BlitzBough: An Efficient Privacy-Preserving Inference Framework for Decision Trees by Communication Optimization

Guopeng Lin[1], Yixin Tu[2], Jingwei Pu[1], Zheng Qu[1], Shuyu Chen[1], and Weili Han[1⋆]

[1] Laboratory for Data Security and Governance, Fudan University
Email: 17302010022@fudan.edu.cn, 25213050311@m.fudan.edu.cn, 24110240070@m.fudan.edu.cn, 23110240005@m.fudan.edu.cn, wlhan@fudan.edu.cn

[2] Beijing Normal-Hong Kong Baptist University
Email: tuyixin1220@163.com

Abstract. Inference frameworks for decision trees based on multi-party computation have been a hot spot recently. For these frameworks, it is a key step to securely select the decision tree nodes with secret-shared indexes. However, most existing frameworks are inefficient due to significant communication size, which is proportional to the number of decision tree nodes, for securely selecting the decision tree nodes. In this paper, we present an efficient privacy-preserving inference framework, namely BlitzBough, for decision trees by communication optimization. BlitzBough optimizes the communication size with an efficient element selection protocol that is based on the share dot product and function secret sharing to securely select decision tree nodes with a communication size only proportional to the square root of the number of decision tree nodes. Experimental evaluations with various decision tree settings demonstrate that BlitzBough outperforms the state-of-the-art frameworks by up to 35.1× in terms of communication size and up to 8.2× in terms of running time.

Keywords: Privacy Preservation · Decision Tree · Multi-Party Computation · Communication Optimization

1 Introduction

Decision trees and their ensemble models, e.g., XGBoost, are among the most popular machine learning models [16]. Due to their interpretability, these models are widely deployed in critical areas, such as medicine [2] and finance [18,1], where it is crucial to understand the influencing factors on model outcomes.

Inference frameworks for decision trees based on secure multi-party computation (MPC) have recently attracted significant research attention. In the applications of medicine and finance, e.g., online health assessment [2] and credit-risk assessment [17], users are usually required to provide their sensitive data,

⋆ Corresponding author

while institutions leverage their well-trained decision trees to provide an inference result (e.g., investment advice) based on the user data. With the inference frameworks based on MPC, both the privacy of user data and the parameter information of decision trees can be protected during the inference process.

However, most existing inference frameworks [19,21,3,25] for decision trees based on MPC are still inefficient due to their significant communication size. The key step of the privacy-preserving inference with a decision tree is to securely select the decision tree nodes with secret-shared indexes. Nevertheless, most existing frameworks suffer from significant communication size, which is proportional to the number of decision tree nodes, for securely selecting the decision tree nodes. For instance, the framework proposed by Ma et al.[21] uses 1-out-of-n oblivious transfer (OT for short) to securely select decision tree nodes. This method requires the decision tree holder to send an encrypted decision tree to the users, which results in a communication size proportional to the number of decision tree nodes. The framework proposed by Bai et al. [3] reduces the online communication size to proportional to the decision tree depth by a shared oblivious selection protocol. Nevertheless, the communication size in the offline phase is still proportional to the number of decision tree nodes for each inference query from new users. A communication size proportional to the number of nodes will be a serious efficiency bottleneck in bandwidth-limited scenarios.

In this paper, we present an efficient privacy-preserving inference framework, namely `BlitzBough`, for decision trees by communication optimization. `BlitzBough` resolves the above communication issue with an efficient element selection protocol based on the share dot product [22] and function secret sharing [7]. The key observation is that the communication size of a share dot product depends only on its output size, not its input size. Leveraging this property, our protocol proceeds in three steps. First, it structures the n decision tree nodes into a $\lceil\sqrt{n}\rceil \times \lceil\sqrt{n}\rceil$ matrix. Second, it converts the secret-shared index into two secret-shared one-hot vectors of size $\lceil\sqrt{n}\rceil$ each (a row and a column indicator) using function secret sharing. Third, it retrieves the target node via two share dot products: the matrix multiplied by the column one-hot vector yields a secret-shared vector of size $\lceil\sqrt{n}\rceil$, which is then dotted with the row one-hot vector to yield the target element. The total communication is therefore only proportional to $\lceil\sqrt{n}\rceil$, i.e., the square root of the number of decision tree nodes.

The extensive experimental results under various decision tree settings demonstrate that `BlitzBough` outperforms the state-of-the-art frameworks [3,21] by up to 35.1× in terms of communication size and up to 8.2× in terms of running time. These results demonstrate that `BlitzBough` can provide privacy-preserving inference service efficiently, especially in bandwidth-limited scenarios.

2 Preliminaries

2.1 Decision Tree Inference

A decision tree is a binary tree consisting of internal nodes and leaf nodes. An internal node $\mathcal{N}$ holds an attribute id $\mathcal{N}.aid$, a threshold $\mathcal{N}.t$, and pointers to

its left and right children $\mathcal{N}.lc$ and $\mathcal{N}.rc$; a leaf node holds only a label $\mathcal{N}.y$. Given an attribute vector $\vec{x}$ (where $\vec{x}[i]$ denotes the i-th attribute), inference proceeds top-down from the root: at each internal node, $\vec{x}[\mathcal{N}.aid]$ is compared with $\mathcal{N}.t$ to choose the left or right child, and the algorithm outputs $\mathcal{N}.y$ upon reaching a leaf.

2.2 Multi-Party Computation

MPC enables multiple parties to jointly compute a function while keeping their inputs private. Mainstream techniques include homomorphic encryption, garbled circuits, and secret sharing. BlitzBough builds on three-party *replicated secret sharing* (RSS) for its simplicity and efficiency, and additionally uses *function secret sharing* (FSS) [7] to convert secret-shared indexes into secret-shared one-hot vectors.

Replicated Secret Sharing. To share a secret x among three parties (P_0, P_1, P_2), three random values $\langle x\rangle_0, \langle x\rangle_1, \langle x\rangle_2 \in \mathbb{Z}_{2^\ell}$ are sampled with $x = (\langle x\rangle_0 + \langle x\rangle_1 + \langle x\rangle_2) \bmod 2^\ell$, and P_i holds the pair $(\langle x\rangle_i, \langle x\rangle_{(i+1) \bmod 3})$. We write $\langle x\rangle$ for such a sharing. Reconstruction is performed by collecting all three shares.

