Efficient Privacy-preserving Vertical Logistic Regression under Two Security Definitions

Zongda Han[1], Xiang Cheng[1*], Changkang Chi[1], Yuejia Huang[1], Zhaofeng He[1], Sen Su[1]
[1]*State Key Laboratory of Networking and Switching Technology,*
Beijing University of Posts and Telecommunications, Beijing, China
{jumdar,chengxiang,ckchi,huangyuejia,zhaofenghe,susen}@bupt.edu.cn

ABSTRACT

Privacy-preserving vertical logistic regression allows multiple data owners to collaboratively build a logistic regression (LR) model over vertically partitioned data without disclosing sensitive information. In this paper, we tackle the problem of vertical LR under infinite solution security and provable security, respectively. Existing protocols that have claimed to satisfy infinite solution security might not be able to fulfill this claim and use public-key cryptography techniques with high computational overhead to perform model updates. For existing protocols with provable security, most of them employ secret sharing to perform model training and inference, which incurs large communication overhead. To address these problems, we first present an efficient vertical LR protocol with infinite solution security (*ISS-VLR*). In *ISS-VLR*, we establish two rigorous security constraints to guarantee infinite solution security during model training and inference, respectively. Moreover, we propose a random mask-based privacy-preserving stochastic gradient descent (*RM-SGD*), which utilizes random masks to protect intermediate results and exploit the homomorphism property of random masks to efficiently perform model updates. Then, we present an efficient vertical LR protocol with provable security (*PS-VLR*). In *PS-VLR*, we adopt matrix triples to reduce the communication overhead introduced by secret sharing. Moreover, we propose a rotation-free Single-Instruction Multiple-Data (SIMD) based matrix-vector multiplication method (*RS-MVM*) to accelerate matrix triple generation by reducing the computational overhead of homomorphic matrix-vector multiplication. We further propose a hybrid sigmoid approximation method (*HY-SIG*) to effectively and efficiently approximate the sigmoid function under secret sharing. Experimental results show that *ISS-VLR* and *PS-VLR* can be more efficient than state-of-the-art protocols under infinite solution security and provable security, respectively.

PVLDB Reference Format:
Zongda Han[1], Xiang Cheng[1*], Changkang Chi[1], Yuejia Huang[1], Zhaofeng He[1], Sen Su[1]. Efficient Privacy-preserving Vertical Logistic Regression under Two Security Definitions. PVLDB, 14(1): XXX-XXX, 2020.
doi:XX.XX/XXX.XX

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/HanZongda-bupt/secure-vlr.

1 INTRODUCTION

Logistic regression (LR) is a popular machine learning model used in many fields, such as advertising [33], finance [4], and medicine [5]. Stochastic gradient descent (SGD) is widely used to train the LR model iteratively, which composes two stages per iteration, namely forward computation and backward updating. The forward computation stage involves sequential evaluation of the linear function and the sigmoid function to obtain model outputs, while the backward updating stage involves gradient computation, followed by updating the model parameters. To train an LR model with high accuracy, it is crucial to have sufficient high-quality data. In many cases, data are distributed vertically among multiple data owners, especially in the business-to-business context. That is, all data owners hold disjoint features but overlap on a subset of the samples. However, due to the presence of sensitive information [46], data owners are reluctant to share the data, which is not conducive to building useful models. To address this issue, various privacy-preserving vertical LR protocols are proposed to facilitate collaborative LR model building among multiple data owners over vertically partitioned data without compromising privacy.

In this paper, we focus on vertical LR under two types of security definitions, namely infinite solution security [21, 47] and provable security [23], in the multi-party setting. Infinite solution security permits the exposure of certain intermediate results on the condition that any adversary, after obtaining these intermediate results, could deduce an infinite amount of possible sensitive information. For provable security, it requires the use of cryptography techniques, such as homomorphic encryption (HE) and secure multi-party computation (MPC), to protect intermediate results. These two types of security definitions have unique advantages. Selecting the appropriate security definition for constructing a protocol based on practical requirements and privacy needs facilitates achieving a desired trade-off between privacy and utility.

Existing protocols with infinite solution security [21, 43, 47–49] expose the linear-functional outputs during forward computation and establish security constraints to ensure that such exposure satisfies infinite solution security. However, after reviewing the process of collaborative LR model building, we found that these protocols ignore two pieces of information available to collaborators: the value ranges of input features and the correlation of model parameters across multiple iterations, which may result in that their declared security constraints cannot provide infinite solution security. Moreover, during backward updating, these protocols employ public-key cryptography techniques, which can provide provable

Proceedings of the VLDB Endowment, Vol. 14, No. 1 ISSN 2150-8097.
doi:XX.XX/XXX.XX

The corresponding author of this paper is Prof. Xiang Cheng.

security while incurring large computational overhead. According to the "barrel theory", we argue that providing infinite solution security as in forward computation is sufficient for backward updating. For existing protocols with provable security [11, 13–15, 30, 31], most of them employ secret sharing (SS) to perform model training and inference, as SS can offload most computations to the offline phase. Nevertheless, SS requires frequent interactions between collaborators in the online phase, which incurs high communication overhead. Although matrix triple [9, 40], a technique for supporting online matrix multiplications, can be adopted to reduce the communication overhead, the offline generation of matrix triples relies on threshold fully homomorphic encryption (threshold FHE) and requires a substantial amount of time in the multi-party setting. Accelerating homomorphic matrix-vector multiplication is the core of improving the efficiency of matrix triple generation. However, existing homomorphic matrix-vector multiplication methods [22, 28, 29] are unable to avoid the use of costly rotation operations while utilizing Single-Instruction Multiple-Data (SIMD) operations to improve the computational efficiency of homomorphic computations.

To address the above problems, we first present an efficient **V**ertical **LR** protocol with **I**nfinite **S**olution **S**ecurity, which is referred to as *ISS-VLR*. In *ISS-VLR*, taking the value ranges of input features and the correlation of model parameters across multiple iterations into consideration, we establish two rigorous security constraints to ensure that the exposure of linear-functional outputs satisfies infinite solution security during training and inference, respectively. Specifically, we formalize the linear-functional outputs and the information related to the input features and model parameters as a joint system of equations and derive security constraints by making such a system have infinitely many solutions. Moreover, to achieve an efficient backward updating while guaranteeing infinite solution security, we propose a **R**andom **M**ask-based privacy-preserving **SGD**, which is referred to as *RM-SGD*. In *RM-SGD*, we utilize random masks to guarantee the confidentiality of intermediate results from all parties while utilizing the homomorphism property of random masks to enable efficient gradient calculation and model parameter update.

Then, we present an efficient **V**ertical **LR** protocol with **P**rovable **S**ecurity, which is referred to as *PS-VLR*. In *PS-VLR*, we employ SS to perform model training and inference, where matrix triples [9, 40] are adopted to compute matrix multiplications. Moreover, to efficiently generate matrix triples, we propose a **R**otation-free **S**IMD-based **M**atrix-**V**ector **M**ultiplication method, which is referred to as *RS-MVM*. In *RS-MVM*, we decompose the homomorphic matrix-vector multiplication into multiple SIMD-based homomorphic multiplications and additions without invoking the expensive rotation operation. Meanwhile, we also exploit the parallel computing capability of SIMD operations to simultaneously perform multiple matrix-vector multiplications during a single execution. Furthermore, since the sigmoid function cannot be directly evaluated by SS, we propose a **HY**brid **SIG**moid approximation method (*HY-SIG*), which leverages the properties of the piecewise function and the third-order polynomial function to effectively and efficiently approximate the sigmoid function under SS.

In summary, we make the following contributions:

- We present *ISS-VLR*, a protocol for LR model training and inference with infinite solution security. In *ISS-VLR*, we establish rigorous security constraints for forward computation and propose *RM-SGD* to perform efficient and privacy-preserving backward updating.
- We present *PS-VLR*, a protocol for LR model training and inference with provable security. In *PS-VLR*, we use matrix triples to reduce the communication overhead introduced by SS and propose *RS-MVM* to improve the efficiency of matrix triple generation. We further propose *HY-SIG* to approximately evaluate the sigmoid function.
- We conduct the experiments using both real and synthetic datasets. The results show that *ISS-VLR* and *PS-VLR* can achieve higher efficiency than state-of-the-art vertical LR protocols under infinite solution security and provable security, respectively.

Roadmap. Section 2 provides the preliminaries. Section 3 and Section 4 give the details of *ISS-VLR* and *PS-VLR*, respectively. Section 5 shows our experimental results. Section 6 reviews the related work. Section 7 concludes this paper.

2 PRELIMINARIES

In this section, we briefly present our system and threat models, as well as necessary background knowledge.

2.1 System Model

We focus on building a logistic regression (LR) model over vertically partitioned data in a p-party setting ($p > 2$) without relying on additional server assistance. The parties are categorized into two types: one active party and many passive parties. Each party P_i holds a local dataset $D_i = (ID_i, X_i)$, where ID_i is a individual identifier and X_i is the feature set. Only the active party holds the labels $Y = \{y_i\}_{i=1}^{n}$, where $y_i \in \{0, 1\}$ and n is the number of samples. For any two local datasets D_i and D_j, there are disjoint features and a common subset of samples. To allow collaborative training, we assume that all parties have aligned their common samples $ID = ID_1 \cap \cdots \cap ID_p$ via private set intersection [6, 10] prior to model training.

