Optimizing Secure Decision Tree Inference Outsourcing

Yifeng Zheng, Cong Wang, *Fellow, IEEE*, Ruochen Wang, Huayi Duan, and Surya Nepal

Abstract—Outsourcing decision tree inference services to the cloud is highly beneficial, yet raises critical privacy concerns on the proprietary decision tree of the model provider and the private input data of the client. In this paper, we design, implement, and evaluate a new system that allows highly efficient outsourcing of decision tree inference. Our system significantly improves upon the state-of-the-art in the overall online end-to-end secure inference service latency at the cloud as well as the local-side performance of the model provider. We first presents a new scheme which securely shifts most of the processing of the model provider to the cloud, resulting in a substantial reduction on the model provider's performance complexities. We further devise a scheme which substantially optimizes the performance for encrypted decision tree inference at the cloud, particularly the communication round complexities. The synergy of these techniques allows our new system to achieve up to $8\times$ better overall online end-to-end secure inference latency at the cloud side over realistic WAN environment, as well as bring the model provider up to $19\times$ savings in communication and $18\times$ savings in computation.

Index Terms—Privacy preservation, decision trees, cloud, inference service, secure outsourcing

✦

1 INTRODUCTION

Machine learning inference services greatly benefit various kinds of application domains (e.g., healthcare [1], [2], [3], finance [4], [5], and intrusion detection [6], [7]), and its rapid development has been largely facilitated by cloud computing [8], [9], [10] in recent years. In this emerging machine learning based service paradigm, a model provider can deploy a trained model in the cloud, which can then provide inference services to the clients. Outsourcing such services to the cloud promises well-understood benefits for both the model provider (*provider* for short) and client, such as scalability, ubiquitous access, and economical cost.

Among others, decisions trees are one of the most popular machine learning models due to its ease of use and effectiveness, and have been shown to benefit real applications like medical diagnosis [1], [11] and credit-risk assessment [4]. Briefly, a decision tree is comprised of some internal nodes, which are called decision nodes, and some leaf nodes. Each decision node is used to compare a threshold with a certain feature in the feature vector, which is the input to decision tree evaluation, and decide the branch to be taken next. And each leaf node carries a prediction value indicating the inference result. Decision tree inference over an input feature vector is equivalent to tree traversal starting at the root node and terminating when a leaf node is reached.

- *Y. Zheng is with the School of Comptuer Science and Technology, Harbin Institute of Technology, Shenzhen. E-mail: yifeng.zheng@my.cityu.edu.hk.*
- *C. Wang is with the Department of Computer Science, City University of Hong Kong, Hong Kong. E-mail: congwang@cityu.edu.hk.*
- *R. Wang and H. Duan are with the Department of Computer Science, City University of Hong Kong, Hong Kong. E-mail: ruochwang@gmail.com, hduan2-c@my.cityu.edu.hk.*
- *S. Nepal is with Data61, CSIRO, Marsfield NSW 2122, Australia, and also with the Cyber Security Cooperative Research Centre (CRC), Joondalup WA 6027, Australia. Email: surya.nepal@data61.csiro.au.*

While outsourcing the decision tree inference service to the cloud is quite beneficial, it also raises critical privacy concerns on the decision tree model and the input data. On the provider side, it is widely known that training a high quality model requires a significant amount of investment on datasets (possibly sensitive), resources, and specialized skills. It is thus important that the decision tree is not exposed in the service so that the intellectual property as well as the profitability and competitive advantage of the provider could be respected. On the client side, the input feature vector may contain sensitive information, e.g., data in medical applications or financial applications. Overcoming the privacy hurdles is thus of paramount importance to help the provider and client gain confidence in outsourced decision tree inference services. Towards this challenge, a recent research endeavor has been presented by Zheng et al. [12], which represents the state-of-the-art. Their design is based on the lightweight additive secret sharing technique and works under a compatible architecture where two cloud servers from independent cloud providers are employed to jointly conduct the decision tree inference in the ciphertext domain. As an initial endeavor, however, their design is not fully satisfactory and yet to be optimized in performance, as we detail below.

Firstly, the performance complexity of the provider is dependent on the size of the decision tree as well as the feature vector. Specifically, the provider needs to construct and encrypt a binary matrix of size scaling to the product of the number J of decision nodes and the dimension I of the feature vector, so as to support secure feature selection (more details in Section 4.2). Such multiplicative complexity $O(J \cdot I)$ leads to practically unfavorable overhead, which would be further aggravated when the provider needs to outsource multiple decision trees, either for different application domains, or for random forest (an ensemble of decision trees) evaluation.

Secondly, at the cloud side, the phase of secure decision node evaluation has communication rounds linear to the number of bits for value representation. This is unfavorable in the real-world scenario when the two cloud servers are situated in different geographic regions and communicate over WAN, which is a more reasonable setting than local networks given that the two cloud servers are assumed from different trust domains [13].

In light of the above observations, In this paper, we present a new highly efficient design for secure decision tree inference outsourcing which significantly improves upon the state-of-the-art. Our design follows the same architecture of [12], and also makes use of additive secret sharing, yet with significant optimizations to achieve largely boosted performance compared to the state-of-the-art work.

Firstly, we design a new scheme which makes the provider's performance complexity *independent* of the feature vector and thus free of the above multiplicative complexity, through a new re-formulation of the secure feature selection problem. We make an observation that secure feature selection can indeed be treated as an oblivious array-entry read problem, where the encrypted feature vector could be treated as an encrypted array, and the encrypted index value is used to obliviously select an entry from the array. During the procedure, it is required that no information about the feature vector, index value, and selected feature be revealed. With this observation, we propose a new secure feature selection design where the provider only needs to construct and encrypt an indexing vector with size $O(J)$, rather than a matrix of size $O(J \cdot I)$ as proposed in [12].

Secondly, we note that the linear communication round complexity of the prior work [12] in the secure decision node evaluation phase is due to the secure realization of a ripple carry adder for secret-shared comparison, which faces a delay problem due to sequential procedure of carry computation. Our observation from the field of digital circuit design is that the carry delay problem presented in the ripple carry adder can be solved via the advanced carry look-ahead adder [14]. With this observation, we craft a new design for secure decision node evaluation, through digging deep into the logic and computation of the carry look-ahead adder and appropriately organizing the computation in a secure and efficient manner. Our new design achieves a *logarithmic* communication complexity for secure decision node evaluation, gaining superior suitability for practical deployment in WAN environments. As a concrete example, we are able to significantly reduce the rounds of secure decision node evaluation at the cloud servers from 125 to 7 (with the bit length for value representation being 64), greatly reducing the network latency due to interaction rounds. We also provide concrete complexity analysis, showing that such significant gain does not sacrifice computational efficiency in terms of the number of secret-shared multiplications.

The synergy of the above optimization techniques lead to a new highly efficient cryptographic inference protocol which achieves a significant reduction on the overall online inference latency at the cloud, as well as a significant boost in the provider's performance, as compared to the state-of-the-art. We provide formal security analysis of our design under the standard simulation based paradigm. We implement our system and make deployment on the Amazon

Fig. 1. Decision tree illustration.

cloud for performance evaluation over various decision trees with realistic sizes. Compared with the state-of-the-art prior work [12], the overall online end-to-end inference latency at the cloud servers over realistic WAN environment is up to $8\times$ better. In the meantime, our system offers the provider up to $19\times$ savings in communication and $18\times$ savings in computation.

The rest of this paper is organized as follows. Section 2 introduces some preliminaries. Section 3 describes the system model and threat model. Section 4 gives the details of our design. Section 5 provides the security analysis. Section 6 shows the experiments. Section 7 discusses the related work. Section 8 concludes the whole paper.

