`Ruyi`: A Configurable and Efficient Secure Multi-Party Learning Framework with Privileged Parties

Lushan Song, Zhexuan Wang, GuopengLin, Weili Han, *Member, IEEE*

***Abstract*—Secure multi-party learning (MPL) enables multiple parties to train machine learning models with privacy preservation. MPL frameworks typically follow the peer-to-peer architecture, where each party has the same chance to handle the results. However, the cooperative parties in business scenarios usually have unequal statuses. Thus, Song *et al.* (CCS'22) presented pMPL, a hierarchical MPL framework with a privileged party. Nonetheless, pMPL has two limitations: (i) it has limited configurability requiring manually finding a public matrix that satisfies four constraints, which is difficult when the number of parties increases, and (ii) it is inefficient due to the huge online communication overhead.**

In this paper, we are motivated to propose Ruyi, a configurable and efficient MPL framework with privileged parties. Firstly, we reduce the public matrix constraints from four to two while ensuring the same privileged guarantees by extending the standard resharing paradigm to vector space secret sharing in order to implement the share conversion protocol and performing all the computations over a prime field rather than a ring. This enhances the configurability such that the Vandermonde matrix can always satisfy the public matrix constraints when given the number of parties, including privileged parties, assistant parties, and assistant parties allowed to drop out. Secondly, we reduce the online communication overhead by adapting the masked evaluation paradigm to vector space secret sharing. Experimental results demonstrate that Ruyi is configurable with multiple parties and outperforms pMPL by up to $53.87\times$, $13.91\times$, and $2.76\times$ for linear regression, logistic regression, and neural networks, respectively.

***Index Terms*—Secure Multi-party Learning, Privacy-preserving Machine Learning, Configurable.**

I. Introduction

Recently, privacy protection regulations and laws, such as the General Data Protection Regulation (GDPR) [1], have imposed stringent restrictions on the direct sharing of data between different organizations or institutions. In order to utilize data distributed among multiple parties to train high-performance machine learning models with privacy preservation, privacy-preserving machine learning based on secure multi-party computation (MPC) [2], referred to as secure multi-party learning (MPL) [3], has emerged as a critical tool. Due to the rigorous security guarantee provided by MPC, MPL enables multiple parties to obtain the trained model while keeping the privacy of their raw data.

In the past few years, researchers have proposed numerous MPL frameworks [4]–[7]. While most of these MPL frameworks, such as `SecureML` [4], `ABY`^{3} [5] and `ABY2.0` [6] follow the peer-to-peer architecture, where each party involved in the training process has the same chance to handle the results, including intermediate results and the final model after training. For example, in `ABY`^{3}, any two parties can collude with each other to reveal the secret. However, a hierarchical architecture where the cooperative parties in business scenarios have unequal statuses is common. In this architecture, some powerful parties (i.e. privileged parties) require to dominate the training process and obtain the trained models due to the payment to other parties (i.e. assistant parties).

To support the hierarchical architecture in common businesses, Song *et al.* [7] proposed a 3PC framework with a privileged party, known as `pMPL`. It provides the privileged party a privileged position that guarantees: (i) only the privileged party can obtain the final trained model, even if assistant parties collude with each other; (ii) the training process can continue to the end even if one assistant party drops out, which saves computing resources. The above two guarantees ensure that the privileged party can protect its business interests even though the two assistant parties collude with each other or one of the assistant parties quits intentionally.

However, `pMPL` has two limitations. The first one is its limited configurability. `pMPL` requires manually finding a public matrix that satisfies four constraints, which is difficult when the number of parties increases. This poses challenges in practical scenarios where the composition of parties may be more complex. For instance, more privileged parties and assistant parties may be involved in the training process to provide more data. At the same time, whenever the number of parties changes, it is required to manually find a different public matrix that satisfies the four constraints again, which is very time-consuming. The second one is `pMPL` is inefficient. In order to ensure the above two guarantees, it proposes a series of protocols with high online communication overhead, including communication rounds and communication sizes.

A. Our Contributions

In this paper, we are motivated to overcome the above limitations by proposing a configurable and efficient MPL framework with privileged parties, referred to as `Ruyi` [1].

Firstly, we reduce the public matrix constraints from four to two while ensuring the same privileged guarantees by extending the standard resharing paradigm to vector space secret sharing in order to implement the share conversion

Manuscript received 29 January 2024; revised 03 June 2024 and 20 August 2024; accepted 04 October 2024. This work was supported in part by the National Natural Science Foundation of China under Grant 92370120, Grant 62172100, and in part by the National Key Research and Development Program of China under Grant 2023YFC3304400. (*Corresponding author: Weili Han.*)

Lushan Song, Zhexuan Wang, Guopeng Lin, and Weili Han are with the School of Computer Science, Fudan University, Shanghai 10246, China. (e-mail: 19110240022@fudan.edu.cn; 21210240331@m.fudan.edu.cn; 17302010022@fudan.edu.cn; wlhan@fudan.edu.cn).

[1]`Ruyi` is an artifact, which represents privileges, in the traditional Chinese culture.

TABLE I
COMMUNICATION ROUNDS AND TOTAL COMMUNICATION SIZES (BITS) COST OF BUILDING BLOCKS IN RUYI AND pMPL. HERE, "3PC", "3PC$_{d1}$", AND "nPC" STAND FOR THREE PARTIES WITH NO ASSISTANT PARTY DROPPING OUT, THREE PARTIES WITH ONE ASSISTANT PARTY DROPPING OUT, AND n PARTIES, RESPECTIVELY. q, k, AND ℓ DENOTE THE SIZE OF THE PRIME FIELD, THE BIT-WIDTH OF THE SIGNED INTEGER, AND THE BIT-WIDTH OF THE RING, RESPECTIVELY. l IS THE VECTOR LENGTH. $a \times b, b \times c$ ARE THE SIZES FOR THE LEFT AND RIGHT INPUTS OF MATRIX-BASED COMPUTATIONS. "LINEAR OPERATION", "DOT PRODUCT", AND "MATRIX MULT." STAND FOR LINEAR OPERATION ON MATRIX, DOT PRODUCT WITH TRUNCATION, AND MATRIX MULTIPLICATION WITH TRUNCATION, RESPECTIVELY. RELU AND SIGMOID ARE EXECUTED ON A SINGLE VALUE. "ROUN." STANDS FOR ONLINE COMMUNICATION ROUNDS AND "COMMU." STANDS FOR TOTAL ONLINE COMMUNICATION SIZES.

	Dot Product		Matrix Mult.		ReLU		Sigmoid	
	Roun.	Commu.	Roun.	Commu.	Roun.	Commu.	Roun.	Commu.
Ruyi (3PC)	1	$6\lceil q \rceil$	1	$6ac\lceil q \rceil$	4	$6(k+2)\lceil q \rceil$	5	$12(k+2)\lceil q \rceil$
pMPL (3PC)	2	$14l\ell$	2	$(6(ab+bc)+2ac)\ell$	$\log \ell + 6$	$(34+24\log \ell)\ell + 6$	$\log \ell + 8$	$(70+48\log \ell)\ell + 18$
Ruyi (3PC$_{d_1}$)	1	$3\lceil q \rceil$	1	$3ac\lceil q \rceil$	4	$3(k+2)\lceil q \rceil$	5	$6(k+2)\lceil q \rceil$
pMPL (3PC$_{d_1}$)	2	$7l\ell$	2	$(3(ab+bc)+ac)\ell$	$\log \ell + 5$	$(16+12\log \ell)\ell + 3$	$\log \ell + 7$	$(32+24\log \ell)\ell + 9$
Ruyi (nPC)	1	$n(n-1)\lceil q \rceil$	1	$n(n-1)ac\lceil q \rceil$	4	$n(n-1)(k+2)\lceil q \rceil$	5	$2n(n-1)(k+2)\lceil q \rceil$
MPClan(nPC)	1	$2t\ell$	1	$2act\ell$	$2\log_4 \ell + 2$	$(2\ell + 230)t$	$2\log_4 \ell + 4$	$(4\ell + 460)t$
MD-ML(nPC)	1	$n(n-1)\ell$	1	$n(n-1)ac\ell$	$\log \ell + 3$	$n(n-1)(4\ell s + 6\ell - 2s - 3)$	$log\ell + 4$	$2n(n-1)(4\ell s + 6\ell - 2s - 3)$

protocol and performing all the computations over a prime field rather than a ring. This enhances the configurability such that the Vandermonde matrix can always satisfy the public matrix constraints when given the number of parties, including privileged parties, assistant parties, and assistant parties allowed to drop out. Besides, we can configure the security model (honest majority or dishonest majority) by configuring the number of parties. Therefore, we could adapt more complex party compositions in practical scenarios and save the time required for manually finding the public matrix before each training when the number of parties changes. The configurability feature in Ruyi provides more support for (future) cooperation among companies when the number of parties exceeds 3 and the composition of parties is more complex. For example, two banks cooperate with a FinTech company, a credit bureau, and a securities firm to train a financial risk management model, where two banks are privileged parties and the other companies are assistant parties.

Secondly, we reduce the online communication overhead by adapting the masked evaluation paradigm to vector space secret sharing, which leverages an input-independent but function-dependent offline phase, similar to [6], [8]–[12]. Then we construct a series of basic primitives, such as secure multiplication, truncation, and comparison. Furthermore, we construct several building blocks for secure machine learning training based on these basic primitives, such as matrix multiplication with truncation and activation functions. Our building blocks outperform pMPL [7] in terms of communication rounds and sizes. The efficiency improvement of these building blocks all stems from the adaptation of the masked evaluation paradigm to vector space secret sharing.

The detailed online communication overhead is shown in Table I. We compare Ruyi with pMPL for 3PC, and Ruyi with MPClan [11] and MD-ML [13] for nPC, where MPClan considers an honest majority while MD-ML considers a dishonest majority.

In summary, we highlight our contributions as follows:

- **Configurability.** We reduce the public matrix constraints from four to two while ensuring the same privileged guarantees by extending the standard resharing paradigm to vector space secret sharing in order to implement the share conversion protocol and performing all the computations over a prime field rather than a ring. This enhances the configurability such that the Vandermonde matrix can always satisfy the public matrix constraints when given the number of parties. Besides, we enhance the configurability of the security model (honest majority or dishonest majority) by configuring the number of assistant parties allowed to drop out.
- **Efficiency.** We reduce the online communication overhead and thus improve the online efficiency of primitives by adapting the masked evaluation paradigm to vector space secret sharing. Besides, we construct several improved basic primitives and building blocks to support machine learning training.

Experimental Results. We evaluate Ruyi[2] in training linear regression, logistic regression, and neural networks on the MNIST dataset. Experimental results demonstrate that Ruyi is configurable with multiple privileged (assistant) parties, multiple parties, and multiple assistant parties allowed to drop out. Besides, Ruyi outperforms pMPL by up to 53.87×, 13.91×, and 2.76× for linear regression, logistic regression, and neural networks, respectively.

B. Related Work

Secure multi-party learning (privacy-preserving machine learning based on MPC) has gained much attention recently. There are lots of MPL frameworks based on different secret-sharing technologies, such as additive secret sharing, replicated secret sharing, and vector space secret sharing.

MPL Frameworks Based on Additive Secret Sharing. Mohassel and Zhang [4] proposed a two-party MPL framework SecureML based on the additive secret sharing that supports training various machine learning models. In the offline phase, they utilize the input-independent and function-independent Beaver's multiplication triplet [14] to improve the online efficiency. In addition, the secure multiplication protocol in SecureML requires one round of communication in the online phase. Wagh *et al.* [15] proposed a secure three-party learning framework SecureNN that introduces a

[2]We open our implementation codes at GitHub (https://github.com/FudanMPL/RuYi.git)

third party to assist computations in improving online efficiency. Turbospeedz [8] and ABY2.0 [6] utilize an input-independent but function-dependent offline phase to reduce online communication from four to two elements per secure multiplication gate. However, in the above frameworks [4], [6], [8], [15], which follow the peer-to-peer architecture, any party cannot drop out during the training. On the other hand, Ruyi follows the hierarchical architecture, where nd assistant parties are allowed to drop out during the training.

MPL Frameworks Based on Replicated Secret Sharing. Mohassel and Rindal [5] proposed ABY^3, a secure three-party learning framework based on replicated secret sharing. It aims to support efficient switching among arithmetic sharing, Boolean sharing, and Yao's sharing. Trident [16] extends ABY^3 to four-party scenarios. By introducing an extra honest party, Trident improves the online efficiency of ABY^3. Wagh *et al.* [17] presented Falcon, one of the most efficient MPL frameworks based on replicated secret sharing. Falcon combines techniques from SecureNN and ABY^3, which results in improved protocol efficiency. Dalskov *et al.* [18] demonstrated Fantastic Four to guarantee the security with abort by finding and excluding the (potential) malicious parties and delegating the training to the remaining semi-honest parties. These MPL frameworks [5], [16]–[18] based on replicated secret sharing all follow the peer-to-peer architecture and theoretically tolerate the dropping out of one party or parties. However, they cannot guarantee only the privileged parties can obtain the final trained model as any two parties can reconstruct it. On the other hand, Ruyi guarantees only the privileged parties can obtain the final trained model, even if assistant parties collude with each other.

MPL Frameworks Based on Vector Space Secret Sharing. Song *et al.* [7] proposed an MPL framework pMPL based on vector space secret sharing. pMPL considers a hierarchical architecture with a privileged party and two assistant parties that guarantees: (i) only the privileged party can obtain the final trained model, even if assistant parties collude with each other; (ii) the training process could be continued to the end, even if one assistant party drops out. pMPL breaks through the fundamental limitations where the peer-to-peer architecture MPL frameworks do not conform to common business scenarios. However, the limited configurability of pMPL requires manually finding a public matrix that satisfies four constraints, which is difficult when the number of parties increases. Besides, it is inefficient due to the huge online communication overhead. Ruyi improves the configurability and efficiency of pMPL. Furthermore, pMPL can be simply achieved by the privileged party P_1 and one assistant party P_2 performing additive secret sharing-based 2PC protocols, and the other assistant party P_3 holds the same shares as P_2.

Other MPL Frameworks for Asymmetric Trust Setting. Some works consider an asymmetric trust setting. For instance, Dong *et al.* [19] proposed Fusion, which focuses on two-party secure inference in an asymmetric trust setting, where the server is malicious and may deviate from the protocol while the client is honest. Brüggemann *et al.* [20] explored a fixed-corruption scenario for honest majority 3PC, where only one dedicated party can be maliciously corrupted. They considered two cases where the helper or the client is malicious and proposed two efficient security inference protocols, one for each case. Rather than the asymmetric trust setting, we consider the asymmetric status setting. In our scenario, privileged parties have more power, which means only they can obtain the final model, and part assistant parties are allowed to drop out. Besides, all parties follow a symmetrical trust setting, where all the parties, including privileged parties and assistant parties, are assumed to be semi-honest.

II. Preliminaries

MPC is a powerful tool that enables multiple parties to cooperatively compute a function without revealing any sensitive information except for the final results. Secret sharing is one of the widely used MPC techniques. The main idea of secret sharing is to break a secret value into several shares, each of which is held by a party. Furthermore, the parties perform computations on these shares to obtain the final result while preserving privacy. Secret sharing comes in many forms, e.g. additive secret sharing and vector space secret sharing. In this paper, we follow pMPL [7] to use vector space secret sharing as the fundamental technology for our hierarchical MPL framework Ruyi with privileged parties.