2.2 Threat Model and Security Definition

We consider the setting where an adversary is semi-honest and can corrupt up to p-1 of the p parties. Within this setting, the adversary adheres to the protocol and aims to extract sensitive information from the messages received by the corrupted parties. The corruption strategy is static, where the set of corrupted clients remains unchanged during the execution of protocols. Our proposed protocols are built on two different security definitions, namely infinite solution security and provable security. In particular, the infinite solution security [21, 47] is defined in Definition 2.1. For the provable security, we follow the definition of that for static semi-honest adversaries in [23], as shown in Definition 2.2.

Definition 2.1. (Infinite Solution Security) Let $\Pi(X, I, Y)$ be a p-party protocol, where X denotes the input of all parties, I denotes the intermediate results generated during the execution of Π, and Y denotes the output of Π. We say that Π satisfies infinite solution security if the system of equations constructed from any $Z \subseteq \{I, Y\}$ has an infinite number of solutions.

Definition 2.2. (Provable Security) Let $\mathcal{F} = \left\{f_1, \ldots, f_p\right\}$ be an ideal functionality and Π be a p-party protocol for computing $\mathcal{F}$. Given input data $\bar{x} = \left(x_1, \ldots, x_p\right)$, where x_i is from P_i, the view of P_i during the execution of Π is denoted by $\text{view}_i^{\Pi}(\bar{x})$, and the outputs of all parties during the execution of Π are denoted by $\text{output}^{\Pi}(\bar{x})$. We say that Π securely computes $\mathcal{F}$ if 1) for any $I = \{i_1, \ldots, i_t\} \subseteq [p] \overset{\text{def}}{=} \{1, \ldots, p\}$, there exists a probabilistic polynomial-time simulator $\mathcal{S}$ such that

$$\begin{aligned}&\mathcal{S}\left(I, \left(x_{i_1}, \ldots, x_{i_t}\right), \left(f_{i_1}(\bar{x}), \ldots, f_{i_t}(\bar{x})\right)\right)\\ &\equiv \left(\text{view}_{i_1}^{\Pi}(\bar{x}), \ldots, \text{view}_{i_t}^{\Pi}(\bar{x})\right)\end{aligned};$$

2) the outputs of the ideal functionality and the protocol satisfy $\mathcal{F}(\bar{x}) \equiv \text{output}^{\Pi}(\bar{x})$.

2.3 Logistic Regression

LR is an efficient linear model using the logistic function for binary classification problems. The standard logistic function is a sigmoid function, which is defined as follows:

$$\sigma(x) = \frac{1}{1 + e^{-x}}. \tag{1}$$

It takes any input x and outputs a value between zero and one. Given a dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{n}$, the training objective of LR is to find a set of parameters $\theta \in \mathbb{R}^{m \times 1}$ that minimize the empirical loss function as follows:

$$\min_{\theta} f(\theta) := \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}\left(\sigma(\theta^{\top} x_i), y_i\right) + \lambda \sum_{j=1}^{m} g\left(\theta_j\right), \tag{2}$$

where $x_i \in \mathbb{R}^{m \times 1}$ is the feature vector of the i-th sample, $y_i \in \{0, 1\}$ is the true label of the i-th sample, $\theta^{\top} x_i = \sum_{j=1}^{m} \theta_j x_{i,j}$ is the linear function, $\mathcal{L}$ denotes the loss function, $\sum_{j=1}^{m} g\left(\theta_j\right)$ is the regularization term, and λ is a constant.

When training a LR model, there are many optimization algorithms that can be used, such as stochastic gradient descent (SGD), iterated Newton-Raphson, and L-BFGS. In this paper, we focus on the widely used SGD, which iteratively updates the model parameters in the following manner:

$$\theta_j^{t+1} = \theta_j^t - \alpha \nabla \mathcal{L}\left(\theta_j^t\right), \tag{3}$$

where θ_j^t is the j-th parameter of the t-th iteration, α is the learning rate, and $\nabla \mathcal{L}$ is the gradient of loss function. In general, the log loss is used to be the loss function, whose gradient is represented as $\nabla \mathcal{L}\left(\theta_j^t\right) = \frac{1}{n} \sum_{i=1}^{n} r_i^t x_{i,j}$, where $r_i^t = \hat{y}_i^t - y_i$ is the residual of the i-th sample in the t-iteration.

2.4 Secret Sharing

Secret sharing (SS) is a cryptographic technique, which is widely used as a building block in many secure protocols. A SS scheme involves a dealer who breaks a secret value into shares and distributes these shares to a group of parties. SPD$\mathbb{Z}_{2^k}$ [13, 14] is a commonly used SS framework, which securely computes an arithmetic circuit over an integer ring $\mathbb{Z}_{2^k}$. It has two kinds of SS schemes, including arithmetic SS and binary SS, which provide the ability to support arbitrary linear and non-linear operations, respectively. For the arithmetic SS, given $x \in \mathbb{Z}_{2^k}$, we denote by $\langle x \rangle^A$ the situation where the parties hold shares $\langle x \rangle_1^A, \ldots, \langle x \rangle_p^A \in \mathbb{Z}_{2^k}$, satisfying $x \equiv \sum_j \langle x \rangle_j^A (mod\ 2^k)$. For the binary SS, given $x \in \mathbb{Z}_2$, we denote by $\langle x \rangle^B$ the situation where the parties hold shares $\langle x \rangle_1^B, \ldots, \langle x \rangle_p^B \in \mathbb{Z}_2$, satisfying $x \equiv \bigoplus_j \langle x \rangle_j^B\ (mod\ 2)$. Moreover, to enable hybrid computations under two kinds of SS schemes, SPD$\mathbb{Z}_{2^k}$ supports the conversion between them. The frequently used primitives in SPD$\mathbb{Z}_{2^k}$ are shown as follows:

(1) **Addition** $\langle z \rangle^A = \langle x \rangle^A + \langle y \rangle^A$. P_i computes $\langle z \rangle_i^A \equiv \langle x \rangle_i^A + \langle y \rangle_i^A (mod\ 2^k)$ locally. If a certain party wants to reconstruct a secret z, the remaining parties send their shared values of z to that party, and the secret $z \equiv \sum_i \langle z \rangle_i^A (mod\ 2^k)$ is constructed.

(2) **Multiplication** $\langle z \rangle^A = \langle x \rangle^A \cdot \langle y \rangle^A$. The multiplication of two shared values relies on Beaver triples [3], which is generated by oblivious transfer [13, 30] or homomorphic encryption [15, 31] in the offline phase. Given a triplet $(\langle u \rangle^A, \langle v \rangle^A, \langle w \rangle^A)$, where u, v are random values in $\mathbb{Z}_{2^k}$ and $w \equiv uv(mod\ 2^k)$, the multiplication between two shared values can be performed with one round of intersection in the online phase.

(3) **The Most Significant Bit Extraction** Π_{MSB} . The parties compute the shared values of the most significant bit of x. This primitive can be used to execute a secure comparison of two signed integers. Specifically, given $\langle x \rangle^A$ and $\langle y \rangle^A$, we first calculate $\langle x - y \rangle^A$, and then extract the most significant bit of $\langle x - y \rangle^A$ as the result of the comparison.

2.5 Threshold Fully Homomorphic Encryption

Threshold fully homomorphic encryption (Threshold FHE) allows multiple parties to evaluate arithmetic circuits directly on encrypted data, as opposed to having to decrypt the data first. In its scheme, each party holds a distinct secret key and engages in interactions to generate a unified public key. The unified public key is used for encryption, and the secret keys are used for distributed decryption. Threshold CKKS [12] is a commonly used threshold FHE scheme, which supports addition and a pre-determined number of multiplications on encrypted data. In particular, the pre-determined number of multiplications is commonly called multiplicative depth and has an inverse relationship with the computational efficiency of the ciphertext operations. Threshold CKKS [1, 12] consists of the following primitives:

(1) **Key generation**. Each party first generates its own secret key sk_i and public key pk_i. Then, all parties exchange their public key for computing a unified public key $pk = (a, pk_1 + pk_2 + \ldots + pk_n)$, where a is a uniform ring element.

(2) **Encryption**. A given complex number x is first encoded to an integral polynomial $\mathcal{P} \leftarrow Ecd(x; \Delta)$, where Δ is a scaling factor, and then encrypted into a ciphertext $[\![x]\!] \leftarrow Enc_{pk}(\mathcal{P})$.

(3) **Decryption**. A ciphertext $[\![x]\!]$ is first decrypted into an integral polynomial $\mathcal{P}' \leftarrow Dec_{sk}([\![x]\!])$, and then decoded to the corresponding complex number $x \leftarrow Dcd(\mathcal{P}'; \Delta)$.

(4) **Operations**. Given two ciphertexts $[\![x]\!]$ and $[\![y]\!]$, there are two types of arithmetic operations, **Addititon**: $[\![x]\!] + [\![y]\!] = [\![x+y]\!]$, and **Multiplication**: $[\![x]\!] \cdot [\![y]\!] = [\![x \cdot y]\!]$. To improve computational efficiency, FHE supports the Single-Instruction Multiple-Data

(SIMD) operations, which pack a vector of N elements into a ciphertext by assigning each element to a slot within the ciphertext, and perform a single homomorphic operation on all elements of this vector in parallel. In particular, the number of slots is half of the degree of integral polynomials.

3 ISS-VLR

In this section, we first elaborate *ISS-VLR* for model training and inference in Sections 3.1-3.4. Then, we give its efficiency and privacy analysis in Section 3.5.

3.1 Overview

We present an efficient **V**ertical **LR** protocol with **I**nfinite **S**olution **S**ecurity (*ISS-VLR*). The training and inference processes of *ISS-VLR* are described as follows:

Training: During forward computation, the active party gathers the local linear-functional outputs from other parties and evaluates the sigmoid function to obtain the model outputs. To make the exposure of local linear-functional outputs satisfy infinite solution security, we establish a security constraint on the number of epochs (See details in Section 3.2). For backward updating, we propose a **R**andom **M**asks-based privacy-preserving **SGD** (*RM-SGD*) to securely and efficiently perform the gradient calculation and model parameter update (See details in Section 3.3). When model training is stopped, each party obtains the masked model parameters that correspond to their input features.