2 PRELIMINARIES

2.1 Decision Tree Inference

Fig. 1 illustrates a decision tree. As shown, each internal node (called decision node $\mathcal{D}_j$) is associated with a threshold y_j, while each leaf node $\mathcal{L}_z$ is associated with a prediction value u_z indicating the possible inference result. Hence, given a decision tree with J decision nodes and Z leaf nodes, a threshold vector $\mathbf{y} = \{y_0, \cdots, y_{J-1}\}$ and a prediction value vector $\mathbf{u} = \{u_0, \cdots, u_{Z-1}\}$ are derived. The input for decision tree inference is an I-dimensional feature vector, denoted by $\mathbf{x} = \{x_0, \cdots, x_{I-1}\}$. There is an associated input selection mapping $\sigma : j \in \{0, 1, \cdots, J-1\} \to i \in \{0, 1, \cdots, I-1\}$. Decision tree inference with $\mathbf{x}$ as input works as follows. Firstly, the mapping σ is used to select a feature x_i from $\mathbf{x}$ for each $\mathcal{D}_j$. Secondly, starting from the root node, the Boolean function $f(x_{\sigma(j)}) = (x_{\sigma(j)} < y_j)$ is evaluated at each $\mathcal{D}_j$. The evaluation result v_j decides whether to next take the left ($v_j = 0$) or right ($v_j = 1$) branch. Such evaluation terminates when a leaf node is reached. The depth d is the length of the longest path between the root node and a leaf node. Table 1 provides a summary of the key notations. Without loss of generality and as the tree structure should be hidden, we will consider complete binary decision trees in our security design, which is also consistent with previous works [12], [15], [16], [17], [18]. It is noted that dummy nodes can be simply added to make non-complete decision trees complete [15].

TABLE 1
Key Notations

Notation	Description
$\mathbf{x}$	Feature vector
$\mathbf{y}$	Threshold vector
x_i	The i-th feature in the feature vector
y_j	The threshold at decision node $\mathcal{D}_j$
d	Depth of a decision tree
J	Number of decision nodes
I	Dimension of feature vector
Z	Number of leaf nodes
l	Number of bits for value representation
v_j	Evaluation result at decision node $\mathcal{D}_j$
u_z	Prediction value of leaf node $\mathcal{L}_z$

Fig. 2. The system architecture.

2.2 Additive Secret Sharing

Given a value $\alpha \in \mathbb{Z}_{2^l}$, its 2-of-2 additive secret sharing is a pair $([\alpha]_0 = \alpha - r, [\alpha]_1 = r)$, where r is a random value in $\mathbb{Z}_{2^l}$ and the subtraction is done in $\mathbb{Z}_{2^l}$ (i.e., result is modulo 2^l). Given either $[\alpha]_0$ or $[\alpha]_1$, the value α is perfectly hidden. Suppose that two values α and β are secret-shared among two parties $\mathcal{P}_0$ and $\mathcal{P}_1$, i.e., $\mathcal{P}_0$ holds $[\alpha]_0$ and $[\beta]_0$ while $\mathcal{P}_1$ holds $[\alpha]_1$ and $[\beta]_1$. The secret sharing $[\alpha + \beta]$ (resp. $[\alpha - \beta]$) of $\alpha + \beta$ (resp. $\alpha - \beta$) can be computed locally where each party $\mathcal{P}_i$ ($i \in \{0, 1\}$) directly computes $[\alpha + \beta]_i = [\alpha]_i + [\beta]_i$ (resp. $[\alpha - \beta]_i = [\alpha]_i - [\beta]_i$). Multiplication by a constant γ on the value α can also be done locally, i.e., $[\alpha \cdot \gamma]_i = \gamma \cdot [\alpha]_i$. Multiplication over two secret sharings $[\alpha]$ and $[\beta]$ can be supported by using the Beaver's multiplication triple [19], [20]. That is, given the secret sharing of a multiplication triple (t_1, t_2, t_3) where $t_3 = t_1 \cdot t_2$, $[\alpha \cdot \beta]$ can be obtained with one round of interaction between the two parties. In particular, each party $\mathcal{P}_i$ first computes $[e]_i = [\alpha]_i - [t_1]_i$ and $[f]_i = [\beta]_i - [t_2]_i$. Then, $\mathcal{P}_i$ broadcasts $[e]_i$ and $[f]_i$, and recovers e and f. Given this, each party P_i computes $[\alpha \cdot \beta]_i = i \cdot e \times f + [t_1]_i \times f + [t_2]_i \times e + [t_3]_i$.

3 PROBLEM STATEMENT

3.1 System Model

Fig. 2 shows our system architecture, comprised of the provider, the client, and two cloud servers hosted by independent and geographically separated cloud services. Such architecture follows the state-of-the-art prior work [12]. The provider (e.g., a medical institution) owns a decision tree model and provides inference services to the client with the power of cloud computing, i.e., outsourcing the inference service to the cloud. Due to concerns on the proprietary decision tree, the provider would only provide an encrypted version. The client (e.g., a patient) holds a feature vector which may encode private information such as weight, height, heart rate, and blood pressure, and wants to use the intelligent inference service to obtain a prediction about, e.g., her health. As the feature vector is privacy-sensitive, the client is only willing to provide a ciphertext.

The power of the cloud is considered to be supplied by the two cloud servers $\mathcal{C}_0$ and $\mathcal{C}_1$, which jointly provide the secure decision tree inference service. Such a two-server model not only appeared in the prior work [12] on secure outsourced decision tree inference, but has also been recently used to facilitate security designs in different applications [13], [21], [22], [23], [24], with tailored use according to problem specifics. The prominent advantage of such a two-server model is that it allows the provider and the client to go offline after supplying the encrypted inputs, and the secure inference computation is can be fully run at the cloud. Besides, it is compatible with the working paradigm of additive secret sharing, which is applied for the encryption of the decision tree and feature vector. Each cloud server receives shares of the decision tree and feature vector. They jointly do the processing and produce secret-shared inference result which can be retrieved by the client on demand to reconstruct the inference result.

It is noted that as the two cloud servers are assumed from different trust domains, a practical consideration on realistic deployment is that the two cloud servers be situated in different geographic regions and communicate over a WAN, which is a more reasonable setting compared to local networks. In this case, the latency due to interactions between the cloud servers should be taken into account as an important factor in the secure system design.

3.2 Threat Model

Following prior work on secure outsourced decision tree inference as well as most of existing works on privacy-preserving machine learning [12], [22], [23], we consider a semi-honest adversary setting in our system. A semi-honest adversary would honestly follow our protocol, yet attempts to infer private information beyond its access rights. In our system, it is considered that each entity (cloud server, client, provider) might be corrupted by such adversary. For the cloud server entity, we follow previous works under the two-server model ([12], [13], [21], [22], [23], [24]) and assume they are non-colluding. Namely, the two cloud servers are not corrupted by an adversary at the same time.

Consistent with [12], we consider that the values in the client's feature vector $\mathbf{x}$ as well as the inference result (i.e., the prediction value u^* corresponding to $\mathbf{x}$) should be kept private for the client. For the provider, there is a need to keep private the proprietary parameters/information of the decision tree model, including each decision node's threshold y, the mapping σ for feature selection, and the prediction

Input: Secret sharings $[\mathcal{I}_j]$ and $[\mathbf{x}]$.
Output: Secret sharing $[x_{\mathcal{I}_j}]$.

