Eds.; ACM: New York, NY, USA, 2022; pp. 563–577. [CrossRef]
10. Tai, R.K.H.; Ma, J.P.K.; Zhao, Y.; Chow, S.S.M. Privacy-Preserving Decision Trees Evaluation via Linear Functions. In Proceedings of the Computer Security—ESORICS 2017—22nd European Symposium on Research in Computer Security, Proceedings, Part II, Oslo, Norway, 11–15 September 2017; Foley, S.N., Gollmann, D., Snekkenes, E., Eds.; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2017; Volume 10493, pp. 494–512. [CrossRef]
11. Zheng, Y.; Duan, H.; Wang, C.; Wang, R.; Nepal, S. Securely and Efficiently Outsourcing Decision Tree Inference. *IEEE Trans. Dependable Secur. Comput.* **2022**, *19*, 1841–1855. [CrossRef]
12. Wu, D.J.; Feng, T.; Naehrig, M.; Lauter, K.E. Privately Evaluating Decision Trees and Random Forests. *Proc. Priv. Enhancing Technol.* **2016**, *2016*, 335–355. [CrossRef]
13. Maddock, S.; Cormode, G.; Wang, T.; Maple, C.; Jha, S. Federated Boosted Decision Trees with Differential Privacy. In Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, 7–11 November 2022; Yin, H., Stavrou, A., Cremers, C., Shi, E., Eds.; ACM: New York, NY, USA, 2022; pp. 2249–2263. [CrossRef]
14. Damgård, I.; Geisler, M.; Krøigaard, M. A correction to 'efficient and secure comparison for on-line auctions'. *Int. J. Appl. Cryptogr.* **2009**, *1*, 323–324. [CrossRef]
15. Lu, W.; Zhou, J.; Sakuma, J. Non-interactive and Output Expressive Private Comparison from Homomorphic Encryption. In Proceedings of the 2018 on Asia Conference on Computer and Communications Security, AsiaCCS 2018, Incheon, Republic of Korea, 4–8 June 2018; Kim, J., Ahn, G., Kim, S., Kim, Y., López, J., Kim, T., Eds.; ACM: New York, NY, USA, 2018; pp. 67–74. [CrossRef]
16. Saha, T.K.; Koshiba, T. An Efficient Privacy-Preserving Comparison Protocol. In Proceedings of the Advances in Network-Based Information Systems, The 20th International Conference on Network-Based Information Systems, NBiS 2017, Ryerson University, Toronto, ON, Canada, 24–26 August 2017; Barolli, L., Enokido, T., Takizawa, M., Eds.; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2018; Volume 7, pp. 553–565. [CrossRef]
17. Wang, L.; Saha, T.K.; Aono, Y.; Koshiba, T.; Moriai, S. Enhanced Secure Comparison Schemes Using Homomorphic Encryption. In Proceedings of the Advances in Networked-Based Information Systems—The 23rd International Conference on Network-Based Information Systems, NBiS 2020, Victoria, BC, Canada, 31 August–2 September 2020; Barolli, L., Li, K.F., Enokido, T., Takizawa, M., Eds.; Advances in Intelligent Systems and Computing; Springer: Berlin/Heidelberg, Germany, 2021; Volume 1264, pp. 211–224. [CrossRef]
18. Duong, D.H.; Mishra, P.K.; Yasuda, M. Efficient Secure Matrix Multiplication Over LWE-Based Homomorphic Encryption. *Tatra Mt. Math. Publ.* **2016**, *67*, 69–83. [CrossRef]
19. Yasuda, M.; Shimoyama, T.; Kogure, J.; Yokoyama, K.; Koshiba, T. Practical Packing Method in Somewhat Homomorphic Encryption. In Proceedings of the Data Privacy Management and Autonomous Spontaneous Security–8th International Workshop, DPM 2013, and 6th International Workshop, SETOP 2013, Revised Selected Papers, Egham, UK, 12–13 September 2013; García-Alfaro, J., Lioudakis, G.V., Cuppens-Boulahia, N., Foley, S.N., Fitzgerald, W.M., Eds.; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2014; Volume 8247, pp. 34–50.
20. Kelly, M.; Longjohn, R.; Nottingham, K. The UCI Machine Learning Repository. Available online: https://archive.ics.uci.edu (accessed on 31 March 2025).

21. Fan, J.; Vercauteren, F. Somewhat Practical Fully Homomorphic Encryption. IACR Cryptology ePrint Archive 2012; p. 144. Available online: https://eprint.iacr.org/2012/144 (accessed on 31 March 2025).
22. *Microsoft SEAL (Release 3.3)*; Microsoft Research: Redmond, WA, USA, 2019. Available online: https://github.com/Microsoft/SEAL (accessed on 31 March 2025).
23. Albrecht, M.; Chase, M.; Chen, H.; Ding, J.; Goldwasser, S.; Gorbunov, S.; Halevi, S.; Hoffstein, J.; Laine, K.; Lauter, K.; et al. *Homomorphic Encryption Security Standard*; Technical report; HomomorphicEncryption.org: Toronto, ON, Canada, 2018.
24. Fukui, S.;Wang, L.; Hayashi, T.; Ozawa, S. Privacy-Preserving Decision Tree Classification Using Ring-LWE-Based Homomorphic Encryption. In Proceedings of the Computer Sequrity Symposium 2019, Nagasaki, Japan, 21–24 October 2019; pp. 321–327.