Inference: The inference process is the same as the process of forward computation in model training. We establish a security constraint on the number of continuous features for providing infinite solution security (See details in Section 3.2).

3.2 Security Constraints

To ensure that the exposure of linear-functional outputs satisfies infinite solution security, existing protocols [21, 43, 47–49] formalize the linear-functional outputs as a system of equations and derive a security constraint under which this system is underdetermined, i.e., there are more unknown variables than equations, as the underdetermined system has infinitely many solutions [18]. However, these protocols ignore two pieces of information accessible to the parties engaged in collaborative modeling, which might invalidate the stated security constraints. On the one hand, the active party may request the value ranges of input features from the passive parties to assess whether these input features are suitable for collaborative modeling. On the other hand, since the model parameters are updated iteratively during the training process, the parties are aware of the linear correlation of model parameters across multiple iterations. These two pieces of information are essentially additional restrictions on the variables in the system of equations constructed by the linear-functional outputs. Therefore, by leveraging the above information, the party receiving the linear-functional outputs may solve the unique solution of this system of equations.

To address this issue, we establish rigorous security constraints by formalizing these two pieces of information as equations and adding these equations into the system of equations constructed by the linear-functional outputs. In the following, we describe the details of the security constraints that provide infinite solution security during model training and inference, respectively.

Training. The active party can receive the local linear-functional outputs during model training. To establish a security constraint for making such exposure satisfy infinite solution security, we first formalize the local linear-functional outputs as the following nonlinear system of equations $\mathcal{S}$:

$$\begin{cases} \theta_1^1 x_{B_1[1],1} + \cdots + \theta_m^1 x_{B_1[1],m} = z_{B_1[1]} \\ \cdots \\ \theta_1^1 x_{B_1[b],1} + \cdots + \theta_m^1 x_{B_1[b],m} = z_{B_1[b]} \\ \cdots \\ \theta_1^T x_{B_T[1],1} + \cdots + \theta_m^T x_{B_T[1],m} = z_{B_T[1]} \\ \cdots \\ \theta_1^T x_{B_T[b],1} + \cdots + \theta_m^T x_{B_T[b],m} = z_{B_T[b]} \end{cases}, \tag{4}$$

where n is the number of samples, m is the number of features, T is the number of iterations, $\{x_{i,j} \mid i \in [1,n], j \in [1,m]\}$ are the input features, $\{\theta_j^i \mid i \in [1,T], j \in [1,m]\}$ are the model parameters, B_t is the t-th batch, b is the batch size, and $\{z_{B_t[i]} \mid t \in [1,T], i \in [1,b]\}$ are the values of linear-functional outputs.

We then formalize the value ranges of input features as equations. Without loss of generality, suppose there are discrete and continuous features, the value ranges of the corresponding unknown variable x is a set or an interval, respectively. The set F_d can be formalized as $\prod_{a \in F_d}(x-a) = 0$. For the interval, it can take one of four forms: $x \geq a$, $x > a$, $x \leq a$, and $x < a$, where a is the endpoint. By introducing an artificial variable y, these forms can be formalized as $y^2 = x - a$, $e^y = x - a$, $y^2 = a - x$, and $e^y = a - x$, respectively. These equations can be directly added into $\mathcal{S}$ to get a new system of equations $\mathcal{S}'$. Note that, the value ranges of other variables in $\mathcal{S}$, such as model parameters, are set by default to the real domain.

Next, we formalize the correlation of model parameters across multiple iterations as equations. Starting with the correlation between the model parameters of two successive iterations, it can be formalized as Equation (3), where the model parameters of the t-th iteration θ_j^t can be computed by the gradients and model parameters of the previous iteration. By analogy, the correlation between θ_j^t and the initial model parameters θ_j^1 can be formalized as follows:

$$\theta_j^t = \theta_j^1 - \frac{\alpha}{b} \sum_{i=1}^{t-1} \sum_{k=1}^{b} r_{B_i[k]} x_{B_i[k],j}, \tag{5}$$

where $r_{B_i[k]}$ is the residual of the k-th sample in the i-th batch and α is the public learning rate. By adding Equation (5) into $\mathcal{S}'$, all the model parameters, except for the initial model parameters, can be eliminated.

Finally, we establish a security constraint on the number of epochs e, such that e is less than the number of continuous features m_c, by making $\mathcal{S}'$ underdetermined, as shown in Theorem 3.1.

Notice that, if the active party is not aware of the value ranges of input features, the security constraint on the number of epochs can be relaxed to $e < m$, where m is the number of features. The proof of this security constraint follows a similar argument as Theorem 3.1 and is omitted due to space limitations.

Theorem 3.1. *If the number of epochs e is less than the number of continuous features m_c, i.e., $e < m_c$, $\mathcal{S}'$ is underdetermined.*

Proof. Let n be the number of samples, m be the number of features, m_d be the number of discrete features, F_c be the set of continuous features, ψ be a function used to count the number of intervals in F_c, m_c be the size of F_c, and e be the number of epochs. We start by counting the number of unknown variables and equations in $\mathcal{S}'$. Suppose the training-related hyper-parameters, including the learning rate, batch size, and number of iterations, are public and constant, the variables present in $\mathcal{S}'$ contain the input features, initial model parameters, residuals, and artificial variables introduced by considering the value ranges of input features. For the active party who receives the local linear-functional outputs, only the residuals are known among the above variables as the active party is allowed to obtain the residuals during training. Therefore, the unknown variables in $\mathcal{S}'$ include the input features, the initial model parameters, and the artificial variables. The total number of unknown variables is $nm + m + n\psi(F_c)$. For the equations in $\mathcal{S}'$, they contain not only the equations formed by the linear-functional outputs but also the equations introduced by considering the value ranges of input features. The total number of equations is $ne + nm_d + n\psi(F_c)$. To ensure that $\mathcal{S}'$ is underdetermined, it is necessary to have a greater number of unknown variables than the number of equations. Consequently, we can establish a constraint on the number of epochs, such that $e < m - m_d + \frac{m}{n} = m_c + \frac{m}{n}$. In practice, n is generally much larger than m, resulting in $\frac{m}{n}$ approaching 0, and $e < m_c$. In other words, if the number of epochs is less than the number of continuous features, $\mathcal{S}'$ is underdetermined. □

Inference.The active party gathers the local linear-functional outputs to compute model outputs during inference. To establish a security constraint for making such exposure satisfy infinite solution security, we first formalize the local linear-functional outputs as the following nonlinear system of equations $\mathcal{S}_o$:

$$\begin{cases} \theta_1 x_{1,1} + \theta_2 x_{1,2} + \cdots + \theta_m x_{1,m} = z_1 \\ \theta_1 x_{2,1} + \theta_2 x_{2,2} + \cdots + \theta_m x_{2,m} = z_2 \\ \cdots \\ \theta_1 x_{n,1} + \theta_2 x_{n,2} + \cdots + \theta_m x_{n,m} = z_n \end{cases}, \tag{6}$$

where $\{x_{i,j} \mid i \in [1,n], j \in [1,m]\}$ are the input features, $\{\theta_j\}_{j=1}^{m}$ are the model parameters, and $\{z_i\}_{i=1}^{n}$ are the values of linear-functional outputs. Then, we formalize the value ranges of input features as equations and add these equations into $\mathcal{S}_o$ to obtain a new system of equations $\mathcal{S}_o'$. Finally, we establish a security constraint on the number of continuous features m_c, such that m_c is more than one, by making $\mathcal{S}_o'$ underdetermined, as shown in Theorem 3.2. It is worth noting that, if the active party is not aware of the value ranges of input features, there is no need to impose any security constraints as $\mathcal{S}_o$ is always underdetermined.

Theorem 3.2. *If the number of continuous features m_c is more than 1, i.e., $m_c > 1$, $\mathcal{S}_o'$ is underdetermined.*

Proof. Let n be the number of samples, m be the number of features, m_d be the number of discrete features, F_c be the set of continuous features, ψ be a function used to count the number of intervals in F_c, and m_c be the size of F_c. We start by counting the number of unknown variables and equations in $\mathcal{S}_o'$. For the active party who receives the local linear-functional outputs, all variables in $\mathcal{S}_o'$ are unknown, including input features, model parameters, and artificial variables introduced by considering the value ranges of input features. Therefore, the total number of unknown variables is $nm + m + n\psi(F_c)$. For the equations in $\mathcal{S}_o'$, they contain not only the equations formed by the linear-functional outputs but also the equations introduced by considering the value ranges of input features. The total number of equations is $n + nm_d + n\psi(F_c)$. To ensure that $\mathcal{S}_o'$ is underdetermined, it is necessary to have a greater number of unknown variables than the number of equations. Consequently, we can establish a constraint on the number of continuous features, such that $m_c > 1 - \frac{m}{n} \approx 1$. In other words, if the number of continuous features is more than one, $\mathcal{S}_o'$ is underdetermined. □