1: Each $\mathcal{C}_m$ creates an array $\mathbf{p}'_m$ where the i-th element is $\mathbf{p}'_m[i] = \mathbf{p}_m[i^*_m] + r_m$. Here, $s_m \leftarrow \mathbb{Z}_{2^l}$ and $r_m \leftarrow \mathbb{Z}_{2^l}$ are random values chosen by $\mathcal{C}_m$; and $i^*_m = ((i + s_m) \bmod 2^l) \bmod I$.
2: $\mathcal{C}_0$ chooses a random value $r \leftarrow \mathbb{Z}_{2^l}$ and sends $[\mathcal{I}_j]'_0 = [\mathcal{I}_j]_0 + r$ to $\mathcal{C}_1$.
3: $\mathcal{C}_1$ computes $[\mathcal{I}_j]'_0 + [\mathcal{I}_j]_1 + s_1 = \mathcal{I}_j + r + s_1$ and sends it to $\mathcal{C}_0$.
4: $\mathcal{C}_0$ removes r and produces $i'_1 = ((\mathcal{I}_j + s_1) \bmod 2^l) \bmod I$.
5: $\mathcal{C}_0$, with i'_1 as input, acts as the receiver to run an OT protocol with $\mathcal{C}_1$ to obtain $\mathbf{p}'_1[i'_1]$.
6: $\mathcal{C}_1$, in a symmetric manner following Steps 2-5, obtains $\mathbf{p}'_0[i'_0]$, where $i'_0 = ((\mathcal{I}_j + s_0) \bmod 2^l) \bmod I$.
7: $\mathcal{C}_1$ chooses a random value $r' \leftarrow \mathbb{Z}_{2^l}$ and sends $\mathbf{p}^*_0[i'_0] = \mathbf{p}'_0[i'_0] - r_1 - r'$ to $\mathcal{C}_0$. Also, $\mathcal{C}_1$ sets r' as its share $[x_{\mathcal{I}_j}]_1$ for the expected feature $x_{\mathcal{I}_j}$.
8: $\mathcal{C}_0$ computes $\mathbf{p}^*_0[i'_0] + \mathbf{p}'_1[i'_1] - r_0 = x_{\mathcal{I}_j} - r'$ and sets the result as its share $[x_{\mathcal{I}_j}]_0$ for $x_{\mathcal{I}_j}$.

Fig. 3. Secure feature selection for a decision node $\mathcal{D}_j$.

value of each leaf node (except the inference result revealed to the client per inference). It is also required that the client learns no additional private information about the decision tree other than the prediction value corresponding to her feature vector. Following prior work [12], [15], we assume some generic meta-parameters as public, including the depth d, the dimension I, and the number l of bits for value representation. We deem dealing with adversarial machine learning attacks out of the scope.

4 OUR PROPOSED DESIGN

4.1 Overview

Our system is aimed at secure outsourcing of decision tree inference with high efficiency. Treating local efficiency as the first priority in our design philosophy, we first aim to shift as much processing as possible to the cloud, reducing the local performance complexities (particularly with respect to the provider in our system). On top of such consideration, we further aim to achieve high efficiency at the cloud through optimizing the processing. Our deign mainly relies on the delicate use of the lightweight additive secret sharing technique, rather than uses resource-intensive garbled circuits and homomorphic encryption.

At a high level, our design is comprised of four phases: secure input preparation, secure feature selection, secure decision node evaluation, and secure inference generation. The secure input preparation phase requires the provider (resp. the client) to encrypt the decision tree (resp. feature vector), and send the ciphertexts to the cloud servers. The secure feature selection phase is to securely select for each decision node a certain feature from the feature vector, in such a way that the cloud servers are oblivious to the mapping between decision nodes and features. The secure decision node evaluation phase securely evaluates the Boolean function at each decision node and output the ciphertext of the evaluation result. The secure inference generation phase is to leverage the ciphertexts of the evaluation results at decision nodes to generate the ciphertext of the ultimate decision tree inference result, which can then be retrieved by the client for recovery. Note that following the previous work [12], we assume that the data-independent multiplication triples are pre-generated and made available to the two cloud servers for use in our design, which can be efficiently achieved via

a semi-honest third party [12], [25]. Our focus is on the latency-sensitive online inference procedure.

4.2 Secure Input Preparation

The client encrypts her feature vector $\mathbf{x}$ via additive secret sharing applied in an element-wise manner. In particular, the client generates two secret shares: $[\mathbf{x}]_0 = \mathbf{x} - \mathbf{r}$ and $[\mathbf{x}]_1 = \mathbf{r}$. For the provider, he encrypts the decision tree as follows. Firstly, the vector $\mathbf{y}$ of thresholds at decision nodes and the vector $\mathbf{u}$ of prediction values at leaf nodes are encrypted through additive secret sharing, with the secret shares $[\mathbf{y}]_0$, $[\mathbf{y}]_1$, $[\mathbf{u}]_0$, and $[\mathbf{u}]_1$ produced. Then, we need to consider how to properly encrypt the mapping σ which is used for feature selection.

We note that the prior work [12] constructs a binary matrix of size $J \times I$ in such a manner that the j-th row vector is a binary vector with I elements where all are 0 except for the one at position $\sigma(j)$ being set to 1. In this way, feature selection is then realized via matrix-vector multiplication between the binary matrix and the feature vector, which can be securely supported under additive secret sharing. Unfortunately, such an approach imposes on the provider multiplicative $O(J \cdot I)$ performance complexity which depends on the number J of decision nodes as well as the dimension I of the feature vector.

Differently, in order to minimize the costs of the provider, our new insight is to instead construct an index vector $\mathcal{I}$ which is comprised of the selection index values for decision nodes and thus the complexity only depends on the number J of decision nodes, i.e., $O(J)$. The selection index value for a decision node $\mathcal{D}_j$ is represented as $\mathcal{I}_j \in [0, I-1]$. The secure usage of this index vector will be described shortly in the phase of secure feature selection. The provider also encrypts this index vector via additive secret sharing and produces $[\mathcal{I}]_0$ and $[\mathcal{I}]_1$. After the above processing, the provider sends the shares $[\mathbf{y}]_m$, $[\mathbf{u}]_m$, and $[\mathcal{I}]_m$ to each cloud server $\mathcal{C}_m$ ($m \in \{0, 1\}$).

4.3 Secure Feature Selection

Upon receiving the shares of the client's feature vector and the provider's decision tree, the cloud servers first perform secure feature selection which produces the secret sharing of a certain feature for each decision node, based on the

Fig. 4. Illustration of a ripple carry adder logic for MSB computation.

$$G^* = G'' + G'P''$$
$$P^* = P''P'$$

(a)

$(G_0^3, P_0^3) \quad G_0^3 \to c_7$

(b)

Fig. 5. (a) Illustration of the defined binary operator; (b) Illustration of carry calculation over 8-bit inputs under the carry look-ahead adder.

encrypted index vector $\mathcal{I}$. Note that hereafter all arithmetic operations are conducted by default in the ring $\mathbb{Z}_{2^l}$, unless otherwise stated.

We now describe how secure feature selection is achieved in our design. For each decision node $\mathcal{D}_j$, the processing of secure feature selection would require as input the shares of the client' feature vector $\mathbf{x}$ and the shares of the corresponding index value $\mathcal{I}_j$ in the index vector $\mathcal{I}$. The output is the secret sharing of the selected feature $x_{\mathcal{I}_j}$ for the decision node $\mathcal{D}_j$.

To accomplish this functionality for our secure outsourced decision tree services, we make an observation that this indeed can be treated as oblivious array-entry read, where the encrypted feature vector could be treated as an encrypted array, and the encrypted index value is used to obliviously select an entry from the array. We leverage this observation and identify that an approach from a very recent work [26] is suited for our scenario. Using this approach as a basis, we devise the scheme for secure feature selection, which is given in Fig. 3.

The intuition is as follows. For ease of notation, we let $\mathbf{p}_0$ (resp. $\mathbf{p}_1$) denote the share of the feature vector $[\mathbf{x}]_0$ (resp. $[\mathbf{x}]_1$) at the cloud server $\mathcal{C}_0$ (resp. $\mathcal{C}_1$). Each cloud server $\mathcal{C}_m$ ($m \in \{0, 1\}$) first creates a new array $\mathbf{p}'_m$ which is derived from $\mathbf{p}_m$ by shifting its indices and entries under fixed random values (per feature selection). Then, given the secret sharing of a target index value $\mathcal{I}_j$, $\mathcal{C}_0$ engages in an interaction with $\mathcal{C}_1$ so as to receive the entry located at the shifted index $((\mathcal{I}_j + s_1) \bmod 2^l) \bmod I$ in $\mathbf{p}'_1$ which corresponds to $\mathcal{I}_j$. Since the random value s_1 in the shifted index is only known to $\mathcal{C}_1$ and the corresponding entry is masked by a random value r_1, $\mathcal{C}_0$ is oblivious to the original index $\mathcal{I}_j$ as well as the share $\mathbf{p}_1[\mathcal{I}_j]$ held by $\mathcal{C}_1$. Similarly, $\mathcal{C}_1$

obtains the entry located at the shifted index in $\mathbf{p}'_0$ which corresponds to $\mathcal{I}_j$, while learning no information about the plain index $\mathcal{I}_j$ and the share of the corresponding entry value held by $\mathcal{C}_0$. Next, the two cloud servers engage in an interaction which essentially performs secret re-sharing so as to obtain the secret sharing of the selected feature $x_{\mathcal{I}_j}$.