Let a, b, c be public constants, $\langle x\rangle, \langle y\rangle$ secret-shared scalars, $\langle \mathbf{M}\rangle$ a secret-shared $n \times m$ matrix, and $\langle \vec{v}\rangle, \langle \vec{u}\rangle$ secret-shared vectors of size m. We use four standard RSS operations: *linear* $\langle z\rangle = a\langle x\rangle + b\langle y\rangle + c$ (no communication); *share multiplication* $\langle z\rangle = \langle x\rangle\langle y\rangle$ (3ℓ bits); *matrix–vector dot product* $\langle \vec{z}\rangle = \langle \mathbf{M}\rangle \cdot \langle \vec{v}\rangle$ ($3n\ell$ bits); and *vector–vector dot product* $\langle z\rangle = \langle \vec{v}\rangle \cdot \langle \vec{u}\rangle$ (3ℓ bits). *The key observation we exploit is that the communication cost of the share dot product depends only on the output size, not the input size.*

Function Secret Sharing. FSS [7] splits a function f into two keys (key_0, key_1) such that neither key reveals f's parameters, but the two parties can locally evaluate additive shares of $f(x)$. It consists of $\mathbf{Gen}(1^\lambda, f) \to (key_0, key_1)$ and $\mathbf{Eval}(b, key_b, x) \to \beta_x^{(b)}$, with $\beta_x^{(0)} + \beta_x^{(1)} = f(x) \pmod{|\mathbb{G}^{out}|}$. Two standard instantiations are the *Distributed Point Function* (DPF) for $f_{\alpha,\beta}(x) = \beta \cdot \mathbf{1}[x = \alpha]$ and the *Distributed Comparison Function* (DCF) for $f_{<\alpha,\beta}(x) = \beta \cdot \mathbf{1}[x < \alpha]$. From DCF, one obtains efficient less-than protocols [13]: $\langle z\rangle = LT(\langle x\rangle, \langle y\rangle)$ and $\langle z\rangle = LT(\langle x\rangle, c)$, where $z = 1$ iff the inequality holds.

3 Overview of BlitzBough

3.1 Architecture and Security Model

The architecture of BlitzBough contains three parties and an arbitrary number of users. Each of the three parties holds an RSS share of a decision tree. Each user holds their sensitive data, such as medical records, and would like to get an inference result for their data while keeping their data private.

The privacy-preserving inference process is performed as follows: (1) Each user shares their data with the three parties using RSS. (2) The three parties perform inference using the inference protocol introduced in Section 4.1 to obtain

the shares of the inference result. (3) The three parties send the shares of the inference result to the user, such that the user can reconstruct the inference result from the shares.

`BlitzBough` adopts a semi-honest security model with an honest majority. Concretely, we assume that the three parties will correctly execute the inference protocol without colluding with each other, and the users will correctly provide the shares of their data.

Besides, consistent with the works [3,21], some parameters in `BlitzBough` are public, including the number of nodes in each layer, the depth of the decision tree, and the number of user attributes.

3.2 Data Representation

Representation of a decision tree: The representation of a decision tree is as shown in Figure 1. Each branch is initially padded to reach the depth d of the decision tree to keep the inference path private. The decision tree is then represented by vectors layer by layer.

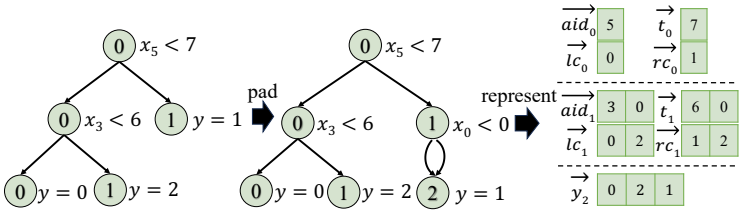

Fig. 1. An example to show the padding process and representation method.

The padding process is performed from the layer of depth 0 to the layer of depth d. If there is a leaf node with a label y in the layer of depth k (where $k < d$), it will be converted into an internal node with a default attribute id (for simplicity, we set the default attribute id to 0), and a default threshold (for simplicity, we set the default threshold to 0). A leaf node with label y is then added in the $k+1$ layer. The left and right children of the internal node are set to this new leaf node.

After padding, an internal layer of depth k is represented using four vectors: $\overrightarrow{aid}_k$, $\vec{t}_k$, $\vec{lc}_k$, and $\vec{rc}_k$. Specifically, $\overrightarrow{aid}_k[i]$ and $\vec{t}_k[i]$ denote the attribute id and threshold for the i-th node in the internal layer. Meanwhile, $\vec{lc}_k[i]$ and $\vec{rc}_k[i]$ denote the indices of the left and right child nodes of the i-th node. The leaf layer at depth d is represented by a label vector $\vec{y}_d$, where each element $\vec{y}_d[i]$ denotes the label of the i-th node.

Representation of user data: We use a vector $\vec{x}$ of size m to represent the user data with m attributes. Each $\vec{x}[i]$ represents the i-th attribute value of the user data.

4 Design of BlitzBough

4.1 Decision Tree Inference Protocol

As is shown in Protocol 1, the protocol *DecisionTreeInference* inputs a secret-shared user data vector $\langle \vec{x} \rangle$, d secret-shared attribute id vectors, $\langle \overrightarrow{aid}_0 \rangle, \ldots, \langle \overrightarrow{aid}_{d-1} \rangle$, d secret-shared threshold vectors, $\langle \vec{t}_0 \rangle, \ldots, \langle \vec{t}_{d-1} \rangle$, d secret-shared left child node index vectors $\langle \vec{lc}_0 \rangle, \ldots, \langle \vec{lc}_{d-1} \rangle$, and d secret-shared right child node index vectors $\langle \vec{rc}_0 \rangle, \ldots, \langle \vec{rc}_{d-1} \rangle$, and a secret-shared label vector $\langle \vec{y}_d \rangle$. And it outputs a secret-shared inference result $\langle res \rangle$. Starting from the root ($\langle idx \rangle = \langle 0 \rangle$, Line 1), the parties iterate over the d internal layers (Line 2-7): in each layer, *ElementSelection*$^+$ (introduced in Section 4.2) retrieves the attribute id, threshold, and child indices of the current node (Line 3); *ElementSelection* fetches the corresponding attribute value (Line 4); the parties then compute $\langle b \rangle = LT(\langle t \rangle, \langle value \rangle)$ (Line 5), and update $\langle idx \rangle$ to the left child $\langle lc \rangle$ when $value \leq t$ (i.e., $b = 0$) or to the right child $\langle rc \rangle$ when $value > t$ (i.e., $b = 1$) (Line 6). For example, at a node with $t = 7$, $lc = 1$, and $rc = 2$: if $value = 4$, then $b = LT(7, 4) = 0$ and $\langle idx \rangle$ is set to $lc = 1$; if $value = 9$, then $b = LT(7, 9) = 1$ and $\langle idx \rangle$ is set to $rc = 2$. After the loop, *ElementSelection* retrieves the leaf label as the inference result (Line 8).