A. Vector Space Secret Sharing

Brickell [21] proposed vector space secret sharing for hierarchical access structure. For a set of parties $\mathcal{P} = \{P_1, \ldots, P_n\}$, in vector space secret sharing, only authorized sets $B_1, \ldots, B_k$ (subsets of $\mathcal{P}$) can reconstruct the secret value from their shares. The family of authorized sets is called the access structure $\varGamma$, i.e. $\varGamma = \{B_1, \ldots, B_m\}$. Suppose there is a public function $\varPhi : \mathcal{P} \to (\mathbb{F}_p)^d$, where $\mathbb{F}_p$ is a field with the size of p (a large prime number), d is a positive integer and $d \geq 2$. Each party P_i is associated with a public vector $\varPhi(P_i)$. Besides, we denote the matrix constructed by the public vectors as public matrix $\varPhi(\mathcal{P})$. This function satisfies the following property:

$$\begin{aligned}&(1, 0, \ldots, 0) \textit{ can be expressed as a linear combination}\\ &\textit{of elements in the set } \{\varPhi(P_i) \mid P_i \in B_j\} \Leftrightarrow B_j \in \varGamma\end{aligned} \quad (1)$$

That is, a set B_j is an authorized set, i.e. $B_j \in \varGamma$, if and only if vector $(1, 0, \ldots, 0)$ can be expressed as a linear combination of all the public vectors in the set $\{\varPhi(P_i) \mid P_i \in B_j\}$. Therefore, there exist m unique public constants $c_1, ..., c_m$ (which we refer to as reconstruction coefficients in this paper), where m is the number of parties in B_j such that:

$$(1, 0, \ldots, 0) = \sum_{P_i \in B_j} c_i \cdot \varPhi(P_i) \quad (2)$$

Sharing and Reconstruction. To secret share a secret value x, the party holding this secret value first generates $d-1$ random values $s_1, \ldots, s_{d-1} \in \mathbb{Z}_p$. Then it constructs a column vector $\vec{s} = (x, s_1, \ldots, s_{d-1})$. After that, this party computes the share $[x]_i = \varPhi(P_i) \cdot \vec{s}$ for $P_i \in \mathcal{P}$.

As the first element in vector $\vec{s}$ is the secret value x, $(1, 0, \ldots, 0) \cdot \vec{s} = x$. To reconstruct the secret value x, the parties in the authorized set B_j exchange their shares and compute:

$$x = (1,0,\ldots,0)\cdot\vec{s} = \Big(\sum_{P_i\in B_j} c_i\cdot\Phi(P_i)\Big)\cdot\vec{s} \\ = \sum_{P_i\in B_j} c_i\cdot[x]_i \tag{3}$$

An example is shown in Figure 1. Assuming $\mathcal{P} = \{P_1, P_2, P_3\}$, access structure $\Gamma = \{B_1\} = \{\{P_1, P_2, P_3\}\}$, and P_1 is the party who holds the secret value $x = 5$.

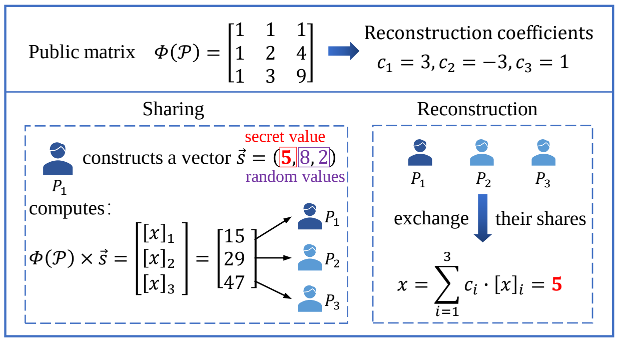
Public matrix $\Phi(\mathcal{P}) = \begin{bmatrix}1&1&1\\1&2&4\\1&3&9\end{bmatrix}$
Reconstruction coefficients
$c_1 = 3, c_2 = -3, c_3 = 1$
Sharing
Reconstruction
secret value
constructs a vector $\vec{s} = (5, 8, 2)$
random values
P_1
computes:
$\Phi(\mathcal{P})\times\vec{s} = \begin{bmatrix}[x]_1\\ [x]_2\\ [x]_3\end{bmatrix} = \begin{bmatrix}15\\29\\47\end{bmatrix}$
P_1
P_2
P_3
exchange
their shares
$x = \sum_{i=1}^{3} c_i\cdot[x]_i = 5$

Fig. 1. An example of sharing and reconstruction of vector space secret sharing.

B. Designs in pMPL

In pMPL, $\mathcal{P} = \{P_1, P_2, P_3\}$, where P_1 is the privileged party, P_2 and P_3 are assistant parties, public matrix $\Phi(\mathcal{P})$ is a 4 × 3 matrix. For each party P_i, the i-th row $\Phi(i)$ of $\Phi(\mathcal{P})$ is its corresponding public vector, and $\Phi(4)$ is the alternate public vector corresponding to the privileged party P_1.

Next, we show the public matrix constraints in pMPL [7]. The public matrix $\Phi(\mathcal{P})$ should satisfy four constraints.

1) The vector $(1,0,0)$ can be represented as a linear combination of the public vectors in the set $\{\Phi(1),\Phi(2),\Phi(3)\}$, where these public vectors are linearly independent. In other words, there exist three non-zero reconstruction coefficients (public constants) c_1, c_2, c_3 such that $(1,0,0) = \sum_{i=1}^{n} c_i\cdot\Phi(i)$.
2) The alternate public vector $\Phi(4)$ can be represented as a linear combination of $\Phi(2)$ and $\Phi(3)$, i.e., $\Phi(4) = a_2\cdot\Phi(2) + a_3\cdot\Phi(3)$, where a_2, $a_3 \neq 0$. Besides, the public vectors in both the sets $\{\Phi(1), \Phi(2), \Phi(4)\}$ and $\{\Phi(1), \Phi(3), \Phi(4)\}$ are linearly independent. Besides, there exist six non-zero reconstruction coefficients (public constants) $c'_1, c'_2, c'_4, c''_1, c''_3, c''_4$, such that $(1,0,0) = \sum_{i=1,i\neq 2}^{4} c'_i\cdot\Phi(i) = \sum_{i=1,i\neq 3}^{4} c''_i\cdot\Phi(i)$.
3) The vector $(1,0,0)$ cannot be represented as a linear combination of the public vectors in both the sets $\{\Phi(1),\Phi(4)\}$ and $\{\Phi(2),\Phi(3)\}$.
4) As pMPL performs the computations over a ring $\mathbb{Z}_{2^\ell}$, both the values of public matrix $\Phi(\mathcal{P})$ and reconstruction coefficients $c_1,\ldots,c''_4$ should be elements of the ring $\mathbb{Z}_{2^\ell}$.

III. Overview of Ruyi

For clarity purpose, we show the notations used in this paper in Table II.

TABLE II
NOTATIONS USED IN THIS PAPER.

Symbol	Description
n	The number of all the parties.
np	The number of privileged parties.
na	The number of assistant parties where $na+np = n$.
nd	The number of assistant parties allowed to drop out, where $nd < na \leq np$.
$\mathcal{P} = \{P_1,\ldots,P_n\}$	A set of parties, where $\{P_1,\ldots,P_{np}\}$ are privileged parties, $\{P_{np+1},\ldots,P_n\}$ are assistant parties, $\{P_{d_1},\ldots,P_{d_{nd}}\}$ are the assistant parties who may drop out ($np+1 \leq d_1,\ldots,d_{nd} \leq n$).
$\Phi(\mathcal{P})$	The public matrix.
$c_{0,i}, c_{m,i}$	The reconstruction coefficients that are used to reconstruct the secret value ($m \in \{1,\ldots,nd\}$).
k	The bit-width of the signed integer $\bar{x}$.
f	The bit-width of the fractional part of the signed integer $\bar{x}$ ($f < k$).
$\mathbb{F}_q$	Prime field with size of q ($q > 2^k$).
$[x]$	$[\cdot]$-shares of the value x, such that $x = \sum_{i=1}^{n} c_{0,i}\cdot[x]_i = \sum_{i=1,i\neq d_1,\ldots,d_m}^{n+m} c_{m,i}\cdot[x]_i$ for $m \in \{1,\ldots,nd\}$.
$[\![x]\!] = ([\eta_x],\gamma_x)$	$[\![\cdot]\!]$-shares of the value x, such that $x = \gamma_x - \sum_{i=1}^{n} c_{0,i}\cdot[\eta_x]_i = \gamma_x - \sum_{i=1,i\neq d_1,\ldots,d_m}^{n+m} c_{m,i}\cdot[\eta_x]_i$ for $m \in \{1,\ldots,nd\}$.
$\langle x\rangle$	$\langle\cdot\rangle$-shares of the value x, such that $x = \sum_{i=1}^{n}\langle x\rangle$.

A. Architecture

As shown in Figure 2, we consider a scenario where n parties $\mathcal{P} = \{P_1,\ldots,P_n\}$, with $n \geq 3$, collaborate to train machine learning models on their raw data with privacy preservation. These parties are divided into two groups: np privileged parties and na assistant parties, where $np \geq 1$, $na = n - np \geq 2$, and $na \geq np$. Without loss of generality, we define the former np parties $\{P_1,\ldots,P_{np}\}$ as privileged parties, the remaining na parties $\{P_{np+1},\ldots,P_n\}$ as assistant parties. Besides, we define $\{P_{d_1},\ldots,P_{d_{nd}}\}$ as nd assistant parties who may drop out, where $np+1 \leq d_1,\ldots,d_{nd} \leq n$ and $nd < na$.

Before the training, all parties $\mathcal{P}$ set the public matrix for the given number of privileged parties, assistant parties, and assistant parties allowed to drop out. After that, all the parties $\mathcal{P}$ secret share their raw data with each other by using the sharing protocol, as described in Section IV-C. During the training, all parties $\mathcal{P}$ execute the building blocks required to train the machine learning models on their shares and output the shared models. After the training, assistant parties send their shared models to the privileged parties, and each privileged party sends its shared model to the other privileged parties. Then only the privileged parties can reveal the final trained model, while none of the assistant parties can obtain it. During the entire process, no party can access the raw data of any other party or infer any private information from the intermediate results. Besides, nd assistant parties are allowed to drop out during the online phase.

B. Data Representation

All the computations in Ruyi are performed over the fixed-point integers, and these integers are mapped into a prime

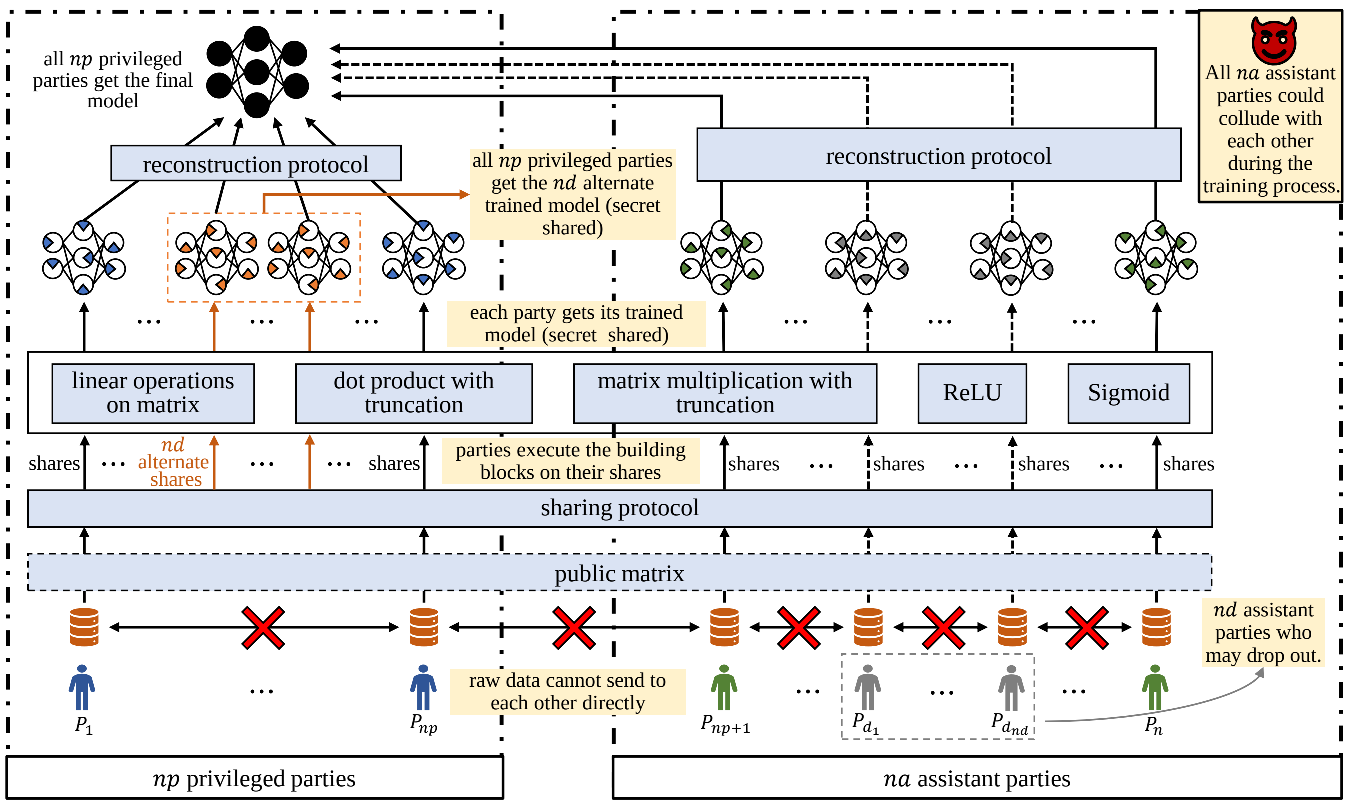
all np privileged parties get the final model
reconstruction protocol
all np privileged parties get the nd alternate trained model (secret shared)
reconstruction protocol
All na assistant parties could collude with each other during the training process.
each party gets its trained model (secret shared)
linear operations on matrix
dot product with truncation
matrix multiplication with truncation
ReLU
Sigmoid
shares
nd alternate shares
parties execute the building blocks on their shares
sharing protocol
public matrix
nd assistant parties who may drop out.
raw data cannot send to each other directly
P_1
P_{np}
P_{np+1}
P_{d_1}
$P_{d_{nd}}$
P_n
np privileged parties
na assistant parties

Fig. 2. Overview of Ruyi

field $\mathbb{F}_q$. Concretely, given a fixed-point rational number $\tilde{x}$, we first encode it as a signed integer $\bar{x} = \tilde{x} \cdot 2^f \in \mathbb{Z}_{2^k}$ (i.e., $-2^{k-1} \leq \bar{x} \leq 2^{k-1} - 1$), where k is the bit-width of $\bar{x}$ and f is the bit-width of fractional part. Then we encode the signed integer $\bar{x}$ into the field $\mathbb{F}_q$ using the mapping $x = \mathcal{F}(\bar{x}) = \bar{x} \mod q$, where q is the size of the prime field $\mathbb{F}_q$. Besides, the output of secure multiplication will double the bit-width, so $q > 2^{2k}$ is required.