4.4 Secure Decision Node Evaluation

With the secret sharings of the threshold y_j and selected feature $x_{\mathcal{I}_j}$ at each decision node $\mathcal{D}_j$, the cloud servers now perform secure decision node evaluation. As this basically requires secure comparison of secret-shared values, the prior work [12] transforms the problem of secure decision node evaluation to a simplified bit extraction problem in the secret sharing domain. The key idea is to securely extract the most significant bit (MSB) of the subtraction result $\Delta = y_j - x_{\mathcal{I}_j}$ as the evaluation result at decision node $\mathcal{D}_j$.

Despite the effectiveness, their solution is limited in that it poses linear $O(l)$ round complexity. This would lead to high performance overhead in the realistic scenario where the two cloud servers are situated in different geographic regions and connected over WAN, which is a more reasonable setting than local networks given that the two cloud servers are assumed to be non-colluding [13]. The basic idea in [12] is to implement a l-bit full adder logic in the secret sharing domain. Specifically, the shares of the difference value Δ at the two cloud servers are represented in bitwise form respectively. Then, a l-bit full adder logic is applied to add in the secret sharing domain the two binary inputs in a bitwise manner, where carry bits are calculated and propagated, and finally produce the MSB of the difference value Δ. We note that the work [12] uses a classical and standard adder logic called ripple carry adder, as shown in Fig. 4.

Input: Secret sharings $[y_j]$ and $[x_{\mathcal{I}_j}]$.
Output: Secret sharing $\langle v_j \rangle$.

1: $\mathcal{C}_m$ computes $[\Delta]_m = [y_j]_m - [x_{\mathcal{I}_j}]_m$.
   // Secure MSB extraction (with $l = 64$ assumed; $\langle \cdot \rangle$ denotes sharing over $\mathbb{Z}_2$)

2: Let a (resp. b) represent the share $[\Delta]_0$ (resp. $[\Delta]_1$), with the bit string being $a_{l-1}, \cdots, a_0$ (resp. $b_{l-1}, \cdots, b_0$). Let $\langle a_q \rangle$ be defined as $(\langle a_q \rangle_0 = a_q, \langle a_q \rangle_1 = 0)$ and $\langle b_q \rangle$ as $\{\langle b_q \rangle_0 = 0, \langle b_q \rangle_1 = b_q\}$, where $q \in [0, l-1]$. Also, let $\langle w_q \rangle$ be defined as $\{\langle w_q \rangle_0 = a_q, \langle w_q \rangle_1 = b_q\}$.
   // Setup round for secure carry computation (SCC):

3: Compute $\langle G_q \rangle = \langle a_q \rangle \cdot \langle b_q \rangle$, for $q \in [0, l-1]$

4: Compute $\langle P_q \rangle = \langle a_q \rangle + \langle b_q \rangle$, for $q \in [0, l-1]$
   // SCC round 1 (with $l = 64$ as example):

5: Compute $(\langle G_0^1 \rangle, \langle P_0^1 \rangle) = (\langle G_0 \rangle, \langle P_0 \rangle)$

6: For $k \in \{1, \cdots, 31\}$
   a) Compute $(\langle G_k^1 \rangle, \langle P_k^1 \rangle) = (\langle G_{2 \cdot k} \rangle, \langle P_{2 \cdot k} \rangle) \tilde{\diamond} (\langle G_{2 \cdot k-1} \rangle, \langle P_{2 \cdot k-1} \rangle)$
   // SCC round 2:

7: For $k \in \{0, \cdots, 15\}$
   a) Compute $(\langle G_k^2 \rangle, \langle P_k^2 \rangle) = (\langle G_{2 \cdot k+1}^1 \rangle, \langle P_{2 \cdot k+1}^1 \rangle) \tilde{\diamond} (\langle G_{2 \cdot k}^1 \rangle, \langle P_{2 \cdot k}^1 \rangle)$
   // SCC round 3:

8: For $k \in \{0, \cdots, 7\}$
   a) Compute $(\langle G_k^3 \rangle, \langle P_k^3 \rangle) = (\langle G_{2 \cdot k+1}^2 \rangle, \langle P_{2 \cdot k+1}^2 \rangle) \tilde{\diamond} (\langle G_{2 \cdot k}^2 \rangle, \langle P_{2 \cdot k}^2 \rangle)$
   // SCC round 4:

9: For $k \in \{0, \cdots, 3\}$
   a) Compute $(\langle G_k^4 \rangle, \langle P_k^4 \rangle) = (\langle G_{2 \cdot k+1}^3 \rangle, \langle P_{2 \cdot k+1}^3 \rangle) \tilde{\diamond} (\langle G_{2 \cdot k}^3 \rangle, \langle P_{2 \cdot k}^3 \rangle)$
   // SCC round 5:

10: For $k \in \{0, 1\}$
   a) Compute $(\langle G_k^5 \rangle, \langle P_k^5 \rangle) = (\langle G_{2 \cdot k+1}^4 \rangle, \langle P_{2 \cdot k+1}^4 \rangle) \tilde{\diamond} (\langle G_{2 \cdot k}^4 \rangle, \langle P_{2 \cdot k}^4 \rangle)$
   // SCC round 6:

11: Compute $\langle G_0^6 \rangle = \langle G_1^5 \rangle + \langle G_0^5 \rangle \cdot \langle P_1^5 \rangle = \langle c_{l-1} \rangle$

12: Compute $\langle v_j \rangle = \langle w_{l-1} \rangle + \langle c_{l-1} \rangle$.

Fig. 6. Secure evaluation of a decision node.

In the ripple carry adder, for each full adder, the two bits that are to be added are available instantly. However, each full adder has to wait for the carry input to arrive from its previous adder. This means that the carry input for the full adder producing the MSB should wait after the carry has rippled through all previous full adders. Note that computing the carry output of each full adder in the secret sharing domain requires interactions between the two cloud servers, thus leading to $O(l)$ round complexity.

Our design follows [12] in terms of the same strategy of secure MSB bit extraction for secure decision node evaluation, yet aims to reduce the round complexity. Through the design introduced below, we manage to reduce the round complexity from linear to logarithmic. Our observation is that the use of the more advanced carry look-ahead adder can solve the carry delay problem presented in the ripple carry adder [14]. At a high level, a carry look-ahead adder is able to calculate the carry in advance based on only the input bits. It works as follows. Firstly, two terms are defined for the carry look-ahead adder: the carry generate signal G_i and the carry propagate signal P_i, where $G_i = a_i \cdot b_i$ and $P_i = a_i + b_i$. Note that these two terms are only based on the input bits and can be computed instantly given the input bits. Then, the original carry calculation $c_{i+1} = a_i \cdot b_i + (a_i + b_i) \cdot c_i$ as in the ripple carry adder can then be re-formulated as $c_{i+1} = G_i + P_i \cdot c_i$. Such re-formulation allows a carry to be computed without waiting for the carry to ripple through all previous stages, as demonstrated by the following example (a 4-bit carry look-ahead adder):

1) $c_1 = G_0 + P_0 \cdot c_0 = G_0$;
2) $c_2 = G_1 + P_1 \cdot c_1 = G_1 + P_1 \cdot G_0$;
3) $c_3 = G_2 + P_2 \cdot c_2 = G_2 + P_2 \cdot (G_1 + P_1 \cdot G_0)$;
4) $c_4 = G_3 + P_3 \cdot c_3 = G_3 + P_3 \cdot (G_2 + P_2 \cdot (G_1 + P_1 \cdot G_0))$.