3.3 Backward Updating with Random Masks

During backward updating, existing protocols [21, 43, 47] employ public-key cryptography techniques, which can privode provable security, resulting in significant computational overhead. Considering that forward computation is carried out with infinite solution security, according to the "barrel theory", we argue that providing infinite solution security for backward updating is sufficient. Therefore, we propose a **R**andom **M**ask-based privacy-preserving **SGD** (*RM-SGD*), which guarantees infinite solution security for backward updating by adding or multiplying a non-zero random mask. In particular, *RM-SGD* exploits the following homomorphic properties of random masks to compute the gradients and update the model parameters: 1) **Addition**: If a secret x is added to a random mask ε, we can perform the addition operation between $x+\varepsilon$ and an unmasked data y as ε can be eliminated from $x+y+\varepsilon$ by subtraction; 2) **Multiplication**: If a secret x is multiplied by a random mask ε, we can perform the multiplication operation between $x \cdot \varepsilon$ and an unmasked data y as ε can be eliminated from $x \cdot y \cdot \varepsilon$ by division; 3) **Inner Product**: If a secret vector v is multiplied by a random mask ε, we can perform the inner product operation between $v \cdot \varepsilon$ and an unmasked vector w as ε can be eliminated from $v \cdot w \cdot \varepsilon$ by division. Since no public-key cryptography techniques are used, *RM-SGD* can achieve high computational efficiency. Specifically, during the backward updating stage of each iteration, *RM-SGD* works as follows:

Step 1. Calculating Gradients. The active party uses the true labels and model outputs to calculate the residuals and further calculate the gradients corresponding to its input features. To enable the passive parties to perform the gradient calculation, involving an inner product, without exposing the residuals, the active party multiplies all the residuals by an identical random mask. Specifically, the active party P_1 first randomly generates a mask $\sigma^t \in \mathbb{R}$ and multiples the set of all residuals r^t by σ^t. Then, P_1 sends the masked residuals $\sigma^t r^t$ to the passive parties. Finally, each passive party P_i can compute the masked gradients $\mathcal{G}_i^t = \sigma^t G_i^t$, where G_i^t is the set of the gradients.

Step 2. Updating Model Parameters. To enable passive parties to update model parameters using $\mathcal{G}_i^t$, which involves a subtraction operation, the passive parties collaborate with the active party to convert the multiplicative mask in $\mathcal{G}_i^t$ to an additive mask without exposing the gradients to the active party. Specifically, each passive party P_i first generates a random matrix mask $K_i^t \in \mathbb{R}^{m_i \times m_i}$, multiples $\mathcal{G}_i^t$ by K_i^t, and sends $K_i^t \mathcal{G}_i^t$ to the active party P_1, where m_i is the number of features held by P_i. Then, P_1 divides $K_i^t \mathcal{G}_i^t$ by σ^t to obtain $K_i^t G_i^t$ and adds a new random vector mask $\mu_i^t \in \mathbb{R}^{m_i \times 1}$ to $K_i^t G_i^t$. Finally, after receiving $K_i^t G_i^t + \mu_i^t$, P_i updates θ_i^t as follows:

$$K_i^t \theta_i^{t+1} - \mu_i^t = K_i^t (\theta_i^t - G_i^t) - \mu_i^t. \tag{7}$$

Step 3. Post-processing of Model Parameters. The model parameters are not only used to calculate local linear-functional outputs, which involves a matrix-vector multiplication operation, but also required to be hidden from passive parties, as the model parameters can be used to invert the gradients. Therefore, the active party eliminates the additive mask in the model parameters and re-masks them by a multiplicative mask. Specifically, each passive party P_i sends $K_i^t \theta_i^{t+1} - \mu_i^t$ to the active party P_1. Then, P_1 eliminates μ_i^t, multiplies $K_i^t \theta_i^{t+1}$ by a random mask $\varphi_i^{t+1} \in \mathbb{R}$, and sends it to P_i. After eliminating K_i^t, P_i obtains $\varphi_i^{t+1} \theta_i^{t+1}$. Note that, before adding μ_i^{t+1} to $K_i^{t+1} G_i^{t+1}$ during the next iteration, P_1 multiplies $K_i^{t+1} G_i^{t+1}$ by φ_i^{t+1} for updating $\varphi_i^{t+1} \theta_i^{t+1}$.

3.4 Protocol Details

Protocol 1 shows the details of model training in *ISS-VLR*. In Protocol 1, each party starts by initializing the model parameters. Afterward, all parties engage in iterative training. For the t-th iteration, each passive party P_i first computes the local linear-functional outputs $\varphi_i^t \theta_i^t X_i^t$ and sends them to the active party P_1 (Lines 3-7). Then, after eliminating the random mask φ_i^t from $\varphi_i^t \theta_i^t X_i^t$ ($i \in [2, p]$), P_1 aggregates all local linear-functional outputs into the global linear-functional outputs and evaluates the sigmoid function to obtain the model outputs (Line 8). Next, P_1 computes the residuals and updates the model parameters (Lines 9-10). To enable other parties to update the model parameters without exposing the gradients and residuals, P_1 masks the residuals and interacts with other passive parties to compute the gradients and update the model parameters with random masks (Lines 11-20). When the number of iterations reaches the upper limit T, the training is complete, and each party P_i obtains the masked model parameters $\varphi_i^T \theta_i^T$.

For the details of model inference in *ISS-VLR*, they follow the same process as forward computation during model training, i.e., Lines 3-8 in Protocol 1.

Protocol 1: Model Training in *ISS-VLR*

```
    Input: $\{X_i\}_{i=1}^p$: input features from each party; $\{y_i\}_{i=1}^n$: input labels from active party; $T$: the number of iterations; $\alpha$: learning rate
    Output: $\varphi_i^T \theta_i^T$: model parameters for $P_i$
 1  // $P_1$ is active party
 2  for $t \in [1, T]$ do
 3      for $i \in [2, p]$ do
 4          if $t = 1$ then
 5              $P_i$ computes $\theta_i^t X_i^t$ and sends it to $P_1$
 6          else
 7              $P_i$ computes $\varphi_i^t \theta_i^t X_i^t$ and sends it to $P_1$
 8      $P_1$ aggregates $\theta^t X^t = \sum_{i=1}^p \theta_i^t X_i^t$ and evaluates the sigmoid function to obtain $\hat{Y}^t$
 9      $P_1$ computes $r^t = \hat{Y}^t - Y^t$
10      $P_1$ computes the gradient $\frac{\partial L}{\partial \theta_1^t} = \frac{r^t X_1^t}{b}$ and updates the parameters $\theta_1^{t+1} = \theta_1^t - \alpha \frac{\partial L}{\partial \theta_1^t}$
11      $P_1$ randomly generates $\sigma^t$ and sends $\sigma^t r^t$ to passive parties
12      for $i \in [2, p]$ do
13          $P_i$ computes $\mathcal{G}_i^t = \sigma^t G_i^t = \frac{\sigma^t r^t X_i^t}{b}$
14          $P_i$ randomly generates $K_i^t$ and sends $K_i^t \mathcal{G}_i^t$ to $P_1$
15          $P_1$ computes $K_i^t G_i^t = \frac{K_i^t \mathcal{G}_i^t}{\sigma^t}$
16          $P_1$ randomly generates $\mu_i^t$ and sends $\alpha \varphi_i^t K_i^t G_i^t + \mu_i^t$ to $P_i$
17          $P_i$ computes $\varphi_i^t K_i^t \theta_i^{t+1} - \mu_i^t$ and sends it to $P_1$
18          $P_1$ eliminates $\varphi_i^t$, $\mu_i^t$ and gets $K_i^t \theta_i^{t+1}$
19          $P_1$ randomly generates $\varphi_i^{t+1}$ and sends $\varphi_i^{t+1} K_i^t \theta_i^{t+1}$ to $P_i$
20          $P_i$ eliminates $K_i^t$ and obtains $\varphi_i^{t+1} \theta_i^{t+1}$
```

3.5 Efficiency and Security Analysis

In this subsection, we give the efficiency and security analysis of Protocol 1, respectively.

3.5.1 Efficiency. Let m be the number of features, b be the batch size, and p be the number of parties. In Protocol 1, since public-key cryptography techniques are not employed, the computation performed locally by each party is inexpensive. The dominating term in the execution overhead of Protocol 1 is the frequent communication between the active party and passive parties. Therefore, we only analyze the communication complexity of Protocol 1. Specifically, during forward computation, each passive party sends $O(b)$ local linear-functional outputs to the active party. When performing backward updating, the active party interacts with each passive party in three rounds, which requires the transmission of $O(b+4m)$ intermediate results. Overall, the total communication complexity of each iteration is $O((2b + 4m)(p - 1))$.

3.5.2 Security. Following the definition of infinite solution security, we must guarantee that the system of equations constructed by any intermediate result in Protocol 1 has infinitely many solutions. Therefore, we have the following theorem:

Theorem 3.3. *If the number of epochs e satisfies $e < m_c^*$, where m_c^* is the minimum value of $\{m_{c,i} \mid i \in [2, p]\}$ and $m_{c,i}$ is the number of continuous features held by P_i, we say that Protocol 1 satisfies infinite solution security against a semi-honest adversary who can statically corrupt up to p-1 out of p parties.*

The proof of Theorem 3.3 is deferred to Appendix B.1 due to space limitations.

4 PS-VLR

In this section, we first elaborate *PS-VLR* for model training and inference in Sections 4.1-4.4. Then, we give its efficiency and privacy analysis in Section 4.5.

4.1 Overview

We present an efficient **V**ertical **LR** protocol with **P**rovable **S**ecurity (*PS-VLR*). The training and inference processes of *PS-VLR* are described as follows:

Training: All parties jointly perform both forward computation and backward updating by employing SS, where matrix triples [9, 40] are used to facilitate online matrix multiplication with low communication overhead and generated offline using threshold FHE. Moreover, since homomorphic matrix-vector multiplication is the efficiency bottleneck of matrix triple generation, we propose a **R**otation-free **S**IMD-based **M**atrix-**V**ector **M**ultiplication method (*RS-MVM*) to accelerate this operation (See details in Section 4.2). We further propose a **HY**brid **SIG**moid approximation method (*HY-SIG*) to approximately evaluate the sigmoid function under SS (See details in Section 4.3). When model training is stopped, the model parameters are stored among all parties in the form of secretly shared values.