We do not follow the data representation method of pMPL over a ring because it is difficult to manually find a public matrix that satisfies four restrictions outlined in Section II-B, especially when the number of parties increases and the composition is more complex. Specifically, as stated in the public matrix constraints in Section II-B, if we perform all the computations over the ring $\mathbb{Z}_{2^\ell}$, both the values of the public matrix and the reconstruction coefficients should be elements on the ring. Besides, the reconstruction coefficients are non-zero values. When the number of parties n and the number of parties allowed to drop out nd increase, the number of reconstruction coefficients $c_{0,i}, c_{m,i}$ for $i \in \{1, \ldots, n\}, m \in \{1, \ldots, nd\}$ will increase. As a result, ensuring that each reconstruction coefficient is a non-zero element over the ring becomes extremely difficult. If a reconstruction coefficient is a decimal, it will lead to incorrect reconstruction results. Therefore, all our calculations are performed on the prime field so as to easily find a public matrix and ensure the correctness of the training process.

C. The Offline/Online Paradigm

In Ruyi, we follow the offline/online paradigm used in the prior works [4]–[6]. In the offline/online paradigm, the computation is divided into two phases: an offline phase and an online phase. The offline phase is independent of the parties' inputs and hence can be run in advance to generate some random values used in the online phase, while the online phase is dependent on the parties' input and executes the actual computations. The online phase is typically much more lightweight than the offline phase.

D. Security Model

In Ruyi, we consider a semi-honest security model, also known as an honest-but-curious or passive security model, widely used in MPL frameworks [4]–[6]. In this model, the semi-honest adversary attempts to infer additional information from the messages they received during the training but follows the protocol specification. Furthermore, unlike the honest majority assumption in most MPL frameworks [4]–[6], where less than half of the parties can collude, all the na ($na \geq n/2$) assistant parties in Ruyi can collude with any $np - 1$ privileged parties, all the np privileged parties can collude with any $na - nd - 1$ assistant parties.

IV. Design of Ruyi

A. Sharing Semantics

- $[\cdot]$-sharing: A value $x \in \mathbb{F}_q$ is said to be $[\cdot]$-shared (vector space secret shared) among parties in $\mathcal{P}$, if each party P_i

for $i \in \{1, \dots, n\}$ holds $[x]_i \in \mathbb{F}_q$ and all privileged parties hold $[x]_{n+1}, \dots, [x]_{n+nd} \in \mathbb{F}_q$ additionally, such that $x = \sum_{i=1}^{n} c_{0,i} \cdot [x]_i \mod q = \sum_{i=1, i \neq d_1, \dots, d_m}^{n+m} c_{m,i} \cdot [x]_i \mod q$ for $m \in \{1, \dots, nd\}$. Here $c_{0,i}$ and $c_{m,i}$ are reconstruction coefficients (public constants) used to reconstruct the value x, and we represent the above equation as $x = \sum_{i=1}^{n} c_{0,i} \cdot [x]_i = \sum_{i=1, i \neq d_1, \dots, d_m}^{n+m} c_{m,i} \cdot [x]_i$ for $m \in \{1, \dots, nd\}$ in the rest of the paper.

- $[\![\cdot]\!]$-sharing: A value $x \in \mathbb{F}_q$ is said to be $[\![\cdot]\!]$-shared among parties in $\mathcal{P}$, if there exist values $\gamma_x, \eta_x \in \mathbb{F}_q$ such that (i) η_x is a random value and $[\cdot]$-shared among parties in $\mathcal{P}$; (ii) $\gamma_x = x + \eta_x$ is known to all parties in $\mathcal{P}$ in clear. Therefore, each privileged party P_j for $j \in \{1, \dots, np\}$ holds $[\![x]\!]_j = ([\eta_x]_j, [\eta_x]_{n+1}, \dots, [\eta_x]_{n+nd}, \gamma_x)$ and each assistant party P_j for $j \in \{np+1, \dots, n\}$ holds $[\![x]\!]_j = ([\eta_x]_j, \gamma_x)$ such that $x = \gamma_x - \sum_{i=1}^{n} c_{0,i} \cdot [\eta_x]_i \mod q = \gamma_x - \sum_{i=1, i \neq d_1, \dots, d_m}^{n+m} c_{m,i} \cdot [\eta_x]_i \mod q$ for $m \in \{1, \dots, nd\}$ (see more details in Section IV-C). We represent the above equation as $x = \gamma_x - \sum_{i=1}^{n} c_{0,i} \cdot [\eta_x]_i = \gamma_x - \sum_{i=1, i \neq d_1, \dots, d_m}^{n+m} c_{m,i} \cdot [\eta_x]_i$ for $m \in \{1, \dots, nd\}$ in the rest of the paper.
- $\langle \cdot \rangle$-sharing: A value $x \in \mathbb{F}_q$ is said to be $\langle \cdot \rangle$-shared (additive secret shared) among parties in $\mathcal{P}$, if each party P_i holds $\langle x \rangle_i \in \mathbb{F}_q$ for $i \in \{1, 2, \dots, n\}$ such that $x = \sum_{i=1}^{n} \langle x \rangle_i \mod q$, which is represented as $x = \sum_{i=1}^{n} \langle x \rangle_i$ in the rest of the paper.

Linearity of the Secret Sharing Schemes: Given the $[\cdot]$-shares of x and y, along with public constants t_1, t_2, $[\cdot]$-sharing satisfies a linearity property that parties in $\mathcal{P}$ can locally compute $[z] = [t_1 \cdot x + t_2 \cdot y] = t_1 \cdot [x] + t_2 \cdot [y]$. Similarly, parties can locally compute $[\eta_z] = t_1 \cdot [\eta_x] + t_2 \cdot [\eta_y]$. As $\gamma_z = t_1 \cdot \gamma_x + t_2 \cdot \gamma_y$ can be computed locally, $[\![\cdot]\!]$-sharing also satisfies the linearity property that parties in $\mathcal{P}$ can locally compute $[\![z]\!] = t_1 \cdot [\![x]\!] + t_2 \cdot [\![y]\!]$. Furthermore, the linearity property can be trivially extended to $\langle \cdot \rangle$-sharing. The linearity property enables parties in $\mathcal{P}$ to execute addition operations non-interactively, as well as execute multiplication operations of their shares with a public constant.

B. Public Matrix

To provide the privileged parties a privileged position that guarantees: (i) only np privileged parties can obtain the final trained model, even if all na assistant parties collude with each other; and (ii) the training process can continue to the end even if nd assistant parties drop out, we define the public matrix $\Phi(\mathcal{P})$ as an $(n+nd) \times n$ matrix. Here, for each party P_i ($i \in \{1, \dots, n\}$), the *i-th* row $\Phi(i)$ of $\Phi(\mathcal{P})$ is its corresponding public vector. Besides, $\Phi(n+1), \dots, \Phi(n+nd)$ are nd alternate public vectors corresponding to all privileged parties $P_1, \dots, P_{np}$.

The public matrix $\Phi(\mathcal{P})$ should satisfy two constraints as follows:

1) The vector $(1, 0, \dots, 0)$ can be represented as a linear combination of the public vectors in the set $\{\Phi(1), \dots, \Phi(n)\}$, where these public vectors are linearly independent. In other words, there exist n unique non-zero reconstruction coefficients (public constants) $c_{0,1}, \dots, c_{0,n}$ such that $(1, 0, \dots, 0) = \sum_{i=1}^{n} c_{0,i} \cdot \Phi(i)$.
2) The vector $(1, 0, \dots, 0)$ can be represented as a linear combination of n public vectors corresponding to all np privileged parties, $na - m$ assistant parties, and m alternate public vectors for $m \in \{1, \dots, nd\}$. Besides, these public vectors are linearly independent. In other words, there exist nd groups of unique non-zero reconstruction coefficients (public constants) $c_{m,i}$ for $i \in \{1, \dots, n+m\}, i \neq d_1, \dots, d_m$ and $m \in \{1, \dots, nd\}$, where each group contains n reconstruction coefficients. Therefore $(1, 0, \dots, 0) = \sum_{i=1, i \neq d_1, \dots, d_m}^{n+m} c_{m,i} \cdot \Phi(i)$ for $m \in \{1, \dots, nd\}$.

The first constraint is to ensure that n parties could reconstruct the secret value according to Equation 3. Besides, it ensures that any $n' < n$ parties cannot reconstruct the secret value for the reason that reconstruction coefficients are unique non-zero public constants. That is, even if na assistant parties collude with each other, it cannot reconstruct the secret value. The second constraint is to ensure np privileged parties (who hold np shares and nd alternate shares) and $na - nd$ assistant parties can reconstruct the secret value. That is, if nd assistant parties drop out, the training could continue to the end. Therefore, the above two constraints can ensure two guarantees.

An $(n+nd) \times n$ Vandermonde matrix $\mathbf{VM}$ satisfies the above two constraints and the proof is as follows.

$$\mathbf{VM} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & 2 & \cdots & 2^{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & n+nd & \cdots & (n+nd)^{n-1} \end{bmatrix} = \Phi(\mathcal{P})$$

Proof. We define the top n rows $\{\Phi(1), \dots, \Phi(n)\}$ of $\mathbf{VM}$ as $\mathbf{VM}_0$, i.e. $\mathbf{VM}_0 = \{\Phi(1), \dots, \Phi(n)\}$, and $c_{0,1}, \dots, c_{0,n}$ as C_0. Similarly, we define $\mathbf{VM}_m = \{\Phi(i) | 1 \leq i \leq n+m, i \neq d_1, \dots, d_m\}$ for $m \in \{1, \dots, nd\}$, and the corresponding reconstruction coefficients $C_m = \{c_{m,i} | 1 \leq i \leq n+m, i \neq d_1, \dots, d_m\}$.

For public matrix constraint 1), according to Theorem 1, $\mathbf{VM}_0$ is invertible, therefore these vectors $\{\Phi(1), \dots, \Phi(n)\}$ of $\mathbf{VM}_0$ are linearly independent. Besides $(1, 0, \dots, 0) = \sum_{i=1}^{n} c_{0,i} \cdot \Phi(i)$ can be represented as $(1, 0, \dots, 0) = C_0 \cdot \mathbf{VM}_0$. Thus $C_0 = (1, 0, \dots, 0) \cdot \mathbf{VM}_0^{-1}$, where $\mathbf{VM}_0^{-1}$ is the inverse of $\mathbf{VM}_0$. In other words, C_0 is the first row of $\mathbf{VM}_0^{-1}$. According to Theorem 1, due to the fact that each element in $\mathbf{VM}_0$ is non-zero, each element in $\mathbf{VM}_0^{-1}$ is non-zero and $C_0 = \{c_{0,1}, \dots, c_{0,n}\} = \{n, \frac{n(1-n)}{2}, \dots, (-1)^{n-1}\}$. Therefore, the Vandermonde matrix $\mathbf{VM}$ satisfies the public matrix constraint 1).

For public matrix constraint 2), according to Theorem 1, each $\mathbf{VM}_m = \{\Phi(i) | 1 \leq i \leq n+m, i \neq d_1, \dots, d_m\}$ for $m \in \{1, \dots, nd\}$ is also invertible. Besides, each element in $\mathbf{VM}_m$ is also non-zero. Thus, the first row of $\mathbf{VM}_m^{-1}$, i.e. $c_{m,i}$ for $i \in \{1, \dots, n+m\}, i \neq d_1, \dots, d_m$ and $m \in \{1, \dots, nd\}$ are all non-zero. Therefore, the Vandermonde matrix $\mathbf{VM}$ satisfies the public matrix constraint 2). $\square$

Theorem 1. *For an $n \times n$ square Vandermonde matrix:*

$$\mathbf{V} = \begin{bmatrix} 1 & \beta_1 & \cdots & \beta_1^{n-1} \\ 1 & \beta_2 & \cdots & \beta_2^{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \beta_n & \cdots & \beta_n^{n-1}, \end{bmatrix}$$

if $\beta_1, \dots, \beta_n$ are distinct, then $\mathbf{V}$ is invertible. In the inverse of this Vandermonde matrix, each element $\mathbf{V}^{-1}(i,j)$ for $i,j = \{1,\dots,n\}$ of is:

$$\mathbf{V}^{-1}(i,j) = (-1)^{n-i} \frac{\sigma_{n-i,j}}{\prod_{k=1,k\neq j}^{n}(\beta_j - \beta_k)},$$

where

$$\begin{aligned}\sigma_{n-i,j} &= \sigma_{n-i}(\beta_1,\dots,\beta_{j-1},\beta_{j+1},\dots,\beta_n) \\ &= \sum_{\substack{1\leq k_1<\cdots<k_{n-i}\leq n \\ k_1,\dots,k_{n-i}\neq j}} (\beta_{k_1}\beta_{k_2}\cdots\beta_{k_{n-i}}),\end{aligned}$$

and $\sigma_{0,j} = 1$.

Proof. Theorem 1 has been proven in [22]. Therefore, we do not provide proof here. □

Compared with pMPL, we reduce the constraints from four to two while providing the same privileged position. Firstly, constraint 2) in pMPL requires the alternate public vector $\Phi(4)$ to be a linear combination of $\Phi(2)$ and $\Phi(3)$ in order to compute the alternate share $[x]_4$ in share conversion A2V protocol. In Ruyi, we extend the standard resharing paradigm to vector space secret sharing so as to implement the share conversion protocol $\prod_{\text{convert}}(\mathcal{P}, \langle x \rangle)$ (Protocol 5) that eliminates this requirement. Secondly, as all reconstruction coefficients are unique non-zero constants, the linear combination of the public vectors corresponding to all assistant parties or all privileged parties cannot be represented as $(1,0,\dots,0)$. Therefore, constraint 3) in pMPL is already included in constraints 1) and 2). Finally, all computations in pMPL are performed over a ring whereas in Ruyi, they are performed over a field. Therefore, we eliminate the constraint 4) in pMPL.

C. Sharing and Reconstruction Protocols

We prove the security of sharing and reconstruction protocols in Appendix A using the standard real/ideal world paradigm [6], [8].

Sharing Protocol. As shown in Protocol 1, the sharing protocol $\prod_{\text{shr}}(P_\alpha, x)$ enables P_α ($\alpha \in \{1,\dots,n\}$), who holds the secret value x, to generate its $[\![\cdot]\!]$-shares to parties in $\mathcal{P}$. During the offline phase, P_α samples a random value η_x and generates its $[\cdot]$-shares for each party. Specifically, P_α first constructs the random value η_x as a random n-dimensional column vector $\vec{s} = (\eta_x, s_1, \dots, s_{n-1})$, where $s_1,\dots,s_{n-1}$ are all random values. Then P_α computes $[\eta_x]_i = \Phi(i) \times \vec{s}$ and sends it to P_i for $i \in \{1,\dots,n\}$. Additionally, P_α computes the alternate shares $[\eta_x]_{n+m} = \Phi(n+m) \times \vec{s}$ for $m \in \{1,\dots,nd\}$ and sends it to all privileged parties. During the online phase, P_α computes $\gamma_x = x + \eta_x$ and sends it to the other parties. Therefore each privileged party P_j for $j \in \{1,\dots,np\}$ obtains its share $[\![x]\!]_j = ([\eta_x]_j, [\eta_x]_{n+1}, \dots, [\eta_x]_{n+nd}, \gamma_x)$, while each assistant party P_j for $j \in \{np+1\dots,n\}$ obtains its share $[\![x]\!]_j = ([\eta_x]_j, \gamma_x)$.

Protocol 1: $\prod_{\text{shr}}(P_\alpha, x)$

Input: The secret value x held by P_α.
Output: $[\![\cdot]\!]$-shares of secret value x.
Offline:

1: P_α constructs a random n-dimensional column vector $\vec{s} = (\eta_x, s_1, \dots, s_{n-1})$, where $\eta_x, s_1, \dots, s_{n-1}$ are all random values.
2: P_α computes $[\eta_x]_i = \Phi(i) \times \vec{s}$ and sends it to P_i for $i \in \{1,\dots,n\}$. P_α computes alternate shares $[\eta_x]_{n+m} = \Phi(n+m) \times \vec{s}$ for $m \in \{1,\dots,nd\}$ and sends them to all privileged parties.