Protocol 1: *DecisionTreeInference*

Input: A secret-shared user data vector $\langle \vec{x} \rangle$, d secret-shared attribute id vectors, $\langle \overrightarrow{aid}_0 \rangle, \cdots, \langle \overrightarrow{aid}_{d-1} \rangle$, d secret-shared threshold vectors, $\langle \vec{t}_0 \rangle, \cdots, \langle \vec{t}_{d-1} \rangle$, d secret-shared left child node index vectors $\langle \vec{lc}_0 \rangle, \cdots, \langle \vec{lc}_{d-1} \rangle$, d secret-shared right child node index vectors $\langle \vec{rc}_0 \rangle, \cdots, \langle \vec{rc}_{d-1} \rangle$, a secret-shared label vector $\langle \vec{y}_d \rangle$
Output: A secret-shared inference result $\langle res \rangle$

1: $\langle idx \rangle = \langle 0 \rangle$.
2: **for** $k = 0$ *to* $d - 1$ **do**
3: $\quad \langle aid \rangle, \langle t \rangle, \langle lc \rangle, \langle rc \rangle = ElementSelection^+(\langle idx \rangle, \langle \overrightarrow{aid}_k \rangle, \langle \vec{t}_k \rangle, \langle \vec{lc}_k \rangle, \langle \vec{rc}_k \rangle)$.
4: $\quad \langle value \rangle = ElementSelection(\langle aid \rangle, \langle \vec{x} \rangle)$.
5: $\quad \langle b \rangle = LT(\langle t \rangle, \langle value \rangle)$.
6: $\quad \langle idx \rangle = (1 - \langle b \rangle) * \langle lc \rangle + \langle b \rangle * \langle rc \rangle$.
7: **end for**
8: $\langle res \rangle = ElementSelection(\langle idx \rangle, \langle \vec{y}_d \rangle)$.

4.2 Element Selection Protocol

Main idea: The communication optimization of our proposed element selection protocol comes from the fact that the communication size of the share dot

product (introduced in Section 2.2) only depends on the size of its output vector. Specifically, if parties perform share dot product between a matrix of dimensions $d_r * d_c$ and a vector of size d_c to get a vector of size d_r, the communication size is only $O(d_r * \ell)$ bits, where ℓ is the bit length of the ring $\mathbb{Z}_{2^\ell}$. Thus, as shown in Figure 2, to select an element from a vector of size n with a secret-shared index, parties structure the vector into a matrix of dimensions $\lceil\sqrt{n}\rceil * \lceil\sqrt{n}\rceil$, where the additional space are padding with '0'. Then, the parties convert the secret-shared index into two secret-shared one-hot vectors (a row one-hot vector and a column one-hot vector) of size $\lceil\sqrt{n}\rceil$. The row one-hot vector contains '1' at the index corresponding to the row index of the desired element, while the column one contains '1' at the index corresponding to the column index of the desired element. The desired element can be retrieved by performing a secret-shared dot product with the matrix and the two one-hot vectors. Using this method, the communication size for selecting an element with a secret-shared index is only $O(\lceil\sqrt{n}\rceil * \ell)$ bits.

Protocol 2: *ElementSelection*

Input: A secret-shared index $\langle idx\rangle$, and a secret-shared vectors $\langle \vec{e}\rangle$
Output: A secret-shared element $\langle u\rangle$ that equals to $\langle \vec{e}[idx]\rangle$

1: $sz = SizeOf(\langle \vec{e}\rangle)$
2: **if** $sz \geq SizeThreshold$ **then**
3: $\quad\langle\overrightarrow{roh}\rangle,\langle\overrightarrow{coh}\rangle = IndexConversion(\langle idx\rangle, sz)$
4: $\quad\langle\mathbb{E}\rangle = Struct(\langle\vec{e}\rangle, SizeOf(\langle\overrightarrow{roh}\rangle), SizeOf(\langle\overrightarrow{coh}\rangle))$.
5: $\quad\langle u\rangle = \langle\mathbb{E}\rangle \cdot \langle\overrightarrow{coh}\rangle \cdot \langle\overrightarrow{roh}\rangle$.
6: **else**
7: $\quad\langle\overrightarrow{oh}\rangle = ToOneHot(\langle idx\rangle, sz)$
8: $\quad\langle u\rangle = \langle\overrightarrow{oh}\rangle \cdot \langle\vec{e}\rangle$.
9: **end if**

Protocol details: Protocol 2 shows *ElementSelection*, which takes a shared index $\langle idx\rangle$ and a shared vector $\langle\vec{e}\rangle$, and outputs $\langle u\rangle = \langle\vec{e}[idx]\rangle$. When $|\langle\vec{e}\rangle|$ exceeds a threshold *SizeThreshold*, the parties convert $\langle idx\rangle$ into row and column one-hot vectors, reshape $\langle\vec{e}\rangle$ into a matrix $\langle\mathbb{E}\rangle$, and retrieve the element via a share dot product (Line 3-5). Otherwise, converting $\langle idx\rangle$ into two one-hot vectors is less efficient than a single conversion, so the parties directly call *ToOneHot* and perform a one-shot dot product with $\langle\vec{e}\rangle$ (Line 7-8).

***ElementSelection*$^+$.** When multiple elements need to be selected at the same shared index $\langle idx\rangle$ (as in Line 3 of Protocol 1), we extend *ElementSelection* to *ElementSelection*$^+$, which takes c input vectors $\langle\vec{e}_0\rangle, \ldots, \langle\vec{e}_{c-1}\rangle$ and outputs $\langle\vec{e}_0[idx]\rangle, \ldots, \langle\vec{e}_{c-1}[idx]\rangle$. It reuses the one-hot vectors converted from $\langle idx\rangle$ across all c dot products, reducing the conversion cost from c calls to 1.