Inference: All parties first jointly compute the linear-functional outputs and approximately evaluate the sigmoid function using *HY-SIG* to obtain the secretly shared model outputs. Then, the active collaborates with other parties to reconstruct the model outputs.

4.2 Rotation-free Matrix-Vector Multiplication

We start by briefly describing the details of matrix triple generation in the multi-party setting [9]. Given the secretly shared first two components of a matrix triple, denoted as $\langle U\rangle^A$ and $\langle V\rangle^A$, all parties jointly reconstruct the encryptions of U and V, denoted as $\llbracket U\rrbracket$ and $\llbracket V\rrbracket$. The active party P_1 proceeds to calculate $\llbracket Z\rrbracket$, which involves a homomorphic matrix-vector multiplication. After converting $\llbracket Z\rrbracket$ into secretly shared values, the secretly shared matrix triple $(\langle U\rangle^A, \langle V\rangle^A, \langle Z\rangle^A)$ is obtained. It can be seen that an efficient homomorphic matrix-vector multiplication is crucial for achieving high efficiency in matrix triple generation. SIMD operations can perform a single homomorphic operation on multiple encrypted data simultaneously, and thus effectively improve the computational efficiency of homomorphic operations. However, existing SIMD-based homomorphic matrix-vector multiplication methods [22, 29] rely on the rotation operation, which has significant computational overhead. To avoid the use of rotation operations, Huang et. al [28] propose a SIMD-free method called Cheetah, which computes the homomorphic matrix-vector multiplication via only a single homomorphic multiplication between two ciphertexts.

To utilize the benefits of SIMD operations while avoiding the costly rotation operations, we propose a **R**otation-free **S**IMD-based **M**atrix-**V**ector **M**ultiplication method (*RS-MVM*). Its main idea is to decompose matrix-vector multiplication into the product of each column of the matrix and each element of the vector, and the summation of all the products. These products and the summation can be efficiently computed by SIMD-based multiplication and addition, respectively. As a result, *RS-MVM* utilizes only SIMD-based multiplications and additions in a single matrix-vector multiplication without involving rotation operations. Moreover, if the number of the matrix's rows is smaller than the number of slots in a ciphertext, *RS-MVM* has the ability to perform multiple matrix-vector multiplications in parallel during a single execution by concatenating the involved matrices and vectors during encryption, thereby maximizing the parallel computing capabilities of SIMD operations. Specifically, given a matrix $A \in \mathbb{R}^{N_r\times N_c}$ and a vector $B \in \mathbb{R}^{N_c\times 1}$, as shown in Figure 1, *RS-MVM* consists of two steps:

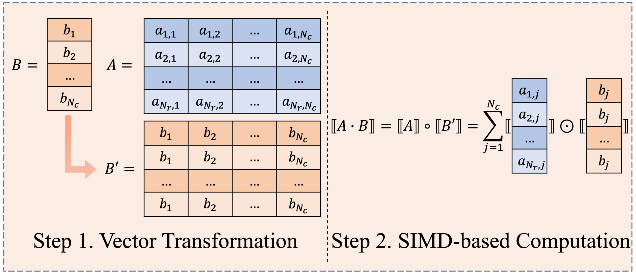

$B =$
b_1
b_2
...
b_{N_c}
$A =$
$a_{1,1}$ $a_{1,2}$ ... a_{1,N_c}
$a_{2,1}$ $a_{2,2}$ ... a_{2,N_c}
$a_{N_r,1}$ $a_{N_r,2}$ ... a_{N_r,N_c}
$B' =$
b_1 b_2 ... b_{N_c}
$\llbracket A\cdot B\rrbracket = \llbracket A\rrbracket \circ \llbracket B'\rrbracket = \sum_{j=1}^{N_c}$
$a_{1,j}$ $a_{2,j}$ ... $a_{N_r,j}$
$\odot$
b_j
Step 1. Vector Transformation
Step 2. SIMD-based Computation

Figure 1: Matrix-vector multiplication using *RS-MVM*

Step 1. Vector Transformation. Each element of B is first copied N_r times to obtain a replicated vector of dimension N_r. Then, all the replicated vectors are combined into a replicated matrix B' of dimension (N_r, N_c), in which the values of all elements in the i-th column are the same and equal to the value of the i-th element in B for $i \in [1, N_c]$.

Step 2. SIMD-based Computation. A and B' are first encrypted in a manner where each column is packed into a ciphertext to obtain $\llbracket A\rrbracket$ and $\llbracket B'\rrbracket$. Then, the result of the multiplication between A and B can be obtained by computing $\llbracket A\rrbracket \circ \llbracket B'\rrbracket$. In particular, the component-wise product between the j-th column of $\llbracket A\rrbracket$ and the j-th column of $\llbracket B'\rrbracket$ is calculated by the SIMD-based multiplication operation for $j \in [1, N_c]$, and all the products are aggregated by the SIMD-based addition operation.

If N_r is smaller than the number of slots in a ciphertext N_s, *RS-MVM* can perform parallel computation of multiple matrix-vector multiplications $A_1 \cdot B_1, \ldots, A_k \cdot B_k$, where $k \le N_s/N_r$. Specifically, all the vectors $\{B_i\}_{i=1}^k$ are first transformed into the replicated matrices $\{B'_i\}_{i=1}^k$. Then, the sets $\{A_i\}_{i=1}^k$ and $\{B'_i\}_{i=1}^k$ are concatenated into A^* and B^* along the row dimension, respectively. Next, A^* and B^* are encrypted and used to compute $\llbracket C\rrbracket = \llbracket A^*\rrbracket \circ \llbracket B^*\rrbracket$. After decrypting $\llbracket C\rrbracket$, the vector C can be split into k sub-vectors, and the i-th sub-vector is the result of $A_i \cdot B_i$. Otherwise, if $N_r > N_s$, the matrix A is sliced into $\zeta = \lceil \frac{N_r}{N_s} \rceil$ sub-matrices $A_1, \cdots, A_\zeta$ by rows, and the task of $A \cdot B$ is converted to subtasks of $A_1 \cdot B, \ldots, A_\zeta \cdot B$. Note that, since N_r is generally equal to either the batch size or the input feature dimension for model training and inference, we can leverage the parallel computing capability of *RS-MVM* to efficiently generate matrix triples.

4.3 Hybrid Sigmoid Approximation

The sigmoid function involves an exponential operation that is hard to support by SS. To approximately evaluate it, polynomial approximation methods [2, 7, 8] are commonly used, which employ a polynomial function to approximate the sigmoid function. However, these methods suffer from the unbounded tail problem, i.e., the polynomial output cannot be restricted to the range of $[0, 1]$. Existing studies [20, 40] solve this problem by combining the piecewise function with the polynomial function. However, they cannot achieve a favorable trade-off between model accuracy and evaluation efficiency. Specifically, SecureML [40] designs a function with three segments, where the output of the first and third segments are

0 and 1, respectively, and the second segment is a first-order linear function. Although this function can be efficiently evaluated using SS, its curve is not close to that of the sigmoid function. NFGen [20] constructs a function with 9 segments and polynomials of order 5, which is consistent with the sigmoid function in terms of function curves, while using SS to evaluate this function incurs significant communication and computational overheads.

To this end, we propose a **HY**brid **SIG**moid approximation method (*HY-SIG*). In *HY-SIG*, we carefully design a function with three segments for approximating the sigmoid function. The function's output is set to 1 and 0 in the first segment $(4, +\infty)$ and the third segment $(-\infty, -4)$, respectively. For the second segment $[-4, 4]$, we set a 3-order polynomial. This design is based on two observations. On the one hand, the output of the sigmoid function is close to 0 and 1 at $x = -4$ and $x = 4$, respectively. On the other hand, the 3-order polynomial function may have two local extremes, and the curvature of this function between these extremes is similar to that of the sigmoid function over the interval $[-4, 4]$. To obtain a 3-order polynomial function that is close to the sigmoid function in $[-4, 4]$, we choose four points $\{(-4, \sigma(-4)), (-2, \sigma(-2)), (2, \sigma(2)), (4, \sigma(4))\}$ as interpolation points, where σ denotes the sigmoid function, and employ Lagrangian interpolation to generate the polynomial function. As a result, the sigmoid function can be approximated by the following function:

$$\mathcal{H}(x) = \begin{cases} 1, x > 4 \\ 0.5 + 0.214x - 0.006x^3, -4 \leq x \leq 4 \\ 0, x < -4 \end{cases}. \tag{8}$$

To efficiently evaluate $\mathcal{H}$, we use two bits to indicate the interval in which the input is located and design an MPC-friendly linear expression with two bits to represent $\mathcal{H}$. Specifically, we first let $b_1 = 0$ if $x \geq -4$, otherwise $b_1 = 1$; $b_2 = 0$ if $x \geq 4$, otherwise $b_2 = 1$. Essentially, b_1 and b_2 are the most significant bits of $x + 4$ and $x - 4$, respectively. Then, $\mathcal{H}$ can be represented as $\mathcal{H}(u) = (1 - b_2) + b_2 (1 - b_1) u$, where u is the output of the 3-order polynomial function in the second segment. Therefore, the approximate results can be obtained by evaluating $\mathcal{H}$ using u and the significant bits of $x + 4$ and $x - 4$ under SS.

4.4 Protocol Details

In this subsection, we describe the details of model training, matrix triple generation, and model inference in *PS-VLR*, respectively.