Online:

1: P_α computes $\gamma_x = x + \eta_x$ and sends it to the other parties.
2: Each privileged party P_j for $j \in \{1,\dots,np\}$ obtains $[\![x]\!]_j = ([\eta_x]_j, [\eta_x]_{n+1}, \dots, [\eta_x]_{n+nd}, \gamma_x)$, while each assistant party P_j for $j \in \{np+1,\dots,n\}$ obtains $[\![x]\!]_j = ([\eta_x]_j, \gamma_x)$.

Reconstruction Protocol. According to constraints (1) and (2) outlined in Section IV-B, we obtain the Equation (4).

$$\begin{aligned}(1,0,\dots,0) &= \sum_{i=1}^{n} c_{0,i} \cdot \Phi(i) \\ &= \sum_{i=1,i\neq d_1,\dots,d_m}^{n+m} c_{m,i} \cdot \Phi(i)(m = \{1,\dots,nd\})\end{aligned} \tag{4}$$

Based on the Equation (4) and sharing protocol $\prod_{\text{shr}}(P_i, x)$ (Protocol 1), parties in $\mathcal{P}$ can reconstruct η_x when given shares $[\eta_x]$ using Equation (5).

$$\begin{aligned}\eta_x &= \sum_{i=1}^{n} c_{0,i} \cdot [\eta_x]_i \\ &= \sum_{i=1,i\neq d_1,\dots,d_m}^{n+m} c_{m,i} \cdot [\eta_x]_i(m = \{1,\dots,nd\})\end{aligned} \tag{5}$$

As is shown in Protocol 2, the reconstruction protocol $\prod_{\text{rec}}(\mathcal{P}, [\![x]\!])$ enables parties in $\mathcal{P}$ to reconstruct the secret value x given its $[\![\cdot]\!]$-shares. We consider two cases as follows: (i) If no assistant party drops out, each party P_i for $i \in \{1,\dots,n\}$ sends its $[\cdot]$-share of η_x to the other parties. Then each party P_i can reconstruct the secret value x by locally computing $x = \gamma_x - \sum_{i=1}^{n} c_{0,i} \cdot [\eta_x]_i$; (ii) If m ($1 \leq m \leq nd$) assistant parties $P_{d_1}, \dots, P_{d_m}$ drop out, each party P_i for $i \in \{1,\dots,n\}, i \neq d_1,\dots,d_m$ sends its $[\cdot]$-shares of η_x to the other non-dropped-out parties. In addition, one of the privileged parties sends the alternate shares $[\eta_x]_{n+1},\dots,[\eta_x]_{n+m}$ to all assistant parties. Without loss of generality, we define this privileged party as P_1. Then each party P_i for $i \in \{1,\dots,n\}, i \neq d_1, \dots, d_m$ can reconstruct the secret value x by locally computing $x = \gamma_x - \sum_{i=1,i\neq d_1,\dots,d_m}^{n+m} c_{m,i} \cdot [\eta_x]_i$.

Note that once the training is completed, all assistant parties send their final model shares to all privileged parties, while the privileged parties do not send their final model shares to the assistant parties. This ensures that only the privileged parties can obtain the final model.

Protocol 2: $\prod_{\text{rec}}(\mathcal{P}, [\![x]\!])$

Input: $[\![\cdot]\!]$-shares of secret value x.
Output: The secret value x.
Online:
- If no assistant party drops out:
```
1: Each party $P_i$ for $i \in \{1, \dots, n\}$ sends its $[\cdot]$-share of $\eta_x$ to the other parties.
2: Each party $P_i$ for $i \in \{1, \dots, n\}$ locally computes $x = \gamma_x - \sum_{i=1}^{n} c_{0,i} \cdot [\eta_x]_i$.
```
- If m ($1 \leq m \leq nd$) assistant parties $P_{d_1}, \dots, P_{d_m}$ drop out:
```
1: Each party $P_i$ for $i \in \{1, \dots, n\}, i \neq d_1, \dots, d_m$ sends its $[\cdot]$-share of $\eta_x$ to the other parties. $P_1$ sends $[\eta_x]_{n+1}, \dots, [\eta_x]_{n+m}$ to all assistant parties additionally.
2: Each party $P_i$ for $i \in \{1, \dots, n\}, i \neq d_1, \dots, d_m$ locally computes $x = \gamma_x - \sum_{i=1, i \neq d_1, \dots, d_m}^{n+m} c_{m,i} \cdot [\eta_x]_i$.
```

D. Basic Primitives

We prove the security of the following basic primitives in Appendix A using the standard real/ideal world paradigm.

Compared to pMPL, the first core technical contribution is that we design multiple primitives based on the adaptation of the masked evaluation paradigm to vector space secret sharing rather than the raw vector space secret sharing, thereby improving the online communication efficiency. The second one is that we utilize a standard resharing paradigm to achieve the share conversion protocol that eliminates the reliance on the relationship between public vectors, thereby reducing the constraints of our public matrix. Moreover, all our protocols are designed over a prime field, while the protocols in pMPL are designed over an integer ring.

Linear Operations. $[\![z]\!] \leftarrow t_1 \cdot [\![x]\!] + t_2 \cdot [\![y]\!] + t_3$. Given two $[\![\cdot]\!]$-shares of x and y held by each party P_i for $i \in \{1, \dots, n\}$ and public constant value t_1, t_2, t_3, parties in $\mathcal{P}$ can locally compute $[\![z]\!] = t_1 \cdot [\![x]\!] + t_2 \cdot [\![y]\!] + t_3$. Specifically, each party P_i for $i \in \{1, \dots, n\}$ locally computes $\gamma_z = t_1 \cdot \gamma_x + t_2 \cdot \gamma_y + t_3$ and $[\eta_z]_i = t_1 \cdot [\eta_x]_i + t_2 \cdot [\eta_y]_i$. Additionally, the privileged parties locally compute alternate shares $[\eta_z]_{n+m} = t_1 \cdot [\eta_x]_{n+m} + t_2 \cdot [\eta_y]_{n+m}$ for $m \in \{1, \dots, nd\}$.

Secure Multiplication. $[\![z]\!] \leftarrow [\![x \cdot y]\!]$. Given two $[\![\cdot]\!]$-shares of x and y held by each party P_i for $i \in \{1, \dots, n\}$, the secure multiplication protocol $\prod_{\text{mul}}(\mathcal{P}, [\![x]\!], [\![y]\!])$ (Protocol 3) enables parties in $\mathcal{P}$ to generate $[\![\cdot]\!]$-shares of z, where $z = x \cdot y$. For correctness of z, we have:

$$\begin{aligned}\gamma_z &= z + \eta_z = x \cdot y + \eta_z \\ &= (\gamma_x - \eta_x) \cdot (\gamma_y - \eta_y) + \eta_z \\ &= \gamma_x \cdot \gamma_y - \gamma_x \cdot \eta_y - \gamma_y \cdot \eta_x + \eta_x \cdot \eta_y + \eta_z\end{aligned} \tag{6}$$

For the above Equation (6), each party P_i for $i \in \{1, \dots, n\}$ holds $\gamma_x, \gamma_y, [\eta_x]_i$ and $[\eta_y]_i$. Besides, each privileged party P_j for $j \in \{1, \dots, np\}$ holds ($[\eta_x]_{n+1}, \dots, [\eta_x]_{n+nd}$) and ($[\eta_y]_{n+1}, \dots, [\eta_y]_{n+nd}$) additionally. Therefore, in order to compute γ_z, the key is to obtain the $[\cdot]$-shares of $\eta_x \cdot \eta_y$ and η_z.

As Protocol 3 shown, in the offline phase, parties in $\mathcal{P}$ generate $[\cdot]$-shares of $\eta_x \cdot \eta_y$ using the randomized multiplication protocol $\prod_{\text{rand\_mul}}(\mathcal{P}, [\eta_x], [\eta_y])$ (Protocol 4) and $[\cdot]$-shares of η_z using the random value generation protocol $\prod_{\text{rand\_gen}}(\mathcal{P})$ (Protocol 6). Their details are provided later in this subsection. In the online phase, each party P_i for $i \in \{1, \dots, n\}$ locally computes $[\gamma_z{}']_i = -\gamma_x \cdot [\eta_y]_i - \gamma_y \cdot [\eta_x]_i + [\eta_x \cdot \eta_y]_i + [\eta_z]_i$. Then parties in $\mathcal{P}$ exchange the $[\cdot]$-shares of $\gamma_z{}'$ to reconstruct it. Finally, each party P_i for $i \in \{1, \dots, n\}$ locally computes $\gamma_z = \gamma_x \cdot \gamma_y + \gamma_z{}'$. Therefore, each privileged party P_j for $j \in \{1, \dots, np\}$ obtains its share $[\![z]\!]_j = ([\eta_z]_j$, $[\eta_z]_{n+1}, \dots, [\eta_z]_{n+nd}, \gamma_z)$, while each assistant party P_j for $j \in \{np+1, \dots, n\}$ obtains its share $[\![z]\!]_j = ([\eta_z]_j, \gamma_z)$.

Protocol 3: $\prod_{\text{mul}}(\mathcal{P}, [\![x]\!], [\![y]\!])$

Input: $[\![\cdot]\!]$-shares of x and y held by each party.
Output: $[\![\cdot]\!]$-shares of z, where $z = x \cdot y$.
Offline:
```
1: Parties in $\mathcal{P}$ generate $[\cdot]$-shares of $\eta_x \cdot \eta_y$ using the randomized multiplication protocol $\prod_{\text{rand\_mul}}$ ($\mathcal{P}, [\eta_x], [\eta_y]$) (Protocol 4).
2: Parties in $\mathcal{P}$ generate $[\cdot]$-shares of random value $\eta_z$ using the random value generation protocol $\prod_{\text{rand\_gen}}$ ($\mathcal{P}$) (Protocol 6).
```
Online:
```
1: Each party $P_i$ for $i \in \{1, \dots, n\}$ locally computes $[\gamma_z{}']_i = -\gamma_x \cdot [\eta_y]_i - \gamma_y \cdot [\eta_x]_i + [\eta_x \cdot \eta_y]_i + [\eta_z]_i$.
2: Parties in $\mathcal{P}$ exchange the shares $[\cdot]$-shares of $\gamma_z{}'$ to reconstruct it.
3: Each party $P_i$ for $i \in \{1, \dots, n\}$ locally computes $\gamma_z = \gamma_x \cdot \gamma_y + \gamma_z{}'$.
4: Each privileged party $P_j$ for $j \in \{1, \dots, np\}$ obtains $[\![z]\!]_j = ([\eta_z]_j, [\eta_z]_{n+1}, \dots, [\eta_z]_{n+nd}, \gamma_z)$, while each assistant party $P_j$ for $j \in \{np+1, \dots, n\}$ obtains $[\![z]\!]_j = ([\eta_z]_j, \gamma_z)$.
```

Protocol 4: $\prod_{\text{rand\_mul}}(\mathcal{P}, [x], [y])$

Input: $[\cdot]$-shares of x and y held by each party.
Output: $[\cdot]$-shares of z, where $z = x \cdot y$.
```
1: Each party $P_i$ for $i \in \{1, \dots, n\}$ locally computes $(c_{0,i})^2 \cdot [x]_i \cdot [y]_i$.
2: Each party pair $(P_i, P_j)$ for $i, j \in \{1, \dots, n\}, i \neq j$ generate the $\langle\cdot\rangle$-shares of $[x]_i \cdot [y]_j + [x]_j \cdot [y]_i$ using the methods in [4].
3: Each party $P_i$ for $i \in \{1, \dots, n\}$ locally computes $\langle z \rangle_i = (c_{0,i})^2 \cdot [x]_i \cdot [y]_i + c_{0,i} \cdot c_{0,j} \cdot \langle [x]_i \cdot [y]_j + [x]_j \cdot [y]_i \rangle_i$ $(j \in \{1, \dots, n\}, i \neq j)$.
4: Parties in $\mathcal{P}$ execute the share conversion protocol $\prod_{\text{convert}}(\langle z \rangle)$ (Protocol 5) to covert the $\langle\cdot\rangle$-shares of $z$ to its equivalent $[\cdot]$-shares.
```

We now provide the details of the randomized multiplication protocol $\prod_{\text{rand\_mul}}(\mathcal{P}, [x], [y])$ (Protocol 4). Given two $[\cdot]$-shares of x and y held by each party P_i for $i \in \{1, \dots, n\}$, the randomized multiplication protocol $\prod_{\text{rand\_mul}}(\mathcal{P}, [x], [y])$ (Protocol 4) enables parties in $\mathcal{P}$ to generate $[\cdot]$-shares of z, where $z = x \cdot y$. We split the computation into two steps, generating the $\langle\cdot\rangle$-shares of $x \cdot y$ and converting its $\langle\cdot\rangle$-shares to its equivalent $[\cdot]$-shares. For the first step, according to Equation 7, each party P_i for $i \in \{1, \dots, n\}$ can locally compute the former part $(c_{0,i})^2 \cdot [x]_i \cdot [y]_i$. Then each party pair (P_i, P_j) for $i, j \in \{1, \dots, n\}, i \neq j$ interactively generate the $\langle\cdot\rangle$-shares of

$[x]_i \cdot [y]_j + [x]_j \cdot [y]_i$ using the methods in [4] based on OT [23]. Finally, each party P_i for $i \in \{1, \ldots, n\}$ locally computes $\langle z \rangle_i = (c_{0,i})^2 \cdot [x]_i \cdot [y]_i + c_{0,i} \cdot c_{0,j} \cdot \langle [x]_i \cdot [y]_j + [x]_j \cdot [y]_i \rangle_i$ ($j \in \{1, \ldots, n\}, i \neq j$). For the second step, parties in $\mathcal{P}$ execute the share conversion protocol $\prod_{\text{convert}}(\mathcal{P}, \langle z \rangle)$ (Protocol 5) to covert the $\langle \cdot \rangle$-shares of $z = x \cdot y$ to its equivalent $[\cdot]$-shares. Firstly, each party P_α for $\alpha \in \{1, \ldots, n\}$ treats its own share $\langle z \rangle_\alpha$ as secret value and performs vector space secret sharing to generate $[\langle z \rangle_\alpha]$. After that, each party P_i for $i \in \{1, \ldots, n\}$ locally sums up all $[\cdot]$-shares they hold as $[z]_i = \sum_{\alpha=1}^{n} [\langle z \rangle_\alpha]_i$. Additionally, all privileged parties P_j for $j \in \{1, \ldots, np\}$ locally compute $[z]_{n+m} = \sum_{\alpha=1}^{n} [\langle z \rangle_\alpha]_{n+m}$ for $m \in \{1, \ldots, nd\}$.

$$\begin{aligned} z = x \cdot y &= (\sum_{i=1}^{n} c_{0,i} \cdot [x]_i) \cdot (\sum_{i=1}^{n} c_{0,i} [y]_i) \\ &= \sum_{i=1}^{n} (c_{0,i})^2 \cdot [x]_i \cdot [y]_i \\ &+ \sum_{1 \leq i,j \leq n, i \neq j} c_{0,i} \cdot c_{0,j} \cdot ([x]_i \cdot [y]_j + [x]_j \cdot [y]_i) \end{aligned} \tag{7}$$

Protocol 5: $\prod_{\text{convert}}(\mathcal{P}, \langle x \rangle)$

Input: $\langle \cdot \rangle$-shares of x held by each party.
Output: $[\cdot]$-shares of x.

```
1: Each party $P_\alpha$ for $\alpha \in \{1, \ldots, n\}$ constructs an $n$-dimensional column vector $\vec{s} = (\langle x \rangle_\alpha, s_1, \ldots, s_{n-1})$, where $s_1, \ldots, s_{n-1}$ are all random values.
2: Each party $P_\alpha$ for $\alpha \in \{1, \ldots, n\}$ computes $[\langle x \rangle_\alpha]_i = \Phi(i) \times \vec{s}$ and sends it to $P_i$ for $i \in \{1, \ldots, n\}$. Each party $P_\alpha$ for $\alpha \in \{1, \ldots, n\}$ computes alternate shares $[\langle x \rangle_\alpha]_{n+m} = \Phi(n+m) \times \vec{s}$ for $m \in \{1, \ldots, nd\}$ and sends them to all privileged parties.
3: Each party $P_i$ for $i \in \{1, \ldots, n\}$ locally computes $[x]_i = \sum_{\alpha=1}^{n} [\langle x \rangle_\alpha]_i$. All privileged parties locally compute $[x]_{n+m} = \sum_{\alpha=1}^{n} [\langle x \rangle_\alpha]_{n+m}$ for $m \in \{1, \ldots, nd\}$.
```

As for the random value generation protocol $\prod_{\text{rand\_gen}}(\mathcal{P})$ (Protocol 6), each party P_i for $i \in \{1, \ldots, n\}$ locally generates a random value r_i and treats it as the $\langle \cdot \rangle$-share of r, i.e. $r_i = \langle r \rangle_i$. Then parties in $\mathcal{P}$ execute the share conversion protocol $\prod_{\text{convert}}(\mathcal{P}, r_i)$ (Protocol 5) to get the $[\cdot]$-shares of the random value r.