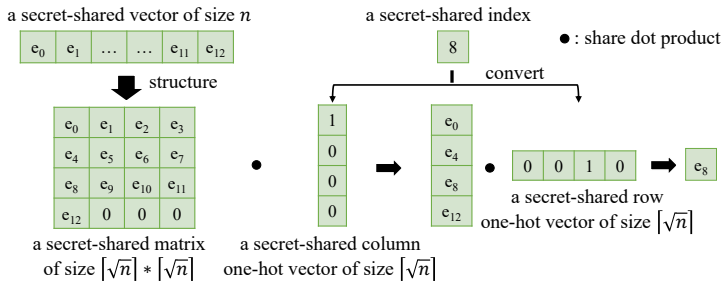

Fig. 2. An example to show the idea of the efficient element selection protocol.

4.3 Index Conversion Protocol

Main idea: To convert an index into two one-hot vectors, the key step is to compute the indices that contain '1' in the two one-hot vectors. This can be achieved by calculating the quotient and remainder between the index and the size of the column one-hot vector with a secure division protocol. However, performing a secure division protocol usually requires a large number of communication rounds. As a solution, we set the size of the column one-hot vector to be a power of 2, such as 2^k. In this way, we can leverage the index's binary representation, dividing it into the most significant $\ell - k$ bits for the quotient and the least significant k bits for the remainder. That is, for an index on $\mathbb{Z}_{2^\ell}$ bits, its most significant $\ell - k$ bits represent the quotient when dividing by 2^k, while its least significant k bits represent the remainder.

To divide the index into the most significant $\ell-k$ bits and the least significant k bits, we utilize Theorem 1, which is a variant of the Trunc Theorem proposed by Zhou et al. [26]. According to Theorem 1, with an almost certain probability (larger than $1 - 2^{l_{idx}+1-\ell}$, where l_{idx} is usually smaller than 15 and ℓ is usually set to 64), the local truncation result $(tmp_0 \gg k)+(-((-tmp_1) \gg k))$ will equal to $\lfloor idx/2^k \rfloor + bit$, where $bit = 1$ if $(tmp_0 \mod 2^k) + (tmp_1 \mod 2^k) < 2^k$ and $(tmp_1 \mod 2^k) > 0$, else $bit = 0$. Hence, the parties first locally truncate the shares of the index to get the most significant $\ell - k$ bits. This process may introduce one-bit error if $(tmp_0 \mod 2^k) + (tmp_1 \mod 2^k) < 2^k$ and $(tmp_1 \mod 2^k) > 0$. Then, the parties judge whether the one-bit error occurs, and then eliminate the one-bit error.

Theorem 1. *Let $idx \in [0, 2^{l_{idx}})$ and $tmp_0, tmp_1 \in [0, 2^\ell)$ satisfy $(tmp_0+tmp_1) \bmod 2^\ell = idx$, and let $k < l_{idx} < \ell-1$. Then with probability greater than $1-2^{l_{idx}+1-\ell}$,*

$$(tmp_0 \gg k) + (-((-tmp_1) \gg k)) = \lfloor idx/2^k \rfloor + bit,$$

where $bit = 1$ if $(tmp_0 \bmod 2^k) + (tmp_1 \bmod 2^k) < 2^k$ and $(tmp_1 \bmod 2^k) > 0$, and $bit = 0$ otherwise.

Proof. Decompose $idx = idx' \cdot 2^k + idx''$ and $tmp_b = tmp'_b \cdot 2^k + tmp''_b$ for $b \in \{0, 1\}$, with $idx'', tmp''_0, tmp''_1 \in [0, 2^k)$. By [26], with probability greater

than $1 - 2^{l_{idx}+1-\ell}$ the truncation result equals $\lfloor idx/2^k \rfloor + bit$, where $bit = 1$ iff $tmp_0'' < idx''$. It thus suffices to show

$$tmp_0'' < idx'' \iff tmp_0'' + tmp_1'' < 2^k \text{ and } tmp_1'' > 0.$$

From $(tmp_0 + tmp_1) \bmod 2^\ell = idx$ we have $(tmp_0'' + tmp_1'') \bmod 2^k = idx''$. The modular sum therefore falls into exactly one of two cases:

- *No-wrap case* ($tmp_0'' + tmp_1'' < 2^k$): here $idx'' = tmp_0'' + tmp_1''$, so $tmp_0'' < idx''$ holds iff $tmp_1'' > 0$.
- *Wrap case* ($tmp_0'' + tmp_1'' \geq 2^k$): here $idx'' = tmp_0'' + tmp_1'' - 2^k$. Since $tmp_1'' < 2^k$, we have $idx'' < tmp_0''$, so $tmp_0'' < idx''$ does not hold.

Combining the two cases, $tmp_0'' < idx''$ holds iff the no-wrap case occurs *and* $tmp_1'' > 0$, which is exactly the right-hand condition.

Protocol 3: *IndexConversion*

Input: A secret-shared index $\langle idx \rangle$, and an integer sz
Output: Two secret-shared one-hot vectors $\langle \overrightarrow{roh} \rangle$,$\langle \overrightarrow{coh} \rangle$

1: $k = \lceil \log sz \rceil / 2$.
2: P_0 locally computes $tmp_0 = (\langle idx \rangle_0 + \langle idx \rangle_1)$ *mod* 2^ℓ. P_1 locally computes $tmp_1 = \langle idx \rangle_2$.
3: P_0 locally computes $hp_0 = tmp_0 \gg k$ and $lp_0 = tmp_0$ *mod* 2^k. P_1 locally computes $hp_1 = -((-tmp_1) \gg k)$, $lp_1 = tmp_1$ *mod* 2^k, and $b = lp_1 > 0$.
4: P_0 shares lp_0 and hp_0 using RSS. P_1 shares lp_1, hp_1, and b using RSS.
5: $\langle hp \rangle = \langle hp_0 \rangle + \langle hp_1 \rangle$.
6: $\langle lp \rangle = \langle lp_0 \rangle + \langle lp_1 \rangle$.
7: $\langle bit \rangle = LT(\langle lp \rangle, 2^k) * \langle b \rangle$.
8: $\langle rowidx \rangle = \langle hp \rangle - \langle bit \rangle$.
9: $\langle colidx \rangle = \langle idx \rangle - \langle rowidx \rangle * 2^k$.
10: $\langle \overrightarrow{coh} \rangle = ToOneHot(\langle colidx \rangle, 2^k)$.
11: $\langle \overrightarrow{roh} \rangle = ToOneHot(\langle rowidx \rangle, \lceil sz/2^k \rceil)$.