It can be seen that each carry can be computed without waiting for the calculation of all previous carries.

With the application of the carry look-ahead adder for MSB computation for secure decision node evaluation, we only need to focus on the calculation of the carry c_{l-1} (e.g., c_3 in the above example for 4-bit inputs). That is, after computing c_{l-1}, the MSB can be derived via $MSB = a_{l-1} + b_{l-1} + c_{l-1}$. Note that $c_{l-1} = G_{l-2} + P_{l-2} \cdot G_{l-3} + \cdots + P_{l-2} \cdots P_1 \cdot G_0$. We now need to consider how to properly organize the computation of the carry c_{l-1} so that we could effectively achieve $O(\log_2 l)$ round complexity. Our observation is that such a computation can be supported by forming a binary tree over the carry generate terms, carry propagate terms, and a binary operator $\diamond$ (illustrated in Fig. 5(a)) defined as: $(G^*, P^*) = (G'', P'') \diamond (G', P')$, where $G^* = G'' + G' \cdot P''$ and $P^* = P'' \cdot P'$. For demonstration of this idea, we show in Fig. 5(b) an example on how the carry bit essential for MSB computation can be computed in a recursive manner based on a tree structure, in the case of

---

Input: Secret sharing $\langle v_j \rangle$ for each decision node $\mathcal{D}_j$ and $[u_z]$ for each leaf node $\mathcal{L}_z$.
Output: Inference result u^* for the client.

1: For each $\langle v_j \rangle$, //Conversion from $\mathbb{Z}_2$ to $\mathbb{Z}_{2^l}$.
   a) Let $\langle v_j \rangle$ be defined as $\{\langle v_j \rangle_0 = t_1, \langle v_j \rangle_1 = t_2\}$.
   b) Let $[t_1]$ over $\mathbb{Z}_{2^l}$ be defined as $\{[t_1]_0 = t_1, [t_1]_1 = 0\}$ and $[t_2]$ as $\{[t_2]_0 = 0, [t_2]_1 = t_2\}$.
   c) $\mathcal{C}_0$ and $\mathcal{C}_1$ compute $[v_j] = [t_1] + [t_2] - 2 \cdot [t_1] \cdot [t_2]$ in $\mathbb{Z}_{2^l}$.
2: For each decision node $\mathcal{D}_j$,
   a) $\mathcal{C}_0$ sets $[E_j^L]_0 = 1 - [v_j]_0$ and $\mathcal{C}_1$ sets $[E_j^L]_1 = [v_j]_1$. This produces the secret-shared value of the left outgoing edge.
   b) $\mathcal{C}_m$ sets $[E_j^R]_m = [v_j]_m$ as the secret-shared value of the right outgoing edge.
3: For each leaf node $\mathcal{L}_z$, $\mathcal{C}_0$ and $\mathcal{C}_1$ compute a secret-shared polynomial term $[g_z]$ by multiplying the secret-shared values of all edges on its path.
4: $\mathcal{C}_0$ and $\mathcal{C}_1$ compute $\sum_z [g_z] \cdot [u_z] = [u^*]$, i.e., the secret sharing of the inference result u^*.
5: Client can retrieve $[u^*]$ and reconstruct u^* as the inference result for the feature vector $\mathbf{x}$.

Fig. 7. Secure inference generation.

8-bit inputs. As shown, a pair of the carry generate and propagate terms is put as a leaf node of the tree. Then, the processing is done upwards attributing to each internal node the value corresponding to the application of the operator $\diamond$ between its two children. Such processing leads to $\lceil \log_2 l \rceil$ rounds in computing the essential carry c_{l-1}.

For simplicity of illustration, we show here the details for the case of 4-bit inputs to concretely demonstrate the computation. In the first round, the following terms are computed: $(G_1^1, P_1^1) = (G_2, P_2) \diamond (G_1, P_1)$, $(G_0^1, P_0^1) = (G_0, P_0)$. In the second round, the following term is computed: $(G_0^2, P_0^2) = (G_1^1, P_1^1) \diamond (G_0^1, P_0^1)$. Based on the definition of the binary operator $\diamond$, we first have $G_1^1 = G_2 + G_1 \cdot P_2, P_1^1 = P_2 \cdot P_1$. Then, we have $G_0^2 = G_1^1 + G_0^1 \cdot P_1^1 = G_2 + G_1 \cdot P_2 + G_0 \cdot P_2 P_1$, which corresponds to the carry c_3.

With all the above insights in mind, we now elaborate on how to support secure decision node evaluation by taking advantage of the carry look-ahead adder in the ciphertext domain. From the definition of the operator $\diamond$, it is noted that only addition and multiplication are required (in $\mathbb{Z}_2$). So it is easy to see this operator can be securely realized in the ciphertext domain via secret-shared addition and multiplication. We denote the secure realization of the operator as $(\langle G^* \rangle, \langle P^* \rangle) = (\langle G'' \rangle, \langle P'' \rangle) \tilde{\diamond} (\langle G' \rangle, \langle P' \rangle)$, where $\langle \cdot \rangle$ denotes secret sharing in $\mathbb{Z}_2$. Note that each call of the operator needs 2 parallel secret-shared multiplications.

The details of secure decision node evaluation are provided in Fig. 6. It is noted that for simplicity and without loss of generality, we demonstrate the procedure assuming that $l = 64$, which is the practical parameter setting to be used in our experiments, and also consistent with the state-the-art work [12]. Also, to clearly show the computation that can be done in parallel in each round of secure carry computation, we intentionally avoid the use of nested loops. From the procedure shown in Fig. 6, we can see that for the practical setting $l = 64$, only 7 rounds are required through our design for obtaining the secret sharing $\langle v \rangle$ of the comparison result, in comparison with 125 rounds in the state-of-the-art work [12].

We also point out that the improvement on communication rounds does not sacrifice the computation efficiency

in terms of number of multiplications (in $\mathbb{Z}_2$). Through analysis, our new design requires $3l - 5$ (187 for $l = 64$) multiplications while the prior work requires $3l - 5$ (187 for $l = 64$) multiplications as well. We remark that although in principle the carry look-ahead adder has higher circuit complexity than the ripple carry adder when computing *all* carries is required, our design only needs the computation of the essential carry c_{l-1}. This accounts for why our new design does not enlarge the computation cost in secure decision node evaluation compared with [12].

4.5 Secure Inference Generation

With the secret-shared evaluation results available at each decision node, we now describe how to leverage them to enable the two cloud servers to generate the encrypted inference result. We note that there are two approaches on how to use the decision node evaluation results [12]: a path cost-based approach and a polynomial-based approach. From the perspective of client cost, the main difference between these approaches is that the path cost-based approach imposes on the client high communication complexity exponentially scaling with the tree depth, while the polynomial-based approach only incurs constant $O(1)$ and minimal communication cost (just two shares) for the client.

Given that high local efficiency is our first priority, our system makes use of the polynomial-based approach. This approach works as follows. Starting from the root node, the left outgoing edge of each decision node $\mathcal{D}_j$ is assigned the value $1 - v_j$ (denoted as $E_j^L = 1 - v_j$), while the right outgoing edge is assigned the value v_j (denoted as $E_j^R = v_j$). Then, a term g_z is computed for each path by multiplying the edge values of that path. As mentioned before, only the term for the path leading to the leaf node carrying the inference result will have the value 1, and all other terms are 0. Then, we can proceed by multiplying each term g_z with the prediction value u_z of each corresponding leaf node and computing the sum, i.e., $u^* = \sum_z u_z g_z$, which will lead to the expected inference result u^*.