Protocol 6: $\prod_{\text{rand\_gen}}(\mathcal{P})$

Input: $\emptyset$.
Output: $[\cdot]$-shares of r, where r is the random value.

```
1: Each party $P_i$ for $i \in \{1, \ldots, n\}$ locally generates a random value $r_i$.
2: Parties in $\mathcal{P}$ execute the share conversion protocol $\prod_{\text{convert}}(\mathcal{P}, r_i)$ (Protocol 5) to get the $[\cdot]$-shares of the random value $r$.
```

Secure Truncation. $[\![z^t]\!] \leftarrow [\![\lfloor z/2^f \rfloor]\!]$. As the fractional part will double in size after each multiplication, to prevent overflow during repeated multiplications, truncation is performed after each multiplication. Given the $[\![\cdot]\!]$-share of z held by each party P_i for $i \in \{1, \ldots, n\}$, the secure truncation protocol $\prod_{\text{trunc}}(\mathcal{P}, [\![z]\!])$ (Protocol 7) enables parties in $\mathcal{P}$ to generate $[\![\cdot]\!]$-shares of z^t, where $z^t = \lfloor z/2^f \rfloor$.

In the offline phase, parties in $\mathcal{P}$ generate $[\cdot]$-shares of η_z^t where $\eta_z^t = \lfloor \eta_z/2^f \rfloor + u$ and $u = (\eta_z/2^f - \lfloor \eta_z/2^f \rfloor \geq 0.5)?1 : 0$. Firstly, each party P_i takes $[\eta_z]_i$ for $i \in \{1, \ldots, n\}$ as input and executes the `TruncPR` protocol in [24] to generate $[\eta_z^t]_i$. To generate the alternate shares $[\eta_z^t]_{n+1}, \ldots, [\eta_z^t]_{n+nd}$, each party P_i locally computes $\langle \eta_z^t \rangle_i = c_{0,i} \cdot [\eta_z^t]_i$ for $i \in \{1, \ldots, n\}$ to convert $[\cdot]$-share of η_z^t to its equivalent $\langle \cdot \rangle$-share. Then, parties in $\mathcal{P}$ execute the share conversion protocol $\prod_{\text{convert}}(\mathcal{P}, \langle \eta_z^t \rangle_i)$ (Protocol 5) to convert $\langle \cdot \rangle$-shares of η_z^t to its equivalent $[\cdot]$-shares, including alternate shares. The correctness proof of $[\eta_z^t]$ is shown in Appendix C.

In the online phase, each privileged party P_j for $j \in \{1, \ldots, np\}$ locally sets $[\![z^t]\!]_j = ([\eta_z^t]_j, [\eta_z^t]_{n+1}, \ldots, [\eta_z^t]_{n+nd}, \gamma_z^t)$, while each assistant party P_j for $j \in \{np+1, \ldots, n\}$ locally sets $[\![z^t]\!]_j = ([\eta_z^t]_j, \gamma_z^t)$, where $\gamma_z^t = \lfloor \gamma_z/2^f \rfloor$. Therefore, the secure truncation protocol designed in `Ruyi` can be executed locally.

Note that Li *et al.* [25] pointed out that for a fixed input $[z]$ in `TrunPR` protocol, whether the output z^t is rounding up or down is solely determined by r (the randomness of mask z). This means z^t is fixed if r is fixed. This violates an ideal functionality for probabilistic truncation. Therefore, it does not satisfy standard simulation-based security. However, the `TrunPR` protocol can satisfy the standard simulation-based security by simply modifying the probabilistic truncation ideal functionality, which is shown in [26].

Protocol 7: $\prod_{\text{trunc}}(\mathcal{P}, [\![z]\!])$

Input: $[\![\cdot]\!]$-share of z held by each party.
Output: $[\![\cdot]\!]$-shares of z^t, where $z^t = \lfloor z/2^f \rfloor$.
Offline:

```
1: Parties in $\mathcal{P}$ executes the TruncPR protocol in [24] to generate $[\eta_z^t]_i$ for $i \in \{1, \ldots, n\}$.
2: Each party $P_i$ locally computes $\langle \eta_z^t \rangle = c_{0,i} \cdot [\eta_z^t]_i$.
3: Parties in $\mathcal{P}$ execute the share conversion protocol $\prod_{\text{convert}}(\mathcal{P}, \langle \eta_z^t \rangle_i)$ (Protocol 5) to convert $\langle \cdot \rangle$-shares of $\eta_z^t$ to its equivalent $[\cdot]$-shares.
```

Online:

```
1: Each privileged party $P_j$ for $j \in \{1, \ldots, np\}$ locally sets the truncated share $[\![z^t]\!]_j = ([\eta_z^t]_j, [\eta_z^t]_{n+1}, \ldots, [\eta_z^t]_{n+nd}, \lfloor \gamma_z/2^f \rfloor)$, while each assistant party $P_j$ for $j \in \{np+1, \ldots, n\}$ locally sets the truncated share $[\![z^t]\!]_j = ([\eta_z^t]_j, \lfloor \gamma_z/2^f \rfloor)$.
```

Secure Comparison. $[\![z]\!] \leftarrow [\![x < 0?1 : 0]\!]$. Given the $[\![\cdot]\!]$-share of x held by each party P_i for $i \in \{1, \ldots, n\}$, the secure comparison protocol $\prod_{\text{compar}}(\mathcal{P}, [\![x]\!])$ enables parties in $\mathcal{P}$ to generate $[\![\cdot]\!]$-shares of z, where $z = x < 0?1 : 0$. Here, we use the `LTZ` protocol in [24], [27], but the computations are replaced by our designed basic primitives based on $[\![\cdot]\!]$-sharing semantic.

E. Design for Part Parties Dropping Out

`Ruyi` ensures that if any m assistant parties $P_{d_1}, \ldots, P_{d_m}$ drop out, where $np+1 \leq d_1, \ldots, d_m \leq n$ and $1 \leq m \leq nd$, the training will continue to guarantee the privileged parties can get the final result. We achieve this by generating nd alternate shares to privileged parties. Additionally, during the training process (online phase), these alternate shares perform the same computations as the other shares. Therefore, even if there are $m(1 \leq m \leq nd)$ assistant parties dropping out, the shares held by the remaining parties, along with the alternate shares held by the privileged parties, can continue with the subsequent computations.

We theoretically analyze the solutions for parties dropping out during the training process (online phase). Firstly, each party saves the current results after executing the current step. Additionally, a party P_i determines whether P_j $(i \neq j)$ drops out by checking if it receives information from P_j within the timeout threshold (a parameter set by all parties before training) during interactive steps (e.g. Step 2 in Protocol 3). If any privileged party drops out, the training process terminates. If d assistant parties drop out and $d > m'$, where m' is the current number of assistant parties allowed to drop out and m' is initialized to nd, the training process terminates. Otherwise, the remaining parties retrieve the saved results from the previous step and re-execute this step. We then update the current number of assistant parties allowed to drop out as $m' = m' - d$. Even if some parties drop out during a non-interactive step (e.g. Step 3 in Protocol 3), i.e. local computation step, the training will continue until the next interactive step. This is reasonable, as in the non-interactive steps, the dropping out of part parties will not affect the normal computations of other parties. In addition, the alternate shares perform the same computations as the other shares during the training process. This ensures that in an interactive step, if it is determined that part parties have dropped out, the remaining parties can utilize the results they retrieve from the previous step, including alternate shares, to correctly re-execute this interactive step.

V. Evaluation

A. Implementation and Experiment Settings

Implementation. We implement `Ruyi` with C++ language over the prime field $\mathbb{F}_q$. Here, we set $q = 10^{17} + 3$, which can be represented within 64-bit. Therefore, variables in `Ruyi` are represented as 64-bit integers in fixed-point form, and we use 128-bit integers to store the intermediate computations in case of overflows. At the same time, we set the bit-width of the signed integer $\bar{x}$ as $k = 20$ and the bit-width of the fractional part of $\bar{x}$ as $f = 16$.

Experiment Environment. We conduct our experiments on one Linux server equipped with a 32-core 2.4 GHz Intel Xeon CPU and 512GB of RAM. Besides, we use one separate process to simulate one computing party and use nd additional processes to simulate the computations performed on alternate shares. Meanwhile, we use OpenMP for parallel computing with 8 threads. The following experiments are carried out upon two network environments: one is the LAN setting with a bandwidth of 1GBps and sub-millisecond RTT (round-trip time) latency, and the other one is the WAN setting with 75Mbps bandwidth and 40ms RTT latency. We apply tc tool [3] to simulate the network environments to ensure that all the experiments are carried out in a reliable environment with bounded latency, thus avoiding accidental errors. For fair comparisons, we re-run `pMPL` in the above settings.

Datasets: We use the MNIST dataset [28] to evaluate the secure training of machine learning models of `Ruyi`. It contains image samples of handwritten digits from "0" to "9". The training set contains 60000 labeled pictures in 10 classes, and the testing set contains 10000 labeled pictures. Each data point consists of 784 features representing 28×28 pixels in the image. Each feature represents a pixel with a gray scale between 0∼255. We denote D as the dimension of the feature and B as the batch size, which is set to 128 by default if not specified. For the machine learning models, we consider binary classification, where the digits "0" as a class and the digits "$1 \sim 9$" as another one.

B. Secure Training of Machine Learning Models

We evaluate the online efficiency of `Ruyi` for secure training of machine learning models, including linear regression, logistic regression, and neural networks. To compare with `pMPL`, we consider the configuration with one privileged party, two assistant parties, and one assistant party allowed to drop out, which is the same as `pMPL`. At the same time, we define no assistant party dropout as 3PC while one assistant party dropout as 3PC$_{d1}$. Besides, we select the dimension of the feature $D \in \{100, 500, 1000\}$ and the batch size $B \in \{128, 784, 512, 1024\}$.

Linear Regression. We use mini-batch stochastic gradient descent (SGD for short) to train a linear regression model. The update function can be expressed as:

$$\vec{w} := \vec{w} - \frac{\alpha}{B}\mathbf{X}_i^T \times (\mathbf{X}_i \times \vec{w} - \mathbf{Y_i})$$

where $\mathbf{X}_i$ is a subset of batch size B. Besides, $(\mathbf{X}_i, \mathbf{Y}_i)$ are randomly selected from the whole dataset in the i-th iteration.

Table III shows the experimental results of `Ruyi` for linear regression training compared to `pMPL`. `Ruyi` significantly improves online throughput, particularly in the WAN setting. Specifically, in the WAN setting, we improve the online throughput by up to 53.87× and 47.86× for 3PC and 3PC$_{d1}$ respectively. Meanwhile, in the LAN setting, we improve the online throughput by up to 3.22× and 3.07× for 3PC and 3PC$_{d1}$ respectively. These improvements primarily stem from the communication overhead optimizations of matrix multiplication with truncation, which is the efficiency bottleneck in linear regression training in `Ruyi`. More details on the communication overhead optimizations achieved by `Ruyi` are provided in Table I.

Logistic Regression. Similar to linear regression, the update function using the mini-batch SGD method in logistic regression can be expressed as:

$$\vec{w} := \vec{w} - \frac{\alpha}{B}\mathbf{X}_i^T \times (\texttt{Sigmoid}(\mathbf{X}_i \times \vec{w}) - \mathbf{Y_i})$$

Table IV shows the experimental results of `Ruyi` for logistic regression training compared to `pMPL`. `Ruyi` significantly

[3]https://man7.org/linux/man-pages/man8/tc.8.html

TABLE III
ONLINE THROUGHPUT OF LINEAR REGRESSION COMPARED TO `pMPL` (3PC) AND `pMPL` (3PC$_{d1}$) (ITERATIONS/SECOND).

Setting	D	Framework	B			
			128	256	512	1024
LAN	100	`Ruyi` (3PC)	2485.73	1457.56	798.74	419.33
		`Ruyi` (3PC$_{d1}$)	2679.56	1577.57	813.48	419.52
		`pMPL` (3PC)	1189.11	730.19	332.16	157.11
		`pMPL` (3PC$_{d1}$)	1284.76	768.53	379.37	164.85
	500	`Ruyi` (3PC)	642.66	337.77	149.42	69.81
		`Ruyi` (3PC$_{d1}$)	652.92	354.52	163.76	70.83
		`pMPL` (3PC)	294.05	116.55	55.32	25.36
		`pMPL` (3PC$_{d1}$)	300.50	127.04	58.08	27.18
	1000	`Ruyi` (3PC)	311.21	144.45	67.56	31.8
		`Ruyi` (3PC$_{d1}$)	332.56	157.14	69.52	32.84
		`pMPL` (3PC)	120.81	50.94	23.52	9.86
		`pMPL` (3PC$_{d1}$)	124.75	55.77	26.01	10.71
WAN	100	`Ruyi` (3PC)	24.38	23.63	23.08	22.07
		`Ruyi` (3PC$_{d1}$)	24.51	24.17	23.54	22.77
		`pMPL` (3PC)	8.69	6.96	5.34	3.64
		`pMPL` (3PC$_{d1}$)	8.71	6.97	5.35	3.64
	500	`Ruyi` (3PC)	23.11	22.14	19.72	16.22
		`Ruyi` (3PC$_{d1}$)	23.15	22.15	20.21	16.25
		`pMPL` (3PC)	4.92	3.13	1.81	0.96
		`pMPL` (3PC$_{d1}$)	4.95	3.13	1.81	0.96
	1000	`Ruyi` (3PC)	21.82	19.82	16.03	12.93
		`Ruyi` (3PC$_{d1}$)	21.90	19.86	16.13	13.40
		`pMPL` (3PC)	3.11	1.81	0.95	0.24
		`pMPL` (3PC$_{d1}$)	3.12	1.81	0.95	0.28

TABLE IV
ONLINE THROUGHPUT OF LOGISTIC REGRESSION TRAINING COMPARED TO `pMPL` (3PC) AND `pMPL` (3PC$_{d1}$) (ITERATIONS/SECOND).