Protocol details: Protocol 3 shows *IndexConversion*, which takes a shared index $\langle idx \rangle$ and an integer *size*, and outputs the shared row and column one-hot vectors. It first sets k as the bit length of the column vector size (Line 1). Then P_0 and P_1 locally perform truncation and modulo on their shares, with P_1 additionally checking whether $tmp_1 \bmod 2^k > 0$, and share the results via RSS (Line 2-6). Next, the parties securely detect and eliminate the potential one-bit error to obtain the shared row and column indices (Line 7-9), which are finally converted into one-hot vectors via *ToOneHot* (Line 10-11).

Protocol 4 shows *ToOneHot*, which converts a shared index $\langle idx \rangle$ into a shared one-hot vector $\langle \overrightarrow{oh} \rangle$ of size *size* using DPF. In the offline phase, P_2 generates DPF keys (key_0, key_1) for $f_{r,1}$ and random shares (r_0, r_1) satisfying

$(r_0+r_1) \bmod 2^\ell = r$, then distributes (key_0, r_0) to P_0 and (key_1, r_1) to P_1. In the online phase, P_0 and P_1 mask their shares with r_0 and r_1, exchange the masked values, and reconstruct $tmp = (idx + r) \bmod 2^\ell$ (Line 1-3). Each party then locally evaluates its DPF key on $(tmp - i) \bmod 2^\ell$ for $i = 0, \ldots, sz-1$ to obtain arrays $\overrightarrow{arr}_0$ and $\overrightarrow{arr}_1$ (Line 4). By the definition of $f_{r,1}$, $\overrightarrow{arr}_0[i] + \overrightarrow{arr}_1[i] = 1$ iff $i = idx$. The parties share the arrays via RSS and sum them to obtain $\langle \overrightarrow{oh} \rangle$ (Line 5-6).

Protocol 4: *ToOneHot*

Input: A secret-shared index $\langle idx \rangle$, an integer $size$
Output: a secret-shared one hot vector $\langle \overrightarrow{oh} \rangle$ of size $size$
offline: P_2 generates the DPF keys key_0 and key_1 of $f_{r,1}$ and two random number r_0 and r_1, such that $(r_0 + r_1)\ mod\ 2^\ell = r$. After that, P_2 sends key_0 and r_0 to P_0, sends key_1 and r_1 to P_1.

1: P_0 locally computes $tmp_0 = (\langle idx \rangle_0 + \langle idx \rangle_1 + r_0)\ mod\ 2^\ell$, P_1 locally computes $tmp_1 = (\langle idx \rangle_2 + r_1)\ mod\ 2^\ell$.
2: P_0 sends tmp_0 to P_1, P_1 sends tmp_1 to P_0.
3: P_0 and P_1 locally computes $tmp = (tmp_0 + tmp_1)\ mod\ 2^\ell$.
4: P_0 locally computes $\overrightarrow{arr}_0[i] = Eval(0, key_0, (tmp - i)\ mod\ 2^\ell)$, for $i = 0$ to $sz - 1$. P_1 locally computes $\overrightarrow{arr}_1[i] = Eval(1, key_1, (tmp - i)\ mod\ 2^\ell)$, for $i = 0$ to $sz - 1$.
5: P_0 and P_1 share $\overrightarrow{arr}_0$ and $\overrightarrow{arr}_1$ using RSS.
6: $\langle \overrightarrow{oh} \rangle = \langle \overrightarrow{arr}_0 \rangle + \langle \overrightarrow{arr}_1 \rangle$.

4.4 Communication Complexity Analysis

We compare the communication complexity of BlitzBough with three state-of-the-art frameworks: (1) The homomorphic encryption (HE for short) based framework proposed by Bai et al.[3]. (2) The pseudorandom function (PRF for short) based framework proposed by Bai et al.[3]. (3) The OT-based framework proposed by Ma et al.[21].

As is shown in Table 1, in terms of total communication size and offline communication size, BlitzBough is the only framework that requires a communication size proportional to the square root of the number of decision tree nodes. While the other frameworks all require a communication size proportional to the number of decision tree nodes. In terms of online communication size, though BlitzBough has a larger complexity than the two frameworks proposed by Bai et al.[3]. However, according to the evaluation results (presented in Section 6), BlitzBough still outperforms them in actual online communication size. This is because the two frameworks proposed by Bai et al. have larger constants, which are primarily introduced by arithmetic shares and boolean shares conversions, in complexity compared to BlitzBough. Nevertheless, the larger constants are

not reflected in the asymptotic complexity as they do not scale with the number of nodes. Finally, all the frameworks require $O(d)$ communication rounds. That is because all these frameworks require processing inferences layer by layer.

Table 1. The communication size and round of `BlitzBough` and the baseline frameworks. We incorporate the one-time setup phase of Bai et al.'s HE-based and PRF-based framework for the user into the offline phase, since this one-time setup phase needs to be performed for each user, and each user typically launches only one or a few inferences.

Frameworks	Communication Size			Rounds
	Total	Online	Offline	
Ours (`BlitzBough`)	$O(d\ell(\sqrt{n}+\sqrt{m}))$	$O(d\ell(\sqrt{n}+\sqrt{m}))$	$O(d\lambda(\log n+\log m))$	$O(d)$
Bai et al. HE-based [3]	$O(d\lambda(n+m))$	$O(d\ell\lambda)$	$O(d\lambda(n+m))$	$O(d)$
Bai et al. PRF-based [3]	$O(d\lambda(n+m))$	$O(d\ell)$	$O(d\lambda(n+m))$	$O(d)$
Ma et al. [21]	$O(d\ell*(\lambda+m))$	$O(d\ell*(\lambda+m))$	$O(n(\ell+\log m+\lambda+d))$	$O(d)$

5 Security Analysis

`BlitzBough` is secure under a three-party semi-honest model with an honest majority, i.e., the adversary $\mathcal{A}$ statically corrupts at most one of P_0, P_1, P_2 and follows the protocol specification.