We use the decision tree in Fig. 1 as an example to concretely demonstrate the computation. Firstly, there are four terms $\{g_z\}_{z=1}^4$ given that the depth is 2 and thus four paths/leaf nodes. These terms are computed as follows:

$g_1 = (1 - v_1) \cdot (1 - v_2)$, $g_2 = (1 - v_1) \cdot v_2$, $g_3 = v_1 \cdot (1 - v_3)$, and $g_4 = v_1 \cdot v_3$. Suppose that the feature vector is evaluated along the path of the leaf node $\mathcal{L}_3$. We have $v_1 = 1$ and $v_3 = 0$, and so we have $g_1 = 0 \cdot (1 - v_2) = 0$, $g_2 = 0 \cdot v_2 = 0$, $g_3 = 1 \cdot (1 - 0) = 1$, and $g_4 = 1 \cdot 0 = 0$. It can be seen that all the terms except the term g_3 corresponding to $\mathcal{L}_3$ has zero value. So we have $\sum_{z=1}^{4} u_z g_z = u_3$, obtaining the expected inference result. The secure realization of this approach in our system for secure inference generation basically follows that of [12]. For completeness, we give the details of secure inference generation in Fig. 7, which realizes polynomial mechanism introduced above in the secret sharing domain.

5 SECURITY ANALYSIS

We define and prove the security of our protocol following the standard simulation-based paradigm. We start with defining the ideal functionality which captures the desired security properties for outsourced decision tree inference, with regard to the threat model mentioned above. We then give the formal security definition under the ideal functionality and show that our protocol securely realizes the ideal functionality. In what follows, we define the ideal functionality for the secure outsourced decision tree inference service targeted in this paper.

Definition 1. *The ideal functionality $\mathcal{F}_{\mathsf{SecODT}}$ of the outsourced decision tree inference service is formulated as follows.*

- *Input. The input to the $\mathcal{F}_{\mathsf{SecODT}}$ consists of the decision tree $\mathcal{T}$ from the provider and the feature vector $\mathbf{x}$ from the client. The two cloud servers $\mathcal{C}_0$ and $\mathcal{C}_1$ provide no input to the $\mathcal{F}_{\mathsf{SecODT}}$.*
- *Computation. Upon receiving the above input, the $\mathcal{F}_{\mathsf{SecODT}}$ performs decision tree inference and produces the inference result denoted as $\mathcal{T}(\mathbf{x})$.*
- *Output. The $\mathcal{F}_{\mathsf{SecODT}}$ outputs the inference result $\mathcal{T}(\mathbf{x})$ to the client, and outputs nothing to the provider and cloud servers.*

Definition 2. *A protocol Π securely realizes the $\mathcal{F}_{\mathsf{SecODT}}$ in the semi-honest adversary setting with static corruption if the following guarantees are satisfied:*

- ***Corrupted provider.*** *A corrupted and semi-honest provider learns nothing about the values in the client's feature vector $\mathbf{x}$. Formally, a probabilistic polynomial time (PPT) simulator $\mathsf{Sim}_{\mathcal{P}}$ should exist so that $\mathsf{View}_{\mathcal{P}}^{\Pi} \overset{c}{\approx} \mathsf{Sim}_{\mathcal{P}}(\mathcal{T})$, where $\mathcal{P}$ denotes the provider and $\mathsf{View}_{\mathcal{P}}^{\Pi}$ refers to the view of $\mathcal{P}$ in the real-world execution of the protocol Π.*
- ***Corrupted cloud server.*** *A corrupted and semi-honest cloud server $\mathcal{C}_m$ $(m \in \{0, 1\})$ learns no information about the client's feature vector $\mathbf{x}$ and the provider's decision tree $\mathcal{T}$. Formally, a PPT simulator $\mathsf{Sim}_{\mathcal{C}_i}$ should exist such that $\mathsf{View}_{\mathcal{C}_m}^{\Pi} \overset{c}{\approx} \mathsf{Sim}_{\mathcal{C}_m}$, where $\mathsf{View}_{\mathcal{C}_m}^{\Pi}$ denotes the view of the cloud server $\mathcal{C}_m$ in the real-world execution of the protocol Π. Note that the two cloud servers have no input and output according to the $\mathcal{F}_{\mathsf{SecODT}}$. Since they are non-colluding, $\mathcal{C}_0$ and $\mathcal{C}_1$ cannot be corrupted by the adversary at the same time.*
- ***Corrupted client.*** *A corrupted and semi-honest client learns no information about the provider's decision tree other than generic meta-parameters as stated before. Formally, a PPT simulator $\mathsf{Sim}_{\mathcal{U}}$ should exist such that*

$\mathsf{View}_{\mathcal{U}}^{\Pi} \overset{c}{\approx} \mathsf{Sim}_{\mathcal{U}}(\mathbf{x}, \mathcal{T}(\mathbf{x}))$, *where $\mathcal{U}$ denotes the client, and $\mathsf{View}_{\mathcal{U}}^{\Pi}$ refers to the view of $\mathcal{U}$ in the real-world execution of the protocol Π.*

Theorem 1. *Our protocol is a secure realization of the ideal functionality $\mathcal{F}_{\mathsf{SecODT}}$ according to Definition 2.*

Proof. As per the security definition, we show the existence of a simulator for different corrupted parties (the provider, the client, and either of the cloud servers).

- *Simulator for the corrupted provider*: In the protocol Π, the provider only needs to supply the secret shares of the decision tree and receives no messages. So, the simulator for the corrupted provider can thus be constructed in a dummy way by just outputting the input of the provider. The output of $\mathsf{Sim}_{\mathcal{P}}(\mathcal{T})$ is identically distributed to the view $\mathsf{View}_{\mathcal{P}}^{\Pi}$ of the corrupted provider.
- *Simulator for a corrupted cloud server*: As two cloud servers have a symmetric role in our protocol Π, it suffices to show a simulator $\mathsf{Sim}_{\mathcal{C}_0}$ for $\mathcal{C}_0$. It is noted that the input/output of $\mathcal{C}_0$ in our protocol are just secret shares of some data. The security of additive secret sharing ensures that these secret shares are purely random and can be perfectly simulated by $\mathsf{Sim}_{\mathcal{C}_0}$ using random values. For the interactions between the two cloud servers in different phases, they are in fact due to the calls of the oblivious array-entry read procedure (only used in the secure feature selection phase) and the secret-shared multiplication procedure based on Beaver's triples. Let $\mathsf{Sim}_{\mathcal{C}_0}^{\mathsf{ORead}}$ and $\mathsf{Sim}_{\mathcal{C}_0}^{\mathsf{SecMul}}$ denote the corresponding simulators which can simulate a view indistinguishable from real view for $\mathcal{C}_0$ in the oblivious array-entry read procedure and the secret-shared multiplication procedure respectively. It is noted that the existence of these two simulators has been proved in prior work. With the existence of these simulators, $\mathsf{Sim}_{\mathcal{C}_0}$ first runs $\mathsf{Sim}_{\mathcal{C}_0}^{\mathsf{ORead}}$ with random strings as input in the secure feature selection phase. Then, $\mathsf{Sim}_{\mathcal{C}_0}$ sets the simulated output as the input to the subsequent phases. On each call of the secret-shared multiplication procedure, $\mathsf{Sim}_{\mathcal{C}_0}$ runs $\mathsf{Sim}_{\mathcal{C}_0}^{\mathsf{SecMul}}$ in order. Finally, $\mathsf{Sim}_{\mathcal{C}_0}$ combines and outputs in order the simulated view by $\mathsf{Sim}_{\mathcal{C}_0}^{\mathsf{ORead}}$ and $\mathsf{Sim}^{\mathsf{BMul}}$ on every secure multiplication as its output. This generates the final simulator $\mathsf{Sim}_{\mathcal{C}_0}$ for the cloud server $\mathcal{C}_0$.
- *Simulator for the corrupted client*: In the protocol Π, the client supplies secret shares of the feature vector $\mathbf{x}$ and only receives the two shares $[u^*]_0$ and $[u^*]_1$ of the inference result, from which the plaintext inference result u^* is reconstructed as the output of the client. The simulator thus only needs to simulate the messages (two shares) received by the client given his output u^*. It can set a random value r as one of the shares, say $[u^*]_1$, and $u^* - r$ for the other share $[u^*]_0$. This is in fact just a direct application of additive secret sharing, the security of which ensures that $[u^*]_0$ and $[u^*]_1$ random values and indistinguishable from the shares received by the client. The combination of the two simulated shares also produces u^*, which is the same as the output in the real protocol execution and thus guarantees cor-

Fig. 8. Communication performance of the provider.