Protocol 2 shows the details of model training in *PS-VLR*. Specifically, to begin with, all parties jointly initialize the secretly shared values of model parameters, compute the secretly shared values of masked input features $\langle E^t \rangle^A$, and reconstruct E^t for $t \in [1, T]$ (Lines 2-3). Afterward, all parties engage in iterative training. For the t-th iteration, during forward computation, all parties first jointly compute the secretly shared values of masked model parameters $\langle F_1^t \rangle^A$ and reconstruct F_1^t (Lines 5-6). Then, each party P_i computes the i-th shared values of linear-functional outputs $\langle \theta^t X^t \rangle_i^A$ (Line 7). Finally, all parties collaborate to approximately evaluate the sigmoid function and obtain the secretly shared model outputs (Line 8). During backward updating, all parties first jointly compute the secretly shared residuals $\langle r^t \rangle^A$ (Lines 9). Then, all parties jointly mask $\langle r^t \rangle^A$ to obtain $\langle F_2^t \rangle^A$, reconstruct F_2^t, and compute the secretly shared gradients (Lines 10-12). Finally, all parties jointly update the model parameters (Line 13). When the number of iterations reaches the upper limit T, the training is stopped, and the secretly shared model parameters $\langle \theta^T \rangle^A$ are output.

```
Protocol 2: Model Training in PS-VLR
  Input: $\{\langle X \rangle^A\}$: secretly shared input features; $\langle Y \rangle^A$:
         secretly shared input labels; $T$: the number of
         iterations; $\alpha$: learning rate;
         $\{\langle U^t \rangle^A, \langle V_1^t \rangle^A, \langle Z_1^t \rangle^A, \langle V_2^t \rangle^A, \langle Z_2^t \rangle^A\}_{t=1}^T$: secretly
         shared matrix triples
  Output: $\langle \theta^T \rangle^A$: secretly shared model parameters
1  // $P_1$ is active party
2  $P_i$ computes $\langle E^t \rangle_i^A = \langle X^t \rangle_i^A - \langle U^t \rangle_i^A$ for $i \in [1, p]$, $t \in [1, T]$
3  All parties jointly reconstruct $E^t$ for $t \in [1, T]$
4  for $t \in [1, T]$ do
5      $P_i$ computes $\langle F_1^t \rangle_i^A = \langle \theta^t \rangle_i^A - \langle V_1^t \rangle_i^A$ for $i \in [1, p]$
6      All parties jointly reconstruct $F_1^t$
7      $P_i$ computes
        $\langle \theta^t X^t \rangle_i^A = \mathcal{I}(i = 1) \cdot E^t \cdot F_1^t + \langle X^t \rangle_i^A \cdot F_1^t + E^t \cdot \langle \theta^t \rangle_i^A + \langle Z_1^t \rangle_i^A$,
        where $\mathcal{I}(\cdot)$ is indicator function, for $i \in [1, p]$
8      All parties jointly compute $\langle \hat{Y}^t \rangle^A$ using HY-SIG
9      $P_i$ computes $\langle r^t \rangle_i^A = \langle \hat{Y}^t \rangle_i^A - \langle Y^t \rangle_i^A$ for $i \in [1, p]$
10     $P_i$ computes $\langle F_2^t \rangle_i^A = \langle r^t \rangle_i^A - \langle V_2^t \rangle_i^A$ for $i \in [1, p]$
11     All parties jointly reconstruct $F_2^t$
12     $P_i$ computes $\langle \frac{\partial L}{\partial \theta^t} \rangle_i^A = \frac{1}{b}(\mathcal{I}(i = 1) \cdot (E^t)^\top \cdot F_2^t$
        $+ \langle (X^t)^\top \rangle_i^A \cdot F_2^t + (E^t)^\top \cdot \langle r^t \rangle_i^A + \langle Z_2^t \rangle_i^A)$, where $\mathcal{I}(\cdot)$
        is indicator function, for $i \in [1, p]$
13     $P_i$ computes $\langle \theta^{t+1} \rangle_i^A = \langle \theta^t \rangle_i^A - \alpha \cdot \langle \frac{\partial L}{\partial \theta^t} \rangle_i^A$
```

Protocol 3 shows the details of generating k matrix triples simultaneously in *PS-VLR*, where $k \leq \frac{N_s}{N_r}$, N_s is the number of slots in a ciphertext, and N_r is the row dimension of matrices in matrix triples. Specifically, each party P_i first randomly generates $\langle U_j \rangle_i^A \in \mathbb{R}^{N_r \times N_c}$ and $\langle V_j \rangle_i^A \in \mathbb{R}^{N_c \times 1}$, the i-th shared values of the matrix U_j and vector V_j, and transforms $\langle V_j \rangle_i^A$ into $\langle V_j' \rangle_i^A$ for $j \in [1, k]$ (Lines 2-3). Then, P_i concatenates all the $\langle U_j \rangle_i^A$ and $\langle V_j' \rangle_i^A$ into $\langle U^* \rangle_i^A$ and $\langle V^* \rangle_i^A$ along the row dimension, respectively (Line 4). P_i next encrypts $\langle U^* \rangle_i^A$ and $\langle V^* \rangle_i^A$, and sends them to P_1 (Line 5). Subsequently, P_1 reconstructs $[\![U^*]\!]$ and $[\![V^*]\!]$, and computes $[\![Z^*]\!] = [\![U^*]\!] \circ [\![V^*]\!]$ (Lines 6-7). To convert $[\![Z^*]\!]$ into secretly shared values, P_1 collects the random vector $c_i \in \mathbb{R}^{N_r \times 1}$ generated by P_i for $i \in [2, p]$, computes $[\![Z^* - \sum_{i=2}^p c_i]\!]$, and cooperates with other parties to decrypt it (Lines 8-9). After that, $Z^* - \sum_{i=2}^p c_i$, c_2, $\cdots$, and c_p are set as the secretly shared values of Z^* (Line 10). Finally, P_i splits $\langle Z^* \rangle_i^A$ into k sub-vectors, and the j-th sub-vector is the i-th shared values of the third component Z_j for the j-th matrix triple (U_j, V_j, Z_j) (Line 11).

For the details of model inference in *PS-VLR*, they follow a similar process as forward computation during model training, i.e., Lines 2-8 in Protocol 2. Specifically, all parties first jointly compute and reconstruct the masked input features E and the masked model parameters F. Then, all parties jointly compute the linear-functional outputs and approximately evaluate the sigmoid function to obtain the secretly shared model outputs. Finally, P_1 gathers the shared values of model outputs from other parties and reconstructs the model outputs.

```
Protocol 3: Generation of $k$ Matrix Triples
  Input: $N_r$: the number of rows; $N_c$: the number of columns;
         $k$: the number of matrix triples; $pk$: public key;
         $\{sk_i\}_{i=1}^{p}$: secret keys from each party
  Output: $\left\{(\langle U_j\rangle^A, \langle V_j\rangle^A, \langle Z_j\rangle^A)\right\}_{j=1}^{k}$: secretly shared matrix
          triples
for $i \in [1, p]$ do
    $P_i$ randomly generates $\langle U_j\rangle_i^A$ and $\langle V_j\rangle_i^A$ for $j \in [1, k]$
    $P_i$ transforms $\langle V_j\rangle_i^A$ into $\langle V_j'\rangle_i^A$ for $j \in [1, k]$
    $P_i$ concatenates $\left\{\langle U_j\rangle_i^A\right\}_{j=1}^{k}$ and $\left\{\langle V_j'\rangle_i^A\right\}_{j=1}^{k}$ into $\langle U^*\rangle_i^A$
      and $\langle V^*\rangle_i^A$ along the row dimension, respectively
    $P_i$ packs each column of $\langle U^*\rangle_i^A$ and $\langle V^*\rangle_i^A$ into a
      ciphertext and sends them to $P_1$
$P_1$ computes $[\![U^*[:, j]]\!] = \sum_{i=1}^{p}[\![\langle U^*[:, j]\rangle_i^A]\!]$ and
  $[\![V^*[:, j]]\!] = \sum_{i=1}^{p}[\![\langle V^*[:, j]\rangle_i^A]\!]$ for $j \in [1, N_c]$
$P_1$ computes $[\![Z^*]\!] = \sum_{j=1}^{N_c}[\![U^*[:, j]]\!] \odot [\![V^*[:, j]]\!]$
$P_i$ randomly generates $c_i$ and sends $[\![c_i]\!]$ to $P_1$ for $i \in [2, p]$
$P_1$ computes $[\![Z^* - \sum_{i=2}^{p} c_i]\!]$ and cooperates with other
  parties to decrypt $Z^* - \sum_{i=2}^{p} c_i$
$P_1$ sets $\langle Z^*\rangle_1^A = Z^* - \sum_{i=2}^{p} c_i$ and $P_i$ sets $\langle Z^*\rangle_i^A = c_i$ for
  $i \in [2, p]$
$P_i$ splits the $\langle Z^*\rangle_i^A$ into $k$ sub-vectors $\left\{\langle Z_j\rangle_i^A\right\}_{j=1}^{k}$ for
  $i \in [1, p]$.
```

4.5 Efficiency and Security Analysis

In this subsection, we give the efficiency and security analysis of Protocols 2-3, respectively.

4.5.1 Efficiency. Let n be the number of samples, m be the number of features, b be the batch size, p be the number of parties, e be the number of epochs, l be the bit length, N_r and N_c are respectively the row and column dimensions of matrices in matrix triples, and N_s is the number of slots in a ciphertext. The efficiency bottleneck of Protocol 2 stems from the frequent communication among involved parties, and the communication complexity of Protocol 2 is $O(((1+\frac{e}{b})m + (15 - \frac{8}{l})e)np(p-1))$. For Protocol 3, its computational and communication complexities are shown in Table 1.