Ideal functionality and leakage. The core building block of `BlitzBough` is a secret-shared oblivious selection functionality $\mathcal{F}_{\text{sel}}$: on input an RSS-shared index $\langle idx\rangle$ and an RSS-shared vector $\langle\vec{e}\rangle$ of public size sz, it returns a fresh RSS sharing $\langle u\rangle$ of $u = \vec{e}[idx]$, revealing neither idx, $\vec{e}$, nor u. Consistent with prior work [3,21], the leakage of `BlitzBough` consists of the public parameters only: the tree depth d, the number of attributes m, and the padded size of each layer vector, i.e., an upper bound on the number of nodes per layer rather than the exact count.

All protocols, except *ToOneHot* (Protocol 4) and *IndexConversion* (Protocol 3), are built on security-proven primitives from Section 2.2. We prove the security of the two remaining protocols in the standard real/ideal paradigm.

Proof. Let $\mathcal{A}$ be a semi-honest adversary corrupting at most one party, and $\mathcal{S}$ the simulator, given only the corrupted party's input and output shares.

ToOneHot. The messages received are: in the offline phase, P_b receives (key_b, r_b) from P_2 for $b \in \{0,1\}$; in the online phase, P_0 and P_1 exchange tmp_0, tmp_1 (Line 2), and all parties receive RSS shares of $\overrightarrow{arr_0}, \overrightarrow{arr_1}$ (Line 5). If $\mathcal{A}$ corrupts P_0, $\mathcal{S}$ samples $\widetilde{r}_0$ uniformly and generates $\widetilde{key}_0$ on an independently sampled point function; by the security of DPF [7], a single key is computationally indistinguishable from one generated for any other point. For the online phase, $\mathcal{S}$ samples $\widetilde{tmp}_1$ and $\langle\overrightarrow{arr_1}\rangle_0$ uniformly, which is a perfect simulation: $tmp_1 = (\langle idx\rangle_2 + r_1) \bmod 2^\ell$ is masked by the uniform r_1, held only by P_1 and

P_2, and any two RSS shares of a fixed secret are uniformly distributed. The cases of P_1 and P_2 are symmetric, with P_2 receiving only the Line 5 shares.

IndexConversion. Lines 1–3 are local computations, and Lines 5, 6, 8, and 9 are linear operations on RSS shares; none involves communication. Line 7 invokes the *LT* protocol and a share multiplication, and Lines 10–11 invoke *ToOneHot*. We analyze *IndexConversion* in the hybrid model where these subprotocols are replaced by calls to their respective ideal functionalities; their security follows from the primitives of Section 2.2 and from the argument above, respectively. Hence the only interactive step outside these subprotocols is Line 4, where P_0 RSS-shares (lp_0, hp_0) and P_1 RSS-shares (lp_1, hp_1, b). For each corruption case, $\mathcal{S}$ replaces the RSS shares received by the corrupted party with uniform ring elements. Since a single party holds only two of the three RSS shares of each value, and any two shares are uniformly distributed regardless of the secret, the simulated view is identically distributed to the real one; in particular the corrupted party learns nothing about hp, lp, or the error indicator b, and therefore nothing about idx.

In both protocols, every message received by the corrupted party is either a uniform ring element, an RSS share, or a value masked by a uniform secret held by an honest party, so the simulated and real views are indistinguishable. □

Composition. *DecisionTreeInference* (Protocol 1) invokes only RSS linear operations and share multiplications, the *LT* protocol, and *ElementSelection* / *ElementSelection*$^+$, which in turn invoke *IndexConversion* and *ToOneHot*. Each component securely realizes its ideal functionality, and every intermediate value passed between components remains RSS-shared and is never reconstructed. Since the components are invoked sequentially, security of the full protocol follows from the sequential composition theorem for semi-honest protocols, and the adversary's view consists solely of shares it is entitled to hold.

6 Evaluation

6.1 Implementation and Experiment Setting

Implementation: We implement `BlitzBough` in *C++* using the open-source function secret sharing library `libfss`[3]. We perform the computation of `BlitzBough` on a ring $\mathbb{Z}_{2^\ell}$ with $\ell = 64$, and set the security parameter λ to 128. These parameters are the same with the literatures [21,3]. Besides, we set $SizeThreshold = 64$ in *ElementSelection* (Protocol 2), which is the size at which converting the index into two one-hot vectors becomes cheaper than a single conversion.

Experiment environment: We conduct experiments on a Linux server equipped with a 2.4 GHz Intel Xeon CPU. For fair comparisons with the baselines [3,21], we limit the maximum number of used cores to 4 and the maximum used RAM to 16GB by leveraging the `taskset` tool[4] and the `ulimit` tool[5], respectively. As for

[3] `https://github.com/frankw2/libfss`

[4] `https://www.man7.org/linux/man-pages/man1/taskset.1.html`

[5] `https://www.man7.org/linux/man-pages/man1/ulimit.1p.html`

the network setting, we simulate three scenarios: (1) a local-area network (LAN) setting with a bandwidth of 1Gbps and 0.1ms round-trip time (RTT) latency; (2) a metropolitan-area network (MAN) setting with a bandwidth of 100Mbps and 6ms round-trip time (RTT) latency; (3) a wide-area network (WAN) setting with 40Mbps bandwidth and 80ms RTT latency.

Datasets and tree parameter: As is shown in Table 2, we use eight real-world datasets from the UCI repository and adopt the same tree parameters as those in the work [3] to evaluate `BlitzBough`.

Table 2. Datasets and corresponding tree parameters used in evaluation

Dataset	Wine	Linnerud	Breast	Digits	Spambase	Diabetes	Boston	MNIST
#Attributes m	7	3	12	47	57	10	13	784
#Tree Depth d	5	6	7	15	17	28	30	20
#Nodes n	23	39	43	337	171	787	851	4179

6.2 Performance Evaluation

Baselines. We benchmark `BlitzBough` against three state-of-the-art frameworks: Bai et al.'s HE-based and PRF-based frameworks [3] (*Bai's HE* and *Bai's PRF* for short), and Ma et al.'s OT-based framework [21] (*Ma's framework* for short). Since their source codes are not publicly available, we adopt the experimental results reported in [3], where Bai's frameworks were evaluated on an Intel Core i9-9900 (16 cores, 3.10 GHz, 32 GB RAM) and Ma's on a Ryzen 7 3700x (4 cores, 3.60 GHz, 16 GB RAM). To ensure fairness, we cap our environment to 4 cores and 16 GB RAM (matching the weakest baseline), so any running-time advantage of `BlitzBough` cannot be attributed to stronger hardware.