Fig. 9. Computation performance of the provider.

rectness. So the output of $\mathsf{Sim}_{\mathcal{U}}(\mathbf{x}, \mathcal{T}(\mathbf{x}))$ is identically distributed to the view $\mathsf{View}_{\mathcal{U}}^{\Pi}$ of the corrupted client. The proof of Theorem 1 is completed.

$\square$

6 EXPERIMENTS

6.1 Setup

Our protocol is implemented in C++. For the oblivious transfer primitive, we rely on the libOTe library [27] which provides implementation of the protocol in [28]. Cloud-side experiments are conducted over two AWS t3.xlarge instances equipped with Intel Xeon Platinum 8175M CPU (2.50GHz and 16GB RAM): one in Europe (London) and one in US East (N. Virginia). The average latency is 75.422 ms and bandwidth is 161 Mbits/s. These two instances are situated in different regions for simulating the real-world scenario that the cloud servers are in different trust domains. The provider and the client are evaluated on an AWS t2.xlarge instance possessing an Intel Xeon E5-2676 v3 processor (2.40GHz and 16GB RAM). We test with synthetic decision trees with realistic configurations, following prior works [12], [15], [16]. The tree depth d varies from 3 to 17, and the dimension I of the feature vector varies from 9 to 57. We make comparison with the state-of-the-art prior work by Zheng et al. [12] (the *ZDWWN* protocol).

6.2 Local-side Performance Evaluation

We first evaluate the performance on the local side, i.e., the provider and the client. Fig. 8 and Fig. 9 show the communication and computation costs of the provider for varying decisions trees, along with comparison with the ZDWWN protocol. As the provider in our design just constructs an index vector of $O(J)$ size rather than a matrix of size $O(J \cdot I)$ as in the ZDWWN protocol, he can enjoy significant cost savings. For different decision trees being tested, the communication cost of the provider ranges from 0.0003 MB to 6 MB in our system, while it is from 0.0016 MB to 118 MB in the ZDWWN protocol. In terms of the provider's running time, it varies from 0.01 ms to 33.842 ms in our system,

TABLE 2
Performace of the Client

d	I	Computation (ms)	Communication (KB)
3	13	0.0057	0.22
4	15	0.0060	0.25
8	9	0.0054	0.16
13	13	0.0056	0.22
17	57	0.0098	0.91

while it is from 0.013 ms to 619.723 ms in the ZDWWN protocol. Overall, our system can offer the provider up to $19\times$ savings (average of $7\times$) in communication. In computation, our system can offer the provider up to $18\times$ savings (average of $5\times$). In Table 2, we give the communication cost and computation cost of the client respectively. It is noted that the client has minimal costs which only scales with the dimension of the feature vector. Recall that the client in our system has the same cost as in the ZDWWN protocol, given that the polynomial-based mechanism is used in the phase of secure inference generation which allows the client to only receive the two shares of the inference result.

6.3 Cloud-side Performance Evaluation

We now examine the performance on the cloud side. Firstly, we show in Fig. 10 the amount of data transferred between the cloud servers in our system and make comparison with the ZDWWN protocol. Our design has relatively higher communication cost (average of $1.9\times$) than the ZDWWN protocol, which is mainly due to the sue of OT in the secure feature selection phase and the secret-shared multiplications in the secure inference generation phase. We emphasize that such overhead in these two phases is the trade-off for the substantial efficiency improvement on the provider side, which is the first priority in our design philosophy.

Significantly reduced overall online cloud service latency. On another hand, it is noted that the overall end-to-end online inference latency in our system is still much less than the ZDWWN protocol, as shown in Fig. 11. This means that

Fig. 10. Communication performance at the cloud.

Fig. 11. Overall *end-to-end* online runtime performance at the cloud over realistic WAN.

compared with the ZDWWN protocol, our system provides much better service experience for the client and fits much better into the practical realm due to the capability in giving faster response. In particular, our new design is up to $8\times$ (average of $5\times$) faster than the ZDWWN protocol. Such efficiency is attributed to the significant optimization on the secure decision node evaluation phase, where the round complexity is largely reduced from linear (as in the ZD-WWN protocol) to logarithmic. A breakdown of the overall cloud-side online inference latency is given in Table 3.

7 RELATED WORK

There has been some work on secure decision tree inference [15], [16], [17], [18], [29], [30]. Most of prior works [15], [16], [17], [18], [29], [30] focus on the non-outsourcing setting where a customized protocol is designed for running between the provider and the client. For example, in [16], to achieve secure feature selection, the client sends to the provider the ciphertext of the feature vector under homomorphic encryption, and then the provider directly selects the ciphertext of each feature for each decision node based on his plaintext selection mapping. Whether these protocols can be effectively adapted to the outsourcing setting remains largely unclear, since the outsourced service requires operations to conducted over encrypted decision tree and feature vector from the very beginning and also raises more design considerations for security and functionality. Moreover, many protocols make use of heavy cryptographic tools (e.g., fully/partially homomorphic encryption, garbled circuits, and ORAM) in the latency-sensitive online interactions. Although the protocol in [18] uses secret sharing, it is yet designed to fully and inefficiently work on binary representations of the decision tree as well as the feature vector provided from the very beginning, with all the secure processing conducted at bitwise level. So their protocol is also not directly adaptable for efficient secure outsourcing.

Very recently, the work [12] presents the first design tailored for secure outsourcing of decision tree inference, which runs under the two-server model and only makes use of additive secret sharing to securely realize the various components for the online execution of the service. As an initial attempt, however, its performance is yet to be optimized. Our new highly efficient design presents significant optimizations which largely improves the overall online end-to-end latency of the secure inference service provided by the cloud, as well as the provider's performance.

Our work is also related to the line of work (e.g., [22], [29], [31], [32], [33], to just list a few) on securely evaluating other machine learning models, such as hyperplane decision [29], Naïve Bayes [29], neural networks [22], [31], [32], [33]. The common blueprint therein is to build specializaedprotocols tailored for the specific computation required by different models through different cryptographic techniques. For example, the work of Liu et al. [31] supports secure neural network evaluation using secret sharing and garbled circuits; the work of Juvekar et al. [32] relies on highly customized use of homomorphic encryption and garbled circuits to support low latency in secure neural network evaluation. Most of these works operate under the non-outsourcing and aim to protect privacy for the model and client input. There are some works [22], [34] also operating under the two-server model as in this work, with the tailored support for secure evaluation of models such as linear regression, logistic regression, and neural networks. Some recent efforts have also been presented on secure machine learning under the three-server [35], [36]/four-server [37] model (for models other than decision trees), where three/four servers have to engage in the online interactions.

8 CONCLUSION

In this paper, we design, implement, and evaluate a new system that allows highly efficient secure outsourcing of decision tree inference. Through the synergy of several delicate optimizations which securely shift most workload of the provider to the cloud and reduce the communication round complexities between the cloud servers, our system significantly improves upon the state-of-the-art prior work. Extensive experiments demonstrated that compared with the state-of-the-art, our new system achieves up to $8\times$ better online end-to-end inference latency between the cloud servers over realistic WAN, as well as allows the provider to

TABLE 3
Breakdown of Runtimes in Different Phases (in seconds) at the Cloud Servers, over WAN

Parameters		Secure Feature Selection		Secure Decision Node Evaluation		Secure Inference Generation	
d	I	ZDWWN	Ours	ZDWWN	Ours	ZDWWN	Ours
3	13	0.151	**0.527**	9.38	**0.529**	0.154	**0.154**
4	15	0.156	**0.529**	9.454	**0.53**	0.306	**0.306**
8	9	0.167	**0.533**	9.456	**0.532**	0.383	**0.383**
13	13	0.393	**0.91**	10.03	**1.735**	0.69	**0.69**
17	57	19.376	**21.006**	11.669	**4.281**	1.785	**1.785**

enjoy $19\times$ savings in communication cost and $18\times$ savings in computation cost.