Comparison with Beaver triples based protocols. For SS-based matrix multiplication in each iteration, the communication complexity of *PS-VLR* and Beaver triples based protocols are $O((b+m)p(p-1))$ and $O(4bmp(p-1))$, respectively. It can be seen that *PS-VLR* has less communication overhead. Moreover, *PS-VLR* has a higher efficiency on the offline triple generation in scenarios that require a large number of triples, such as LR model training and inference. The reason can be explained as follows. *PS-VLR* can simultaneously generate k matrix triples, enabling the execution of k matrix multiplications, where $k \leq \frac{N_s}{N_r}$. If using threshold FHE to generate an adequate number of Beaver triples for supporting k matrix multiplications, the computational and communication complexities are shown in Table 1. It can be observed that although the generation of matrix triples requires a slightly higher communication complexity and greater number of encryptions, homomorphic multiplications, and homomorphic additions, it significantly reduces the number of distributed decryptions.

4.5.2 Security. We prove that Protocol 2 and Protocol 3 satisfy the provable security by the real/ideal simulation paradigm [37]. This paradigm utilizes a simulator, denoted as S, to generate the adversary's views in the ideal world by executing the ideal functionalities. Then, these views are compared with the adversary's views generated by executing these protocols in the real world. If both views are consistent, these protocols satisfy provable security. We identify two ideal functionalities $\mathcal{F}_{Train}$ and $\mathcal{F}_{Triple}$ for Protocol 2 and Protocol 3, respectively, and have the following theorem:

Theorem 4.1. *Protocol 2 and Protocol 3 securely compute $\mathcal{F}_{Train}$ and $\mathcal{F}_{Triple}$ against a semi-honest adversary who can statically corrupt up to p-1 out of p parties, respectively.*

These ideal functionalities and the proof of Theorem 4.1 are given in Appendix B.2 due to space limitations.

5 EXPERIMENTS

In this section, we evaluate the performance of our proposed *ISS-VLR* and *PS-VLR*.

5.1 Experimental Setup

Datasets. We use four real datasets, including MNIST [34] and three UCI datasets [19], and one synthetic dataset in our experiments. For each real dataset, we first uniformly split the features into p partitions, which are held by p parties, and assign one party to hold the label set of datasets. By default, p is set to 3. Then, we perform one-hot encoding on the discrete features and max-min normalization on the continuous features. After preprocessing, the number of samples, the number of features, and the number of classes for each dataset are listed in Table 2. For the synthetic dataset, we randomly generate a batch of data for each party, which contains 64 samples with 40 features.

Hyper-parameters. The main hyperparameters for training LR models using SGD include batch size, learning rate, and number of epochs. For our experiments, we configure the batch size, learning rate, and number of parties as 64, 0.001, and 10, respectively.

We conduct experiments on many machines equipped with Intel Ice Lake 32 cores CPU and 32GB of RAM.

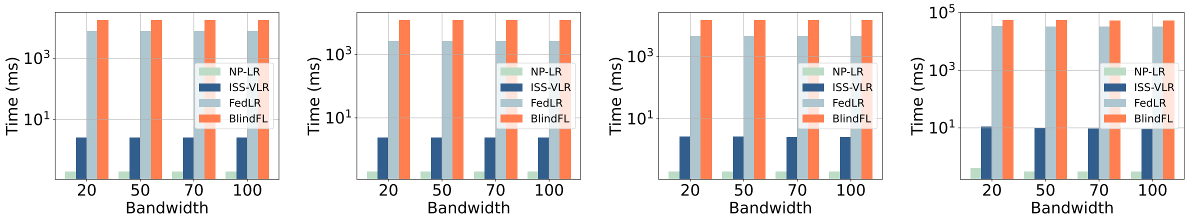
5.2 Performance of ISS-VLR

Since *ISS-VLR* can obtain the same model accuracy as the non-private LR as long as the model can converge within the maximum number of epochs specified by the security constraint (refer to Theorem 3.1), we only evaluate the efficiency of *ISS-VLR* in this subsection. All the protocols evaluated in this subsection are implemented using Python.

To illustrate the efficiency of *ISS-VLR*, we compare it with three baselines in terms of the running time of model training for one iteration: 1) NP-LR allows all parties to collaborate on the training without protecting any intermediate results; 2) FedLR [46] exposes the local linear-functional outputs during forward computation and uses the Paillier cryptosystem to protect the intermediate results

Table 1: The computational and communication complexities of generating triples

	Encryption	Addition	Multiplication	Distributed decryption	Communication
Matrix triples	$O(2pN_c)$	$O((2p-1)N_c-1)$	$O(N_c)$	$O(1)$	$O(2(p-1)N_c)$
Beaver triples	$O(2p\lceil\frac{N_rN_ck}{N_s}\rceil)$	$O(2(p-1)\lceil\frac{N_rN_ck}{N_s}\rceil))$	$O(\lceil\frac{N_rN_ck}{N_s}\rceil)$	$O(\lceil\frac{N_rN_ck}{N_s}\rceil)$	$O(2(p-1)\lceil\frac{N_rN_ck}{N_s}\rceil))$



(a) Adult **(b) Bank** **(c) Credit-card** **(d) MNIST**

Figure 2: Comparison of *ISS-VLR* and baselines on the running time (ms) of model training for one iteration

Table 2: Preprocessed datasets in our experiments

Dataset	Samples	Features	Classes
Adult	45222	104	2
Bank	45211	51	2
Credit-card	30000	91	2
MNIST	60000	784	10

during backward updating; 3) BlindFL [21] exposes the global linear-functional outputs during forward computation and performs the rest of the computations by SS and the Paillier cryptosystem. In this experiment, we implement the Paillier cryptosystem by Python-Paillier [17] and set the key size to 2048 bits. Figure 2 presents the comparison results under different bandwidths. As expected, we can observe that *ISS-VLR* is 10× slower than NP-LR, while it is about 1000× faster than FedLR [46] and BlindFL [21], respectively. This reason lies in that *ISS-VLR* does not use any public-key cryptography techniques, while FedLR [46] and BlindFL [21] employ the Paillier cryptosystem during backward updating.

Additionally, we also verify the scalability of *ISS-VLR* on the number of parties p. In this experiment, we fix the bandwidth to 70 Mbps and vary p from 3 to 6. Table 3 illustrates the training time of one iteration on the synthetic dataset with different settings of p. We can observe that the training time is consistently 2.8 ms as p grows. This result can be explained by the fact that the interaction between each passive party and the active party can be carried out independently and in parallel. Therefore, *ISS-VLR* has favorable scalability as the number of parties increases.

Table 3: Effect of the number of parties p

	p=3	p=4	p=5	p=6
ISS-VLR	2.8 ms	2.8 ms	2.8 ms	2.8 ms

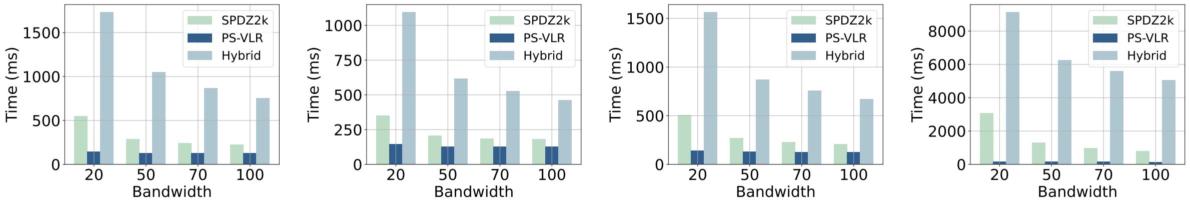
5.3 Performance of PS-VLR

In this subsection, we first evaluate the efficiency of *PS-VLR*. Then, we validate the effectiveness of *RS-MVM* for accelerating matrix triple generation. Finally, we validate the effectiveness and efficiency of *HY-SIG*. All the protocols evaluated in this subsection are implemented using C++.

5.3.1 Efficiency of PS-VLR. We compare *PS-VLR* with the following two baselines. The first one is the SS-based vertical LR protocol, which is implemented directly by the SPD$\mathbb{Z}_{2^k}$ framework [14] and referred to as SPDZ2k. The second one is the hybrid vertical LR protocol, which is the extension of the multi-party setting of CAESAR [7], a two-party protocol based on HE and SS, and referred to as HybridLR. For comparing fairness, the running time of the evaluation of the sigmoid function is not recorded, and the matrix triples and Beaver triples are both generated by threshold FHE. In particular, we use the threshold CKKS scheme in the PALISADE library [1] and set its three key parameters, namely multiplicativeDepth, scalingFactorBits, and batchSize, to 1, 50, and 4096, respectively. In addition, to support floating-point operations by SS, we convert the inputs from floating-point to fixed-point with a 128-bit length and a 12-bit decimal part.

Figure 3 and Table 4 respectively show the comparison results on the running time of model training and offline triple generation under different bandwidths. We can observe that *PS-VLR* has less running time than SPDZ2k [14] in both model training and offline triple generation, which is consistent with our efficiency analysis described in Section 4.5.1. We also observe that *PS-VLR* and SPDZ2k [14] take less time to perform model training than HybridLR. This is because HybridLR requires performing homomorphic computations and transformations between SS and HE during training, which incurs significant computational and communication overheads. Nevertheless, for the total running time of model training and offline triple generation, *PS-VLR* has a worse efficiency than HybridLR when the bandwidth is low as it has a higher communication overhead during offline triple generation. Note that, we also compare *PS-VLR* with HybridLR in terms of the overall training time, which additionally includes the running time of sharing input data, and evaluate the effect of various feature sparsities on the training efficiency of these two protocols. Due to space limitations, we show these experimental results and their analysis in Appendix A.