Setting	D	Framework	B			
			128	256	512	1024
LAN	100	`Ruyi` (3PC)	454.10	249.98	130.41	67.47
		`Ruyi` (3PC$_{d1}$)	465.71	258.9	133.72	69.10
		`pMPL` (3PC)	382.66	272.59	209.99	123.03
		`pMPL` (3PC$_{d1}$)	387.23	282.25	211.21	116.66
	500	`Ruyi` (3PC)	266.28	139.44	76.02	36.20
		`Ruyi` (3PC$_{d1}$)	296.26	154.47	88.26	42.76
		`pMPL` (3PC)	188.40	73.21	40.04	20.85
		`pMPL` (3PC$_{d1}$)	199.58	74.74	50.15	24.97
	1000	`Ruyi` (3PC)	199.21	99.07	47.17	25.86
		`Ruyi` (3PC$_{d1}$)	222.18	114.02	50.94	27.54
		`pMPL` (3PC)	72.62	48.58	25.75	10.52
		`pMPL` (3PC$_{d1}$)	88.11	50.28	24.52	11.71
WAN	100	`Ruyi` (3PC)	6.87	6.41	5.84	5.09
		`Ruyi` (3PC$_{d1}$)	6.92	6.44	5.89	5.18
		`pMPL` (3PC)	1.25	1.20	1.14	1.04
		`pMPL` (3PC$_{d1}$)	1.28	1.24	1.17	1.06
	500	`Ruyi` (3PC)	6.74	6.19	5.57	4.62
		`Ruyi` (3PC$_{d1}$)	6.81	6.33	5.67	4.77
		`pMPL` (3PC)	1.12	0.99	0.80	0.58
		`pMPL` (3PC$_{d1}$)	1.15	1.02	0.82	0.60
	1000	`Ruyi` (3PC)	6.66	6.15	5.34	4.45
		`Ruyi` (3PC$_{d1}$)	6.75	6.17	5.37	4.49
		`pMPL` (3PC)	0.99	0.79	0.57	0.32
		`pMPL` (3PC$_{d1}$)	1.01	0.82	0.60	0.33

improves the online throughput, particularly in the WAN setting. Specifically, in the WAN setting, we improve the online throughput by up to $13.91\times$ and $13.62\times$ for 3PC and 3PC$_{d1}$, respectively. Meanwhile, in the LAN setting, we improve the online throughput by up to $2.74\times$ and $2.52\times$ for 3PC and 3PC$_{d1}$, respectively. The efficiency bottleneck in logistic regression is `Sigmoid` rather than matrix multiplication with truncation in linear regression. Moreover, the improvement of `Sigmoid` over `pMPL` is smaller than that of matrix multiplication with truncation. For instance, according to Table I, for matrix multiplication of size 128×784 and 784×1 with truncation, we reduce the communication rounds by $2\times$ and the communication size by $790\times$. In contrast, for `Sigmoid`, we reduce the communication rounds by $2.8\times$, but the communication size is only reduced by $1.3\times$. Additionally, our `Sigmoid` protocol incurs more local computational overhead [24]. Consequently, the efficiency improvement of logistic regression is not as substantial as that of linear regression.

Neural Networks. We consider a neural network structure consisting of four layers, including one input layer, two hidden layers with 64 nodes, and one output layer with 1 node. Besides, we use `ReLU` as the activation function.

Table V shows the experimental results of `Ruyi` for neural networks training compared to `pMPL`. Specifically, in the WAN setting, we improve the online throughput by up to $2.76\times$ and $2.71\times$ for 3PC and 3PC$_{d1}$, respectively. However, in the LAN setting, `pMPL` is more efficient than `Ruyi`. This is because the most significant efficiency bottleneck in neural networks is `ReLU`. Additionally, the efficiency bottleneck of WAN is communication overhead, while for the LAN setting, local computation overhead is also an important bottleneck. Although we reduce the communication overhead of matrix multiplication with truncation and `ReLU`, we introduce significant local computation overhead in the `ReLU` protocol, which results in lower efficiency in the LAN setting. Furthermore, for `ReLU`, we achieve a reduction of $3\times$ in communication rounds and a reduction of $1.4\times$ in communication size. As a result, as the training task becomes less linear, our efficiency improvement over `pMPL` diminishes.

TABLE V
ONLINE THROUGHPUT OF NEURAL NETWORKS TRAINING COMPARED TO `pMPL` (3PC) AND `pMPL` (3PC$_{d1}$) (ITERATIONS/SECOND).

Setting	D	Framework	B			
			128	256	512	1024
LAN	100	`Ruyi` (3PC)	9.20	3.95	1.92	0.90
		`Ruyi` (3PC$_{d1}$)	9.27	4.06	1.99	0.96
		`pMPL` (3PC)	47.74	27.11	13.82	6.20
		`pMPL` (3PC$_{d1}$)	48.92	27.32	13.68	6.22
	500	`Ruyi` (3PC)	7.55	3.53	1.68	0.82
		`Ruyi` (3PC$_{d1}$)	7.86	3.59	1.78	0.88
		`pMPL` (3PC)	35.88	20.06	10.22	4.66
		`pMPL` (3PC$_{d1}$)	37.96	20.68	10.67	4.72
	1000	`Ruyi` (3PC)	6.38	3.11	1.57	0.76
		`Ruyi` (3PC$_{d1}$)	6.49	3.27	1.61	0.77
		`pMPL` (3PC)	23.21	12.65	6.94	3.58
		`pMPL` (3PC$_{d1}$)	28.77	14.02	7.98	3.86
WAN	100	`Ruyi` (3PC)	1.13	0.68	0.37	0.19
		`Ruyi` (3PC$_{d1}$)	1.14	0.71	0.38	0.19
		`pMPL` (3PC)	0.44	0.35	0.25	0.16
		`pMPL` (3PC$_{d1}$)	0.45	0.35	0.25	0.16
	500	`Ruyi` (3PC)	1.07	0.66	0.37	0.18
		`Ruyi` (3PC$_{d1}$)	1.07	0.67	0.37	0.19
		`pMPL` (3PC)	0.41	0.32	0.22	0.14
		`pMPL` (3PC$_{d1}$)	0.41	0.32	0.23	0.14
	1000	`Ruyi` (3PC)	1.02	0.65	0.36	0.18
		`Ruyi` (3PC$_{d1}$)	1.03	0.66	0.37	0.19
		`pMPL` (3PC)	0.37	0.29	0.20	0.10
		`pMPL` (3PC$_{d1}$)	0.38	0.29	0.20	0.11

C. Evaluation for Offline and Overall

In this subsection, we evaluate the online, offline, and overall efficiency of `Ruyi` against `pMPL` for linear regression, logistic regression, and neural networks. Here, we use the same methods [4] based on OT [23] to generate the $\langle\cdot\rangle$-shares of $[x]_i \cdot [y]_j + [x]_j \cdot [y]_i$ for fair comparison. We set the batch size $B = 128$, and the dimension of feature $D = 784$.

As is shown in Table VI, `Ruyi` performs better than `pMPL` in terms of runtime and communication sizes both offline and overall. This is because, `Ruyi` requires 16 multiplication triples to generate a truncation pair, while `pMPL` needs 52 multiplication triples. Furthermore, `Ruyi` requires 58 and 116 multiplication triplets to complete the offline phase of `ReLU` and `Sigmoid`, respectively, while `pMPL` requires 93 and 187 multiplication triplets, respectively.

TABLE VI
THE ONLINE, OFFLINE, AND OVERALL EFFICIENCY OF RUYI AGAINST pMPL FOR LINEAR REGRESSION (LR), LOGISTIC REGRESSION (LOR), AND NEURAL NETWORKS (NN). "FRAM." STANDS FOR FRAMEWORK AND "COMMU." STANDS FOR COMMUNICATION SIZES. THE RESULTS WITH * ARE SIMULATED.

Model	Fram.	Online		Offline		Overall	
		Time	Comm.	Time	Comm.	Time	Comm.
LR	`Ruyi`	2.45ms	0.05MB	0.73h	4.12GB	0.73h	4.13GB
	`pMPL`	8.43ms	9.31MB	2.37h	10.18GB	2.37h	10.19GB
LOR	`Ruyi`	5.69ms	0.35MB	1.32h	6.47GB	1.32h	6.48GB
	`pMPL`	11.75ms	9.67MB	2.98h	12.46GB	2.98h	12.47GB
NN	`Ruyi`	117.48ms	24.78MB	87.08h*	414.23GB*	87.08h*	414.26GB*
	`pMPL`	44.81ms	46.49MB	1705.73h*	7365.91GB*	1705.73h*	7365.96GB*

D. Evaluation for Multiple Parties

In this subsection, we evaluate the online efficiency of `Ruyi` for linear regression training, which involves multiple privileged (assistant) parties, multiple parties, and multiple assistant parties allowed to drop out. We set the dimension of the feature $D \in \{100, 500, 1000\}$ and the batch size $B = 128$.

TABLE VII
ONLINE THROUGHPUT OF LINEAR REGRESSION TRAINING WITH A VARYING NUMBER OF PRIVILEGED PARTIES np FROM 1 TO 4 (CORRESPONDING TO ASSISTANT PARTIES na FROM 8 TO 5, RESPECTIVELY) WHEN THE NUMBER OF PARTIES $n = 9$ AND ASSISTANT PARTIES ALLOWED TO DROP OUT $nd = 1$ (ITERATIONS/SECOND).

$np : na$	D					
	100		500		1000	
	LAN	WAN	LAN	WAN	LAN	WAN
1:8	642.98	22.42	173.33	20.52	106.66	18.53
2:7	630.47	22.44	170.24	20.58	107.98	18.67
3:6	652.98	22.43	173.42	20.49	102.02	18.54
4:5	645.84	22.42	177.61	20.47	106.49	18.43

Multiple Privileged (Assistant) Parties. We set the number of parties $n = 9$, and the number of assistant parties allowed to drop out $nd = 1$. Meanwhile, we set a varying number of privileged parties np from 1 to 4 (corresponding to assistant parties na from 8 to 5, respectively). As shown in Table VII, the online throughput remains constant as the number of privileged (assistant) parties changes. This is because the communication overhead remains constant as long as the number of parties (n) and assistant parties allowed to drop out nd remains constant, even if the number of privileged parties and assistant parties changes.

Multiple Parties. We set the number of assistant parties allowed to drop out $nd = 1$ and a varying number of parties n from 1 to 9. As shown in Table VIII, the online throughput decreases as the number of parties n increases. This is because the communication overhead increases as n increases. More details on the communication overhead in `Ruyi` are provided in Table I.

TABLE VIII
ONLINE THROUGHPUT OF LINEAR REGRESSION TRAINING WITH THE NUMBER OF ASSISTANT PARTIES ALLOWED TO DROP OUT $nd = 1$ AND A VARYING NUMBER OF PARTIES n FROM 1 TO 9 (ITERATIONS/SECOND).

n	D					
	100		500		1000	
	LAN	WAN	LAN	WAN	LAN	WAN
4	1789.73	24.27	558.91	22.94	268.49	21.52
5	1397.81	24.14	432.28	22.81	205.51	20.73
6	989.24	23.93	280.98	22.38	137.72	20.23
7	861.68	23.69	215.81	22.25	124.93	19.73
8	786.15	23.06	198.67	21.44	114.17	19.14
9	642.98	22.42	173.33	20.52	106.66	18.53

Multiple Assistant Parties Allowed to Drop Out. We set the number of parties, privileged parties, and assistant parties $n = 6, np = 1$, and $na = 5$, respectively. Meanwhile, we set a varying number of assistant parties allowed to drop out nd from 1 to 4. As shown in Table IX, the online throughput decreases as the number of assistant parties allowed to drop out nd increases. This is because the local computation increases as nd increases.

TABLE IX
ONLINE THROUGHPUT OF LINEAR REGRESSION WITH A VARYING NUMBER OF ASSISTANT PARTIES ALLOWED TO DROP OUT nd FROM 1 TO 4 WHEN THE NUMBER OF PARTIES, PRIVILEGED PARTIES, AND ASSISTANT PARTIES $n = 6, np = 1$ AND $na = 5$, RESPECTIVELY (ITERATIONS/SECOND).

nd	D					
	100		500		1000	
	LAN	WAN	LAN	WAN	LAN	WAN
1	989.24	23.93	280.98	22.38	137.72	20.23
2	942.28	23.91	272.80	22.31	129.48	19.93
3	911.94	23.87	253.44	22.27	127.68	19.87
4	843.25	23.85	243.99	22.21	127.08	19.75

VI. DISCUSSION AND FUTURE WORK

Configurability in the Security Model. `Ruyi` provides the configurability in the security model. According to the sharing protocol $\prod_{\text{shr}}(P_\alpha, x)$(Protocol 1) and the reconstruction protocol $\prod_{\text{rec}}(\mathcal{P}, [\![x]\!])$ (Protocol 2) of `Ruyi`, the secret value cannot be reconstructed if any privileged party drops out. Therefore, all the na assistant parties can collude with any $np - 1$ privileged parties. Besides, as privileged parties hold nd alternate shares, np privileged parties can collude with any $na - nd$ assistant parties to reconstruct the secret value. Therefore, all the np privileged parties can collude with any $na - nd - 1$ assistant parties. Based on the above analysis, any $n - nd - 1$ parties in `Ruyi` can collude with each other, providing configurability in the security model. Specifically, if $n - nd - 1 \geq n/2$, the security model of `Ruyi` is the dishonest majority. Conversely, if $n - nd - 1 < n/2$, it is the honest majority.

Comparisons with Additive Secret Sharing. The configurability in the security model provided by `Ruyi` cannot be achieved by the framework based on additive secret sharing. If each privileged party P_j for $j \in \{1, \ldots, np\}$ holds $\langle\cdot\rangle$-share (additive secret share) of the secret value x and all the assistant parties hold the same $\langle\cdot\rangle$-share of the secret value x, i.e. $x = \sum_{j=1}^{n} \langle x \rangle_j + \langle x \rangle_{na}$, $na-1$ assistant parties are allowed to drop out. In this scheme, all privileged parties can collude with any one assistant party to obtain the secret value. This security model is weak for assistant parties. However, in real-world scenarios, the interests of the assistant parties also need to be considered, otherwise, there may be a situation where no assistant party participates in the cooperation. Our proposed Ruyi can realize the configurability of the security model by configuring the number of assistant parties that may drop out so that it can be applied to more real-world scenarios.

Advantages of the Configurability. The configurability feature in Ruyi is practical and provides more support for (future) cooperation among companies. The advantages of adding assistant parties in the system compared to a scheme where only the privileged parties carry out the execution are as follows. Firstly, adding assistant parties could add more training data, thereby (may) training a better machine learning model. Secondly, it protects the interests of assistant parties and motivates more assistant parties to participate in the MPL frameworks. If only the privileged parties carry out the execution, then all privileged parties colluding with each other will obtain the raw data of the assistant parties. In practical scenarios, the assistant parties may be reluctant to lose control of their own data, thus all the parties may reach, before training, a consensus on which parties carry out the execution, that is, configuring the number of privileged parties, assistant parties, and assistant parties allowed to drop out. Besides, the configurability of Ruyi not only allows adding the number of assisting parties, but also configures the number of privileged parties, the number of assistant parties allowed to drop out, and the security model. Therefore, the configurability feature in Ruyi provides more support for (future) cooperation between companies when the number of parties exceeds 3 and the composition of parties is more complex.