REFERENCES

[1] A. T. Azar and S. M. El-Metwally, "Decision tree classifiers for automated medical diagnosis," *Neural Computing and Applications*, vol. 23, no. 7-8, pp. 2387–2403, 2013.

[2] M. W. Libbrecht and W. S. Noble, "Machine learning applications in genetics and genomics," *Nature Reviews Genetics*, vol. 16, no. 6, pp. 321–332, 2015.

[3] F. Wang, H. Zhu, R. Lu, Y. Zheng, and H. Li, "Achieve efficient and privacy-preserving disease risk assessment over multi-outsourced vertical datasets," *IEEE Trans. Dependable Secure Comput.*, pp. 1–1, 2020.

[4] B. W. Yap, S. Ong, and N. H. M. Husain, "Using data mining to improve assessment of credit worthiness via credit scoring models," *Expert Syst. Appl.*, vol. 38, no. 10, pp. 13 274–13 283, 2011.

[5] D. Delen, C. Kuzey, and A. Uyar, "Measuring firm performance using financial ratios: A decision tree approach," *Expert Syst. Appl.*, vol. 40, no. 10, pp. 3970–3983, 2013.

[6] S. S. S. Sindhu, S. Geetha, and A. Kannan, "Decision tree based light weight intrusion detection using a wrapper approach," *Expert Syst. Appl.*, vol. 39, no. 1, pp. 129–141, 2012.

[7] N. B. Amor, S. Benferhat, and Z. Elouedi, "Naive bayes vs decision trees in intrusion detection systems," in *Proc. of ACM SAC*, H. Haddad, A. Omicini, R. L. Wainwright, and L. M. Liebrock, Eds.

[8] Microsoft Azure, "Deploy models with Azure Machine Learning," https://docs.microsoft.com/en-us/azure/machine-learning/service/how-to-deploy-and-where, 2019, [Online].

[9] Google Cloud, "Deploying your model," https://cloud.google.com/vision/automl/object-detection/docs/deploy, 2019, [Online].

[10] AWS, "Deploy a Model on Amazon SageMaker Hosting Services ," https://docs.aws.amazon.com/sagemaker/latest/dg/how-it-works-hosting.html, 2019, [Online].

[11] J. Liang, Z. Qin, S. Xiao, L. Ou, and X. Lin, "Efficient and secure decision tree classification for cloud-assisted online diagnosis services," *IEEE Trans. Dependable and Secure Computing*, 2019, doi: 10.1109/TDSC.2019.2922958.

[12] Y. Zheng, H. Duan, C. Wang, R. Wang, and S. Nepal, "Securely and efficiently outsourcing decision tree inference," *IEEE Trans. Dependable Secure Comput.*, pp. 1–1, 2020.

[13] W. Chen and R. A. Popa, "Metal: A metadata-hiding file sharing system," in *Proc. of NDSS*, 2020.

[14] D. Harris, "A taxonomy of parallel prefix networks," in *Proc. of Asilomar Conference on Signals, Systems & Computers*, 2003.

[15] D. J. Wu, T. Feng, M. Naehrig, and K. E. Lauter, "Privately evaluating decision trees and random forests," *PoPETs*, vol. 2016, no. 4, pp. 335–355, 2016.

[16] R. K. H. Tai, J. P. K. Ma, Y. Zhao, and S. S. M. Chow, "Privacy-preserving decision trees evaluation via linear functions," in *Proc. of ESORICS*, 2017.

[17] A. Tueno, F. Kerschbaum, and S. Katzenbeisser, "Private evaluation of decision trees using sublinear cost," *PoPETs*, vol. 2019, no. 1, pp. 266–286, 2019.

[18] M. D. Cock, R. Dowsley, C. Horst, R. Katti, A. Nascimento, W. Poon, and S. Truex, "Efficient and private scoring of decision trees, support vector machines and logistic regression models based on pre-computation," *IEEE Trans. Dependable Secure Comput.*, DOI: 10.1109/TDSC.2017.2679189, 2017.

[19] D. Beaver, "Efficient multiparty protocols using circuit randomization," in *Proc. of CRYPTO*, 1991.

[20] H. Corrigan-Gibbs and D. Boneh, "Prio: Private, robust, and scalable computation of aggregate statistics," in *Poc. of USENIX NSDI*, 2017, pp. 259–282.

[21] Q. Wang, J. Wang, S. Hu, Q. Zou, and K. Ren, "Sechog: Privacy-preserving outsourcing computation of histogram of oriented gradients in the cloud," in *Proc. of ACM AsiaCCS*, 2016.

[22] P. Mohassel and Y. Zhang, "Secureml: A system for scalable privacy-preserving machine learning," in *Proc. of IEEE S&P*, 2017.

[23] N. Agrawal, A. S. Shamsabadi, M. J. Kusner, and A. Gascón, "QUOTIENT: two-party secure neural network training and prediction," in *Proc. of ACM CCS*, 2019.

[24] Y. Zheng, H. Duan, and C. Wang, "Learning the truth privately and confidently: Encrypted confidence-aware truth discovery in mobile crowdsensing," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 10, pp. 2475–2489, 2018.

[25] M. S. Riazi, C. Weinert, O. Tkachenko, E. M. Songhori, T. Schneider, and F. Koushanfar, "Chameleon: A hybrid secure computation framework for machine learning applications," in *Proc. of AsiaCCS*, 2018.

[26] M. Curran, X. Liang, H. Gupta, O. Pandey, and S. R. Das, "Procsa: Protecting privacy in crowdsourced spectrum allocation," in *Proc. of ESORICS*, 2019.

[27] P. Rindal, "libOTe: an efficient, portable, and easy to use Oblivious Transfer Library," https://github.com/osu-crypto/libOTe.

[28] V. Kolesnikov, R. Kumaresan, M. Rosulek, and N. Trieu, "Efficient batched oblivious PRF with applications to private set intersection," in *Proc. of ACM CCS*, 2016.

[29] R. Bost, R. A. Popa, S. Tu, and S. Goldwasser, "Machine learning classification over encrypted data," in *Proc. of NDSS*, 2015.

[30] A. Tueno, Y. Boev, and F. Kerschbaum, "Non-interactive private decision tree evaluation," in *Proc. of DBSec*, A. Singhal and J. Vaidya, Eds., 2020.

[31] J. Liu, M. Juuti, Y. Lu, and N. Asokan, "Oblivious neural network predictions via minionn transformations," in *Proc. of ACM CCS*, 2017.

[32] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, "GAZELLE: A low latency framework for secure neural network inference," in *Proc. of USENIX Security Symposium*, 2018.

[33] P. Mishra, R. Lehmkuhl, A. Srinivasan, W. Zheng, and R. A. Popa, "Delphi: A cryptographic inference service for neural networks," in *Proc. of USENIX Security Symposium*, 2020.

[34] V. Nikolaenko, U. Weinsberg, S. Ioannidis, M. Joye, D. Boneh, and N. Taft, "Privacy-preserving ridge regression on hundreds of millions of records," in *Proc. of IEEE SP*, 2013.

[35] P. Mohassel and P. Rindal, "Aby^3: A mixed protocol framework for machine learning," in *Proc. of ACM CCS*, 2018.

[36] S. Wagh, D. Gupta, and N. Chandran, "Securenn: 3-party secure computation for neural network training," *PoPETs*, vol. 2019, no. 3, pp. 26–49, 2019.

[37] H. Chaudhari, R. Rachuri, and A. Suresh, "Trident: Efficient 4pc framework for privacy preserving machine learning," in *Proc. of NDSS*, 2020.