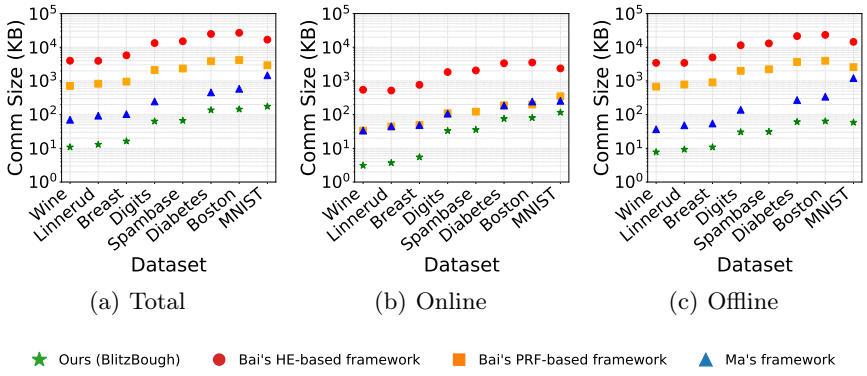

Fig. 3. Communication size (KB) of `BlitzBough` and the baseline frameworks.

Communication size: As shown in Figure 3, BlitzBough outperforms the three frameworks by $3.3\times \sim 35.1\times$, $2.2\times \sim 12.0\times$, and $4.4\times \sim 71.4\times$ in total, online, and offline communication size, respectively. Although Bai's HE-based and PRF-based frameworks have lower asymptotic online complexity (Table 1), their arithmetic-boolean share conversions introduce a large constant overhead, and Ma's framework incurs online cost proportional to the number of attributes due to 1-out-of-m OT. For offline communication, the three baselines all transmit an encrypted or secret-shared array covering all nodes, whereas BlitzBough only requires communication proportional to $\sqrt{n}$ thanks to our efficient element selection protocol.

Running time: As is shown in Figure 4, when running in the LAN setting, BlitzBough notably outperforms the three frameworks by $1.3\times \sim 8.2\times$. When running in the MAN and WAN settings, BlitzBough is marginally slower than the most efficient frameworks (Ma's framework in MAN and Bai's HE-based framework in WAN). The main reason is that when executing a single inference, the running time in the LAN setting is predominantly influenced by the communication size, while the running time in the MAN and WAN settings is predominantly influenced by the communication rounds. Though BlitzBough significantly outperforms the three frameworks in terms of communication size, the communication rounds of BlitzBough are a little larger than those of Ma's framework and Bai's HE-based framework, since BlitzBough requires performing conversions between FSS and RSS.

However, in real-world applications, the importance of communication size will be amplified, especially in bandwidth-limited scenarios involving decision trees with considerable nodes or multiple inference queries launched nearly simultaneously. In these cases, a smaller communication size will result in a faster response time.

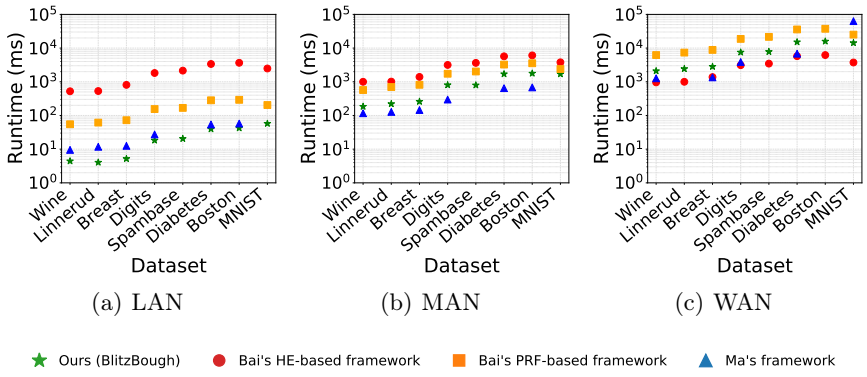

Fig. 4. Online running time (ms) of BlitzBough and the baseline frameworks.

7 Discussion

Practicality of the security model. In this paper, we adopt a semi-honest security model with an honest majority to implement `BlitzBough`. Though this security model is weaker than some privacy-preserving inference frameworks [21,6,3], its security assumption should be practical. For example, the industry-standard MPC framework SecretFlow (developed by Ant Group) also adopts this security model.

Extension to support more parties. `BlitzBough` can be extended to support more parties. Note that except for the protocols *IndexConversion* and *ToOneHot*, the other protocols in `BlitzBough` can be implemented using Shamir secret sharing [5], which supports three or more parties. To implement the protocols *IndexConversion* and *ToOneHot* using Shamir secret sharing, we can leverage the secure division protocol [11] and equality test protocol [10]. Using this method, the communication size of `BlitzBough` is still proportional to the square root of the number of decision tree nodes in scenarios with three or more parties.

Future work: In the future, we will extend `BlitzBough` to support more tree-based models, such as XGBoost [12] and more parties. Furthermore, the FSSTree framework proposed by Fu et al. [15] successfully extends Function Secret Sharing (FSS) to the malicious security model. In future work, we plan to refer to such paradigms and introduce lightweight verifiable mechanisms into our protocol to defend against malicious adversaries.

8 Related Work

Existing privacy-preserving decision tree inference frameworks fall into two categories: communication-intensive (based on MPC) and computation-intensive (based on homomorphic encryption).

Communication-intensive frameworks. Early garbled-circuit-based frameworks [9,4] transmit (parts of) an encrypted tree to the feature provider, incurring communication proportional to the number of nodes. Liu et al. [19] eliminate node transmission via secret sharing but require a secure comparison per node. Ma et al. [21] use OT to reduce comparisons, yet still send an encrypted tree to the user in the offline phase. Bai et al. [3] introduce an oblivious selection functionality that reduces online communication to $O(d)$, but their offline phase still transmits a secret-shared array covering all nodes. In contrast, `BlitzBough` requires only $O(\sqrt{n})$ communication, making it more efficient in bandwidth-limited scenarios.