Downside of Masked Evaluation Paradigm. Although the masked evaluation paradigm improves online efficiency, it requires an input-independent but function-dependent offline phase. That is, it does not support reactive computation, where we need to know the online-evaluated function in advance during the offline phase.

Future Work. In the future, we aim to support more complex machine learning models, such as AlexNet [29] and VGG16 [30]. Additionally, we intend to support the malicious security model.

VII. Conclusion

In this paper, we present a configurable and efficient MPL framework with privileged parties, referred to as Ruyi. Firstly, to enhance the configurability, we reduce the public matrix constraints of pMPL from four to two while providing the same privileged position. We achieved this by extending the standard resharing paradigm to vector space secret sharing to implement the share conversion protocol. Additionally, we performed all the computations on a prime field instead of a ring. As a result, the Vandermonde matrix always satisfies the public matrix constraints for any given number of parties. Besides, we support configuring the security model (honest majority or dishonest majority) by configuring the number of assistant parties allowed to drop out. Secondly, to improve efficiency, we adapt the masked evaluation paradigm to vector space secret sharing, which leverages an input-independent but function-dependent offline phase to reduce the online communication overhead. Experimental results demonstrate that Ruyi is configurable with multiple privileged (assistant) parties, multiple parties, and multiple assistant parties allowed to drop out. Besides, Ruyi outperforms pMPL by up to 53.87×, 13.91×, and 2.76× for linear regression, logistic regression, and neural networks, respectively.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China under Grant 92370120, Grant 62172100, and in part by the National Key Research and Development Program of China under Grant 2023YFC3304400. We thank all anonymous reviewers, Min Li and Wenyan Li for their insightful comments. Weili Han is the corresponding author.

References

[1] P. Voigt and A. Von dem Bussche, *The EU General Data Protection Regulation (GDPR)*. New York, USA: Springer, 2017.

[2] A. C. Yao, "Protocols for secure computations," in *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*, 1982, pp. 160–164.

[3] L. Song, G. Lin, J. Wang, H. Wu, W. Ruan, and W. Han, "Sok: Training machine learning models over multiple sources with privacy preservation," *arXiv preprint arXiv:2012.03386*, 2020.

[4] P. Mohassel and Y. Zhang, "Secureml: A system for scalable privacy-preserving machine learning," in *2017 IEEE Symposium on Security and Privacy (S&P)*, San Jose, CA, USA, 2017, pp. 19–38.

[5] P. Mohassel and P. Rindal, "ABY3: A mixed protocol framework for machine learning," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '18. New York, NY, USA: Association for Computing Machinery, 2018, pp. 35–52.

[6] A. Patra, T. Schneider, A. Suresh, and H. Yalame, "ABY2.0: Improved Mixed-Protocol secure Two-Party computation," in *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, Aug. 2021, pp. 2165–2182.

[7] L. Song, J. Wang, Z. Wang, X. Tu, G. Lin, W. Ruan, H. Wu, and W. Han, "pMPL: A robust multi-party learning framework with a privileged party," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '22. New York, NY, USA: Association for Computing Machinery, 2022, pp. 2689–2703.

[8] A. Ben-Efraim, M. Nielsen, and E. Omri, "Turbospeedz: Double your online spdz! improving spdz using function dependent preprocessing," in *Applied Cryptography and Network Security: 17th International Conference, ACNS 2019, Bogota, Colombia, June 5–7, 2019, Proceedings 17*. Springer, 2019, pp. 530–549.

[9] Y. Dong, C. Xiaojun, W. Jing, L. Kaiyun, and W. Wang, "Meteor: Improved secure 3-party neural network inference with reducing online communication costs," in *Proceedings of the ACM Web Conference 2023*, ser. WWW '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 2087–2098. [Online]. Available: https://doi.org/10.1145/3543507.3583272

[10] S. D. Gordon, S. Ranellucci, and X. Wang, "Secure computation with low communication from cross-checking," in *Advances in Cryptology–ASIACRYPT 2018: 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2–6, 2018, Proceedings, Part III 24*. Springer, 2018, pp. 59–85.

[11] N. Koti, S. Patil, A. Patra, and A. Suresh, "Mpclan: Protocol suite for privacy-conscious computations," *Journal of Cryptology*, vol. 36, no. 3, p. 22, 2023.

[12] A. Suresh, "Mpcleague: robust mpc platform for privacy-preserving machine learning," *arXiv preprint arXiv:2112.13338*, 2021.

[13] B. Yuan, S. Yang, Y. Zhang, N. Ding, D. Gu, and S.-F. Sun, "Md-ml: Super fast privacy-preserving machine learning for malicious security with a dishonest majority," in *33th USENIX Security Symposium (USENIX Security 24)*. USENIX Association, 2024.

[14] D. Beaver, "Efficient multiparty protocols using circuit randomization," in *Advances in Cryptology - CRYPTO'91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings 11*, ser. Lecture Notes in Computer Science, vol. 576. Springer, 1991, pp. 420–432.

[15] S. Wagh, D. Gupta, and N. Chandran, "SecureNN: 3-party secure computation for neural network training." *Proceedings on Privacy Enhancing Technologies*, vol. 2019, no. 3, pp. 26–49, 2019.

[16] H. Chaudhari, R. Rachuri, and A. Suresh, "Trident: Efficient 4pc framework for privacy preserving machine learning," in *27th Annual Network and Distributed System Security Symposium, NDSS 2020, San Diego, California, USA, February 23-26, 2020*. The Internet Society, 2020.

[17] S. Wagh, S. Tople, F. Benhamouda, E. Kushilevitz, P. Mittal, and T. Rabin, "Falcon: Honest-majority maliciously secure framework for private deep learning," *Proceedings on Privacy Enhancing Technologies*, vol. 2021, no. 1, pp. 188–208, 2021.

[18] A. Dalskov, D. Escudero, and M. Keller, "Fantastic Four: Honest-Majority Four-Party secure computation with malicious security," in *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, Aug. 2021, pp. 2183–2200.

[19] C. Dong, J. Weng, J. Liu, Y. Zhang, Y. Tong, A. Yang, Y. Cheng, and S. Hu, "Fusion: Efficient and secure inference resilient to malicious servers," in *30th Annual Network and Distributed System Security Symposium, NDSS 2023, San Diego, California, USA, February 27 - March 3, 2023*. The Internet Society, 2023.

[20] A. Brüggemann, O. Schick, T. Schneider, A. Suresh, and H. Yalame, "Don't eject the impostor: Fast three-party computation with a known cheater," in *2024 IEEE Symposium on Security and Privacy (S&P)*. Los Alamitos, CA, USA: IEEE Computer Society, may 2024, pp. 167–167.

[21] E. F. Brickell, "Some ideal secret sharing schemes," in *Advances in Cryptology - EUROCRYPT '89, Workshop on the Theory and Application of Cryptographic Techniques, Houthalen, Belgium, April 10-13, 1989, Proceedings*, ser. Lecture Notes in Computer Science, J. Quisquater and J. Vandewalle, Eds., vol. 434. Springer, 1989, pp. 468–475.

[22] E. A. Rawashdeh, "A simple method for finding the inverse matrix of vandermonde matrix," *Mat. Vesn*, vol. 71, pp. 207–213, 2019.

[23] G. Asharov, Y. Lindell, T. Schneider, and M. Zohner, "More efficient oblivious transfer and extensions for faster secure computation," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*. New York, NY, USA: Association for Computing Machinery, 2013, pp. 535–548.

[24] O. Catrina and S. De Hoogh, "Improved primitives for secure multiparty integer computation," in *Security and Cryptography for Networks: 7th International Conference, SCN 2010, Amalfi, Italy, September 13-15, 2010. Proceedings 7*. Springer, 2010, pp. 182–199.

[25] Y. Li, Y. Duan, Z. Huang, C. Hong, C. Zhang, and Y. Song, "Efficient 3PC for binary circuits with application to Maliciously-Secure DNN inference," in *32nd USENIX Security Symposium (USENIX Security 23)*. Anaheim, CA: USENIX Association, Aug. 2023, pp. 5377–5394.

[26] M. B. Santos, D. Mouris, M. Ugurbil, S. Jarecki, J. Reis, S. Sengupta, and M. de Vega, "Curl: Private LLMs through wavelet-encoded look-up tables," Cryptology ePrint Archive, Paper 2024/1127, 2024. [Online]. Available: https://eprint.iacr.org/2024/1127

[27] O. Catrina, "Round-efficient protocols for secure multiparty fixed-point arithmetic," in *2018 International Conference on Communications (COMM)*. IEEE, 2018, pp. 431–436.

[28] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[29] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[30] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015.

[31] R. Canetti, "Security and composition of multiparty cryptographic protocols," *Journal of CRYPTOLOGY*, vol. 13, no. 1, pp. 143–202, 2000.

Appendix

A. Security Proof

In this section, we provide security proof for our designed sharing protocol, reconstruction protocol, and basic primitives in Section IV using the standard real/ideal world paradigm.

Proof. Let the semi-honest adversary $\mathcal{A}$ corrupt at most $n_p - 1$ privileged parties and all n_a assistant parties. We use $\mathcal{S}$ to denote an ideal-world static adversary (simulator) for $\mathcal{A}$, who acts as the honest parties and simulates the messages received by $\mathcal{A}$ during the protocol. We present the steps of $\mathcal{S}$ for in the stand-alone model with security under sequential composition [31]. Without loss of generality, for each of the constructions, we provide the simulation for the case of corrupt $(P_2, \ldots, P_n)$. Note that the case where the semi-honest adversary $\mathcal{A}$ corrupts at most n_p privileged parties and any $n_a - nd - 1$ assistant parties is similar to the case discussed above, thus is not discussed here.

Security for Sharing Protocol $\prod_{\text{shr}}(P_x, x)$ (Protocol 1). For the offline phase, if $P_\alpha \neq P_1$, $\mathcal{S}$ has to do nothing since $\mathcal{A}$ is not receiving any messages. $\mathcal{S}$ receives $[\eta_x]_1$, $[\eta_x]_{n+1}, \ldots, [\eta_x]_{n+nd}$ from $\mathcal{A}$ on behalf of P_1. If $P_\alpha = P_1$, $\mathcal{S}$ follows the offline phase of the sharing protocol $\prod_{\text{shr}}(P_i, x)$ (Protocol 1) honestly on behalf of P_1. For the online phase, if $P_\alpha \neq P_1$, $\mathcal{S}$ has to do nothing since $\mathcal{A}$ is not receiving any messages. $\mathcal{S}$ receives γ_x from $\mathcal{A}$ on behalf of P_1. If $P_\alpha = P_1$, $\mathcal{S}$ sets $x = 0$ and performs the protocol steps honestly.

Security for Reconstruction Protocol $\prod_{\text{rec}}(\mathcal{P}, [\![x]\!])$ (Protocol 2). If no assistant party drops out, $\mathcal{S}$ is given the output x, which is the output of $\mathcal{A}$. $\mathcal{S}$ sends $[\eta_x]_1$ to $\mathcal{A}$ on behalf of P_1. $\mathcal{S}$ receives $[\eta_x]_2, \ldots, [\eta_x]_n$ from $\mathcal{A}$ on behalf of P_1. If m $(1 \leq m \leq nd)$ assistant parties $P_{d_1}, \ldots, P_{d_m}$ drop out, $\mathcal{S}$ is given the output x, which is the output of $\mathcal{A}$. $\mathcal{S}$ sends $[\eta_x]_1, [\eta_x]_{n+1}, \ldots, [\eta_x]_{n+nd}$ to $\mathcal{A}$ on behalf of P_1. $\mathcal{S}$ receives $[\eta_x]_i$ for $i \in \{2, \ldots, n\}, i \neq d_1, \ldots, d_m$ from $\mathcal{A}$ on behalf of P_1.

Security for Linear Operations. There is nothing to simulate as the linear operations are non-interactive.

Security for Secure Multiplication Protocol $\prod_{\text{mul}}(\mathcal{P}, [\![x]\!], [\![y]\!])$ (Protocol 3). For the offline phase, we first consider the randomized multiplication protocol $\prod_{\text{rand\_mul}}(\mathcal{P}, [\eta_x], [\eta_y])$(Protocol 4) as an ideal functionality $\mathcal{F}^{[\![\cdot]\!]}_{\text{rand\_mul}}$. In step 2, we use the method in [4] in a black-box manner, so the simulation for the same follows from the security analyzed in [4]. In other steps, $\mathcal{S}$ performs the protocol honestly. Secondly, we consider the share conversion protocol $\prod_{\text{convert}}(\mathcal{P}, \langle x \rangle)$ (Protocol 5) as an ideal functionality $\mathcal{F}^{[\![\cdot]\!]}_{\text{convert}}$. $\mathcal{S}$ sends $[\langle x \rangle_1]_2, \ldots, [\langle x \rangle_1]_n$ to $\mathcal{A}$ on behalf of P_1. If $np > 1$, $\mathcal{S}$ sends $[\langle x \rangle_1]_{n+1}, \ldots, [\langle x \rangle_1]_{n+nd}$ to $\mathcal{A}$ on behalf of P_1. $\mathcal{S}$ receives $([\langle x \rangle_2]_1, [\langle x \rangle_2]_{n+1}, \ldots, [\langle x \rangle_2]_{n+nd})$, $\ldots$, $([\langle x \rangle_n]_1, [\langle x \rangle_n]_{n+1}, \ldots, [\langle x \rangle_n]_{n+nd})$ on behalf of P_1. Finally, we consider the random value generation protocol $\prod_{\text{rand\_gen}}(\mathcal{P})$ (Protocol 6) as an ideal functionality $\mathcal{F}^{[\![\cdot]\!]}_{\text{rand\_gen}}$. $\mathcal{S}$ performs the protocol honestly. The security of the offline phase in secure multiplication protocol $\prod_{\text{mul}}(\mathcal{P}, [\![x]\!], [\![y]\!])$ (Protocol 3) follows in the $(\mathcal{F}^{[\![\cdot]\!]}_{\text{rand\_mul}}, \mathcal{F}^{[\![\cdot]\!]}_{\text{convert}}, \mathcal{F}^{[\![\cdot]\!]}_{\text{rand\_gen}})$-hybrid model. For the online phase, $\mathcal{S}_{\text{mul}}$ follows the step honestly using the data obtained from the corresponding offline phase.

Security for Secure Truncation Protocol $\prod_{\mathrm{trun}}(\mathcal{P}, [\![z]\!])$ (Protocol 7). For the offline phase, we invoke `TruncPR` in a black-box manner as [24], [27]. Therefore, the simulation for the same follows from the security analyzed in [24], [27]. For the online phase, there is nothing to simulate as the secure truncation protocol is non-interactive.

Security for Secure Comparison Protocol $\prod_{\mathrm{compar}}(\mathcal{P}, [\![x]\!])$. We invoke the comparison protocol $\prod_{\mathrm{compar}}$ in a black-box manner as [24], [27]. Therefore, the simulation for the same follows from the security analyzed in [24], [27].

Since all the messages sent and received in the protocol are uniformly random values in both the real protocol and the simulation, the $\mathcal{A}$'views in the real and ideal worlds are identically distributed and indistinguishable. This concludes the proof.

□

B. Building Blocks

Linear Operations on Matrix. It is trivial to extend linear operations on shared values x and y to shared matrices $\mathbf{X}$ (with the size of $a \times b$) and $\mathbf{Y}$ (with the size of $a \times b$). To get $[\![\mathbf{Z}]\!] = t_1 \cdot [\![\mathbf{X}]\!] + t_2 \cdot [\![\mathbf{Y}]\!] + t_3$, each party P_i for $i \in \{1, \ldots, n\}$ locally computes $\gamma_{\mathbf{Z}} = t_1 \cdot \gamma_{\mathbf{X}} + t_2 \cdot \gamma_{\mathbf{Y}} + t_3$ and $[\eta_{\mathbf{Z}}]_i = t_1 \cdot [\eta_{\mathbf{X}}]_i + t_2 \cdot [\eta_{\mathbf{Y}}]_i$. Besides, privileged parties locally compute $[\eta_{\mathbf{Z}}]_{n+m} = t_1 \cdot [\eta_{\mathbf{X}}]_{n+m} + t_2 \cdot [\eta_{\mathbf{Y}}]_{n+m}$ for $m \in \{1, \ldots, nd\}$ additionally.

Dot Product with Truncation. Given the $[\![\cdot]\!]$-shares of l-element vectors $\vec{\mathbf{x}}$ and $\vec{\mathbf{y}}$ held by each party P_i for $i \in \{1, \ldots, n\}$, the secure dot product with truncation protocol $\prod_{\mathrm{dotp}}(\mathcal{P}, [\![\vec{\mathbf{x}}]\!], [\![\vec{\mathbf{y}}]\!])$ (Protocol 8) is to generate $[\![z^t]\!]$ such that $z^t = \lfloor z/2^f \rfloor = \lfloor (\vec{\mathbf{x}} \odot \vec{\mathbf{y}})/2^f \rfloor = \lfloor (\sum_{j=1}^{l} x_j \cdot y_j)/2^f \rfloor$. In the offline phase, parties in $\mathcal{P}$ generate $[\cdot]$-shares of $\eta_{x_j} \cdot \eta_{y_j}$ for each multiplication $x_j \cdot y_j$ ($j \in \{1, \ldots, l\}$) in parallel executing the randomized multiplication protocol $\prod_{\mathrm{rand\_mul}}(\mathcal{P}, [\eta_{x_j}], [\eta_{y_j}])$ (Protocol 4). Besides, instead of generating $[\cdot]$-shares of η_{z_j} and $\eta^t_{z_j}$, where $\eta^t_{z_j} = \lfloor \eta_{z_j}/2^f \rfloor$ for each multiplication $x_j \cdot y_j$, parties in $\mathcal{P}$ only need to generate the $[\cdot]$-shares of η_z and η^t_z, where $\eta^t_z = \lfloor \eta_z/2^f \rfloor$ by executing the random value generation protocol $\prod_{\mathrm{rand\_gen}}(\mathcal{P})$ (Protocol 6) and the offline phase of the secure truncation protocol $\prod_{\mathrm{trunc}}(\mathcal{P}, [\![z]\!])$ (Protocol 7), respectively. In the online phase, each party P_i for $i \in \{1, \ldots, n\}$ locally computes $[\gamma_z{}']_i = \sum_{j=1}^{l}(-\gamma_{x_j} \cdot [\eta_{y_j}]_i - \gamma_{y_j} \cdot [\eta_{x_j}]_i + [\eta_{x_j} \cdot \eta_{y_j}]_i) + [\eta_z]_i$. Then parties in $\mathcal{P}$ exchange the $[\cdot]$-shares of $\gamma_z{}'$ to reconstruct it. After that, each party P_i for $i \in \{1, \ldots, n\}$ locally computes $\gamma_z = \sum_{j=1}^{l} \gamma_{x_j} \cdot \gamma_{x_j} + \gamma_z{}'$. Finally, parties in $\mathcal{P}$ obtain the $[\![\cdot]\!]$-shares of z by executing the online phase of the secure truncation protocol $\prod_{\mathrm{trunc}}(\mathcal{P}, [\![z]\!])$ (Protocol 7).

Matrix Multiplication with Truncation. We follow the previous method in [4], [6], [7] to generalize the multiplication operation on shared values x and y to shared matrices $\mathbf{X}$ (with the size of $a \times b$) and $\mathbf{Y}$ (with the size of $b \times c$). Given the $[\![\cdot]\!]$-shares of matrices $\mathbf{X}$ and $\mathbf{Y}$ held by each party P_i for $i \in \{1, \ldots, n\}$, the secure matrix multiplication with truncation protocol $\prod_{\mathrm{mmul}}(\mathcal{P}, [\![\mathbf{X}]\!], [\![\mathbf{Y}]\!])$ is to generate $[\![\cdot]\!]$-shares of $\mathbf{Z^t}$ such that $\mathbf{Z^t} = \lfloor \mathbf{Z^t}/2^f \rfloor = \lfloor (\mathbf{X} \times \mathbf{Y})/2^f \rfloor$. In the offline phase, parties in $\mathcal{P}$ generate $[\cdot]$-shares of $\eta_{\mathbf{X}} \times \eta_{\mathbf{Y}}$, $\eta_{\mathbf{Z}}$ and $\eta^t_{\mathbf{Z}}$, where $\eta^t_{\mathbf{Z}} = \lfloor \eta_{\mathbf{Z}}/2^f \rfloor$. In the online phase, each party P_i for $i \in \{1, \ldots, n\}$ locally computes $[\gamma_{\mathbf{Z}}{}']_i = -\gamma_{\mathbf{X}} \times [\eta_{\mathbf{Y}}]_i - [\eta_{\mathbf{X}}]_i \times \gamma_{\mathbf{Y}} + [\eta_{\mathbf{X}} \times \eta_{\mathbf{Y}}]_i + [\eta_{\mathbf{Z}}]_i$. Then parties in $\mathcal{P}$ exchange the $[\cdot]$-shares of $\gamma_{\mathbf{Z}}{}'$ to reconstruct it. Furthermore, each party P_i for $i \in \{1, \ldots, n\}$ locally computes $\gamma_{\mathbf{Z}} = \gamma_{\mathbf{X}} \times \gamma_{\mathbf{Y}} + \gamma_{\mathbf{Z}}{}'$. Finally, parties in $\mathcal{P}$ locally truncate $\gamma_{\mathbf{Z}}$ to obtain the truncated value $\gamma_{\mathbf{Z}}{}^t = \lfloor \gamma_{\mathbf{Z}}/2^f \rfloor$ and set the truncated share of $[\eta_{\mathbf{Z}}]$ as $[\eta^t_{\mathbf{Z}}]$.

Activation Functions. We consider two most widely used non-linear activation functions for MPL, i.e. `ReLU` and `Sigmoid`, where `ReLU` is used in BP neural networks and `Sigmoid` is used in logistic regression.

Protocol 8: $\prod_{\mathrm{dotp}}(\mathcal{P}, [\![\vec{\mathbf{x}}]\!], [\![\vec{\mathbf{y}}]\!])$

Input: $[\![\cdot]\!]$-shares of $\vec{\mathbf{x}}$ and $\vec{\mathbf{y}}$ held by each party.
Output: $[\![\cdot]\!]$-shares of z^t, where $z^t = \lfloor (\vec{\mathbf{x}} \odot \vec{\mathbf{y}})/2^f \rfloor$.

Offline:

1: Parties in $\mathcal{P}$ generate $[\cdot]$-shares of $\eta_{x_j} \cdot \eta_{x_j}$ for $j \in \{1, \ldots, l\}$ in parallel by executing the randomized multiplication protocol $\prod_{\mathrm{rand\_mul}}(\mathcal{P}, [\eta_{x_j}], [\eta_{y_j}])$ (Protocol 4).
2: Parties in $\mathcal{P}$ generate $[\cdot]$-shares of random value η_z and η^t_z where $\eta^t_z = \lfloor \eta_z/2^f \rfloor$ by executing the random value generation protocol $\prod_{\mathrm{rand\_gen}}(\mathcal{P})$ (Protocol 6) and the offline phase of the secure truncation protocol $\prod_{\mathrm{trunc}}(\mathcal{P}, [\![z]\!])$ (Protocol 7), respectively.

Online:

1: Each party P_i for $i \in \{1, \ldots, n\}$ locally computes $[\gamma_z{}']_i = \sum_{j=1}^{l}(-\gamma_{x_j} \cdot [\eta_{x_j}]_i - \gamma_{x_j} \cdot [\eta_{x_j}]_i + [\eta_{x_j} \cdot \eta_{x_j}]_i) + [\eta_z]_i$.
2: Parties in $\mathcal{P}$ exchange the $[\cdot]$-shares of $\gamma_z{}'$ to reconstruct it.
3: Each party P_i for $i \in \{1, \ldots, n\}$ locally computes $\gamma_z = \sum_{j=1}^{l} \gamma_{x_j} \cdot \gamma_{x_j} + \gamma_z{}'$.
4: Each privileged party P_j for $j \in \{1, \ldots, np\}$ locally sets the truncated share $[\![z^t]\!]_j = ([\eta^t_z]_j, [\eta^t_z]_{n+1}, \ldots, [\eta^t_z]_{n+nd}, \lfloor \gamma_z/2^f \rfloor)$, while each assistant party P_j for $j \in \{np+1, \ldots, n\}$ locally sets the truncated share $[\![z^t]\!]_j = ([\eta^t_z]_j, \lfloor \gamma_z/2^f \rfloor)$.

Protocol 9: $\prod_{\mathrm{relu}}(\mathcal{P}, [\![x]\!]))$

Input: $[\![\cdot]\!]$-shares of value x held by each party.
Output: $[\![\cdot]\!]$-shares of z, where $z = ((1 - cp) \cdot x), cp = x < 0?1:0$.

Offline:

1: Parties in $\mathcal{P}$ execute the offline phase of the secure comparison protocol $\prod_{\mathrm{compar}}(\mathcal{P}, [\![x]\!])$ and the offline phase of the secure multiplication protocol $\prod_{\mathrm{mul}}(\mathcal{P}, [\![1 - cp]\!], [\![x]\!])$ (Protocol 3).

Online:

1: Parties in $\mathcal{P}$ execute the secure comparison protocol $\prod_{\mathrm{compar}}(\mathcal{P}, [\![x]\!])$ to generate the $[\![\cdot]\!]$-shares of cp.
2: Each party P_i for $i \in \{1, \ldots, n\}$ locally computes $[\![1 - cp]\!]_i$.
3: Parties in $\mathcal{P}$ execute the secure multiplication protocol $\prod_{\mathrm{mul}}(\mathcal{P}, [\![(1 - cp)]\!], [\![x]\!])$ (Protocol 3).

`ReLU`. The `ReLU` function is defined as `ReLU`$(x) =$

$max(0, x)$, which can be viewed as $\texttt{ReLU}(x) = (1 - cp) \cdot x$, where $cp = x < 0?1 : 0$. Given the $[\![\cdot]\!]$-shares of x held by each party P_i for $i \in \{1, \ldots, n\}$ the protocol $\prod_{\text{relu}}(\mathcal{P}, [\![x]\!])$ (Protocol 9) enables parties in $\mathcal{P}$ to compute $[\![\cdot]\!]$-shares of z, where $z = (1 - cp) \cdot x$. Firstly, parties in $\mathcal{P}$ execute the secure comparison protocol $\prod_{\text{compar}}(\mathcal{P}, [\![x]\!])$ to generate $[\![\cdot]\!]$-shares of cp. Then each party P_i for $i \in \{1, \ldots, n\}$ locally computes $[\![1 - cp]\!]_i$. Finally, parties in $\mathcal{P}$ execute the secure multiplication protocol $\prod_{\text{mul}}(\mathcal{P}, [\![(1 - cp)]\!], [\![x]\!])$ (Protocol 3).

As for the derivative of $\texttt{ReLU}$ used in the back-propagation of BP neural networks, it is denoted as $\texttt{ReLU}^{-1}(x) = (1 - cp)$.

`Sigmoid`. The `Sigmoid` function is defined as $\texttt{Sigmoid}(x) = 1/(1 + e^{-x})$. In this paper, we use an MPC-friendly version of the `Sigmoid` function as previous works [4]–[7], which can be viewed as $\texttt{Sigmoid}(x) = (1 - cp1) \cdot cp2 \cdot (x + 1/2) + (1 - cp2)$, where $cp1 = (x + 1/2) < 0?1 : 0, cp2 = (x - 1/2) < 0?1 : 0$. $\prod_{\text{sig}}(\mathcal{P}, [\![x]\!])$ is similar to $\prod_{\text{relu}}(\mathcal{P}, [\![x]\!])$ (Protocol 9) and therefore we omit the details here.

C. Correctness Proof

The η_z^t is correct generated by the secure truncation protocol $\prod_{\text{trun}}(\mathcal{P}, [\![z]\!])$ (Protocol 7) because η_z is not locally truncated among parties, i.e. $[\eta_z^t]_i \neq \lfloor [\eta_z]_i / 2^f \rfloor$ for $i \in \{1, \ldots, n\}$. ABY^3 points out that extending local truncation to the three-party setting leads to a high probability of the most significant bit (MSB) error, thus causing incorrect truncation results. However, in our secure truncation protocol, η_z is probabilistically truncated without any MSB error. Specifically, our secure truncation protocol generates $[\cdot]$-shares of η_z^t, where $\eta_z^t = \lfloor \eta_z / 2^f \rfloor + u$ and $u = (\eta_z / 2^f - \lfloor \eta_z / 2^f \rfloor \geq 0.5)?1 : 0$. The correctness of our secure truncation protocol in generating $[\cdot]$-shares of η_z^t is based on the correctness of the TrunPR protocol in [24] and the detailed correctness proof is shown as follows.

Firstly, after executing the TruncPR protocol with input $[\eta_z]_i$ from each party P_i for $i \in \{0, \ldots, n\}$, each party P_i obtains $[\eta_z^t]_i$, where $\eta_z^t = \lfloor \eta_z / 2^f \rfloor + u$, $u \in \{0, 1\}$ (the detailed correctness proof is shown in [24]). Here, $[\eta_z]$ is probabilistically truncated to $[\eta_z^t]$ without any MSB error by the TruncPR protocol rather than locally truncated.

Then each party P_i computes $\langle \eta_z^t \rangle = c_{0,i} \cdot [\eta_z^t]_i$ for $i \in \{0, \ldots, n\}$ to convert $[\cdot]$-share of η_z^t to its equivalent $\langle \cdot \rangle$-share so that $\eta_z^t = \sum_{i=0}^{n} \langle \eta_z^t \rangle_i$. In fact, the conversion is to use $\langle \eta_z^t \rangle_i$ to represent $c_{0,i} \cdot [\eta_z^t]_i$, which does not affect the correctness of the secure truncation protocol results. Therefore, even if we convert $[\cdot]$-share to $\langle \cdot \rangle$-share, η_z is also probabilistically truncated without any MSB error rather than locally truncated.

Finally, parties in $\mathcal{P}$ execute the share conversion protocol $\prod_{\text{convert}}(\mathcal{P}, \langle \eta_z^t \rangle_i)$ to convert $\langle \cdot \rangle$-shares of η_z^t to its equivalent $[\cdot]$-shares, including alternate shares. This step is the standard resharing paradigm, which does not affect the correctness of the secure truncation protocol results either.

In summary, our secure truncation protocol results in correct $[\cdot]$-shares of η_z^t, because the above steps are all correct without any MSB error.