Computation-intensive frameworks. HE-based approaches [6,20,24,14] avoid most interaction but suffer heavy computation cost. Bost et al. [6] treat the tree as a high-degree polynomial evaluated under FHE; Lu et al. [20] target non-interactivity at the cost of substantial client-side overhead; Tueno et al. [24] amortize cost via BGV [8] with SIMD slots [23]; and Cong et al. [14] employ a fast homomorphic comparison. All four nonetheless rely on FHE for most operations, whereas `BlitzBough` relies solely on secret sharing and thus achieves substantially better computational efficiency.

9 Conclusion

In this paper, we propose BlitzBough, an efficient inference framework for decision trees with privacy preservation. The key component of BlitzBough is an efficient element selection protocol. With this protocol, BlitzBough reduces the communication size of existing frameworks from proportional to the number of decision tree nodes to proportional to the square root of the number of decision tree nodes. Experimental evaluations with various settings on the number of decision tree nodes demonstrate that BlitzBough outperforms the state-of-the-art frameworks by up to 35.1× in terms of communication size and up to 8.2× in terms of running time.

Acknowledgments

This paper is supported by the National Cryptologic Science Fund of China (2025NCSF01010), and the Natural Science Foundation of China (92370120). We thank all anonymous reviewers for their insightful comments.

References

1. Arya, M., Sastry, H.G.: Stock indices price prediction in real time data stream using deep learning with extra-tree ensemble optimisation. International Journal of Computational Science and Engineering **25**(2), 140–151 (2022)
2. Azar, A.T., El-Metwally, S.M.: Decision tree classifiers for automated medical diagnosis. Neural Computing and Applications **23**, 2387–2403 (2013)
3. Bai, J., Song, X., Cui, S., Chang, E.C., Russello, G.: Scalable private decision tree evaluation with sublinear communication. In: Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security. pp. 843–857 (2022)
4. Barni, M., Failla, P., Kolesnikov, V., Lazzeretti, R., Sadeghi, A.R., Schneider, T.: Secure evaluation of private linear branching programs with medical applications. In: Proceedings of ESORICS 2009. vol. 5789, pp. 424–439 (2009)
5. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for noncryptographic fault-tolerant distributed computations. In: Proceedings of the 20th Annual Symposium on the Theory of Computing (STOC'88). pp. 1–10 (1988)
6. Bost, R., Popa, R.A., Tu, S., Goldwasser, S.: Machine learning classification over encrypted data. Cryptology ePrint Archive (2014)
7. Boyle, E., Chandran, N., Gilboa, N., Gupta, D., Ishai, Y., Kumar, N., Rathee, M.: Function secret sharing for mixed-mode and fixed-point secure computation. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 871–900. Springer (2021)
8. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (leveled) fully homomorphic encryption without bootstrapping. ACM Transactions on Computation Theory (TOCT) **6**(3), 1–36 (2014)
9. Brickell, J., Porter, D.E., Shmatikov, V., Witchel, E.: Privacy-preserving remote diagnostics. In: Proceedings of the 14th ACM conference on Computer and communications security. pp. 498–507 (2007)

10. Catrina, O., De Hoogh, S.: Improved primitives for secure multiparty integer computation. In: International Conference on Security and Cryptography for Networks. pp. 182–199. Springer (2010)
11. Catrina, O., Saxena, A.: Secure computation with fixed-point numbers. In: International Conference on Financial Cryptography and Data Security. pp. 35–50. Springer (2010)
12. Chen, T., Guestrin, C.: Xgboost: A scalable tree boosting system. In: Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining. pp. 785–794 (2016)
13. Cheng, N., Gupta, N., Mitrokotsa, A., Morita, H., Tozawa, K.: Constant-round private decision tree evaluation for secret shared data. Proceedings on Privacy Enhancing Technologies (2024)
14. Cong, K., Das, D., Park, J., Pereira, H.V.: Sortinghat: Efficient private decision tree evaluation via homomorphic encryption and transciphering. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security. pp. 563–577 (2022)
15. Fu, J., Cheng, K., Xia, Y., Song, A., Li, Q., Shen, Y.: Private decision tree evaluation with malicious security via function secret sharing. In: European Symposium on Research in Computer Security (ESORICS). pp. 310–330. Springer (2024)
16. Innovation & Tech Today: Top 10 machine learning algorithms in 2024. `https://innotechtoday.com/top-10-machine-learning-algorithms-in-2024/` (2024), accessed: 2025-10-25
17. Koh, H.C., Tan, W.C., Goh, C.P.: A two-step method to construct credit scoring models with data mining techniques. International Journal of Business and Information **1**(1), 96–118 (2006)
18. Lee, C.S., Cheang, P.Y.S., Moslehpour, M.: Predictive analytics in business analytics: decision tree. Advances in Decision Sciences **26**(1), 1–29 (2022)
19. Liu, L., Chen, R., Liu, X., Su, J., Qiao, L.: Towards practical privacy-preserving decision tree training and evaluation in the cloud. IEEE Transactions on Information Forensics and Security **15**, 2914–2929 (2020)
20. Lu, W.j., Zhou, J.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. pp. 67–74 (2018)
21. Ma, J.P., Tai, R.K., Zhao, Y., Chow, S.S.: Let's stride blindfolded in a forest: Sublinear multi-client decision trees evaluation. In: NDSS (2021)
22. Mohassel, P., Rindal, P.: Aby3: A mixed protocol framework for machine learning. In: Proceedings of the 2018 ACM SIGSAC conference on computer and communications security. pp. 35–52 (2018)
23. Smart, N.P., Vercauteren, F.: Fully homomorphic simd operations. Designs, codes and cryptography **71**, 57–81 (2014)
24. Tueno, A., Boev, Y., Kerschbaum, F.: Non-interactive private decision tree evaluation. In: IFIP Annual Conference on Data and Applications Security and Privacy. pp. 174–194. Springer (2020)
25. Tueno, A., Kerschbaum, F., Katzenbeisser, S.: Private evaluation of decision trees using sublinear cost. Proceedings on Privacy Enhancing Technologies **1**, 266–286 (2019)
26. Zhou, L., Wang, Z., Cui, H., Song, Q., Yu, Y.: Bicoptor: Two-round secure three-party non-linear computation without preprocessing for privacy-preserving machine learning. In: Proceedings of the 2023 IEEE Symposium on Security and Privacy. pp. 534–551 (2023)