Additionally, we also verify the scalability of *PS-VLR* on the number of parties p. In this experiment, we fix the bandwidth to 70 Mbps and vary p from 3 to 6. Table 5 illustrates the running

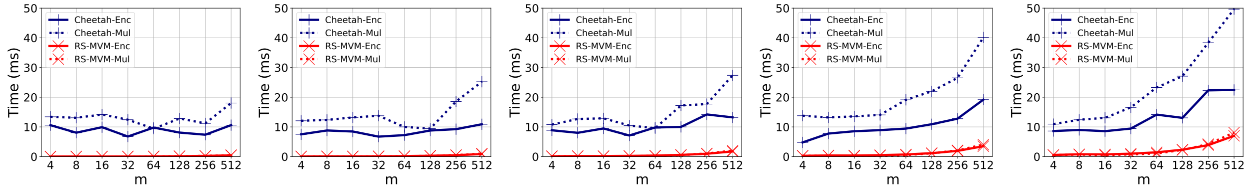


(a) Adult **(b) Bank** **(c) Credit-card** **(d) MNIST**

Figure 3: Comparison of *PS-VLR* and baselines on the running time (ms) of model training for one iteration

Table 4: Comparison of *PS-VLR* and SPDZ2k on the running time (ms) of offline triple generation for one iteration

	Adult				Bank				Credit-card				MINST			
Bandwidth	20	50	70	100	20	50	70	100	20	50	70	100	20	50	70	100
SPDZ2k	6049	2505	1826	1310	2957	1224	893	640	5267	2181	1590	1140	45652	18907	13779	9885
PS-VLR	2241	931	677	486	1107	460	336	240	1952	810	590	424	17924	7530	5472	3945



(a) $n = 16$ **(b)** $n = 32$ **(c)** $n = 64$ **(d)** $n = 128$ **(e)** $n = 256$

Figure 4: Comparison of *RS-MVM* and Cheetah [28] on the running time (ms) of encryptions (Enc) and multiplications (Mul) during matrix triple generation

Table 5: Effect of the number of parties p

	p=3	p=4	p=5	p=6
PS-VLR	112 ms	127 ms	152 ms	187 ms

time of model training for one iteration on the synthetic dataset with different settings of p. We can observe that the running time increases quadratically with the growth of p. This is because the communication complexity of SS is at least quadratic in p.

5.3.2 Effectiveness of RS-MVM. We compare the performance of *RS-MVM* against Cheetah [28], the state-of-the-art homomorphic matrix-vector multiplication method, on accelerating matrix triple generation. Since the open-source code of Cheetah [28] is implemented using the BFV scheme from the SEAL library [41], we adopt the same scheme and library to implement *RS-MVM* for comparing fairness and set the polynomial modulus to 4096. However, SEAL [41] does not support the threshold version of BFV. Therefore, we only measure the running time required for encrypting the matrix and vector, and performing the multiplication of their corresponding ciphertexts during the generation of a matrix triple. Let n and m be the numbers of rows and columns of the matrix, respectively. Figure 3 shows the results under different (n, m) settings. We can observe that *RS-MVM* outperforms Cheetah [28] in terms of the running time required for encryption and multiplication under any (n, m) settings. The reason lies in that *RS-MVM* can generate multiple matrix triplets simultaneously through a single matrix-vector multiplication. We also observe that the discrepancy in efficiency between *RS-MVM* and Cheetah [28] widens as n and m increase. This is because Cheetah [28] requires to divide the matrix and the vector into a large number of sub-matrices and sub-vectors when nm is larger than the number of slots in a ciphertext, leading to a superlinear growth in the number of encryptions and multiplications as n and m increase. In contrast, *RS-MVM* solely necessitates $O(m)$ encryptions and multiplications while displaying insensitivity towards variations in n.

Table 6: Comparison of various approximation methods

	Sigmoid	SecureML	NFGen	HY-SIG
Adult	0.8757	0.8666	0.8757	0.8752
Bank	0.7869	0.8793	0.7869	0.8006
Credit-card	0.7390	0.7242	0.7390	0.7399
MNIST	0.9862	0.9753	0.9862	0.9854
Running time on one batch	-	71 ms	77 ms	109 ms

5.3.3 Effectiveness and Efficiency of HY-SIG. We compare the hybrid sigmoid approximation method with three baselines, namely the original sigmoid, SecureML [40], and NFGen [20]. For comparing fairness, we use SPD$\mathbb{Z}_{2^k}$ [14] to securely evaluate the approximation functions proposed by these methods. In this experiment, we set two metrics: 1) the Area Under the ROC Curve (AUC) of models trained by *HY-SIG* and these baselines; 2) the running time on one batch of input data with a batch size of 64. Table 6 illustrates the comparison results. We can observe that *HY-SIG* and NFGen [20] can achieve the same model accuracy as the original sigmoid. In contrast, SecureML [40] may cause a loss of model accuracy. This result can be attributed to the fact that the curve of the function proposed by SecureML [40] differs considerably from that of the sigmoid function, as opposed to the curves of the functions proposed by NFGen [20] and *HY-SIG*. We also observe that NFGen [20] has more running time than SecureML [40] and *HY-SIG*. This is because NFGen [20] leverages more segments and orders to approximately the sigmoid function. In summary, *HY-SIG* can obtain a lossless model accuracy with less running time, providing a favorable trade-off between efficiency and model accuracy.

6 RELATED WORK

In this section, we give literature reviews on privacy-preserving vertical LR protocols with infinite solution security and provable security, respectively.

6.1 Protocols with Infinite Solution Security

Existing protocols with infinite solution security [21, 43, 47–49] expose linear-functional outputs during forward computation and prove that such exposure satisfies infinite solution security through matrix decomposition or system of equations. Yang et.al [47] assign the active party to collect and aggregate all local linear-functional outputs held by passive parties to obtain the global linear-functional outputs. Several studies [21, 43, 48, 49] argue that the exposure of local linear-functional outputs has the potential risk of privacy leakage, and use cryptography techniques to securely aggregate the local linear-functional outputs. Specifically, Zhang et.al [49], Fu et.al [21], and Zhang et.al [48] aggregate the local linear-functional outputs using SS. Xu et.al [43] exploit the additive homomorphism property of function encryption to achieve secure aggregation. However, after reviewing the process of collaborative LR model building, we found that all the above studies ignore two pieces of information available to collaborators, which may result in the failure to provide infinite solution security.

During backward updating, several studies [48, 49] directly perform non-private gradient calculations and model parameter updates for high efficiency. However, the exposure of residuals results in privacy leakage, as the signs of the residuals can directly reflect the true labels. To address this issue, Xu et.al [43] use the function encryption technique to implement gradient calculation without exposing residuals. However, the gradients are still exposed to the active party, who can exploit them to reconstruct the features held by passive parties [16, 36, 50]. Yang et.al [47] avoid this risk by combining HE with random masks. However, the passive parties can obtain the gradients for updating the model parameters, which may lead to the leakage of residuals. For instance, suppose the passive party holds a binary feature and there exists a batch of samples whose values of this feature is $\{1, 0, 0\}$, the passive party can determine that the gradient value corresponding to this binary feature is equal to the residual value of the first sample. Therefore, Fu et.al [21] adopt SS and HE to securely compute gradients and update model parameters, while ensuring the non-disclosure of any intermediate results. However, according to the "barrel theory", we argue that providing infinite solution security for backward updating is sufficient, and the usage of public-key cryptography techniques can be avoided.

6.2 Protocols with Provable Security

Existing protocols with provable security use cryptography techniques, such as HE and MPC, to prevent the disclosure of any intermediate results.

Most HE-based protocols are applied in centralized LR modeling [2, 8, 22, 24, 25, 27, 32, 44]. That is, data owners send their encrypted data to a central server who performs the model training using the encrypted data. However, these protocols necessitate a substantial allocation of computing resources for the central server to support costly ciphertext computations. Hardy et.al [26] and Yang et.al [45] employ HE to perform distributed LR modeling, where the data owners collaborate to build models. However, they rely on a third non-colluding server for assistance. In real-world scenarios, it is difficult to find an authoritative third party that can be trusted by all involved data owners.

Most MPC-based protocols use SS to perform LR model training and inference. Several studies [35, 38–40, 42] set up many non-colluding servers and require data owners to secretly share their vertically partitioned data with these servers for building models. For instance, Mohassel et.al [40] propose SecureML, which sets two non-colluding servers and utilizes matrix triples to reduce the communication overhead. However, the setting of many non-colluding servers can present practical challenges. Other studies [11, 13–15, 30, 31] secretly share the data among the data owners, who jointly perform LR model training and inference. SPD$\mathbb{Z}_{2^k}$ [14] is one of the representatives, which designs many primitives over the ring of integers modulo 2^k to support the collaborative evaluation of arbitrary arithmetic circuits by all data owners.

Recently, Chen et.al [7] propose CAESAR, a hybrid two-party protocol based on HE and SS, which employs HE and SS to perform secure matrix multiplication and the remaining computations, respectively. For high-dimensional and sparse datasets, CAESAR [7] can improve the computational efficiency of secure matrix multiplication by exploiting sparse computation. Nevertheless, CAESAR [7] is limited to the two-party setting and has large computational and communication overheads introduced by the transformation between SS and HE.

7 CONCLUSION

In this paper, we present two privacy-preserving vertical LR protocols, namely *ISS-VLR* and *PS-VLR*, which guarantee infinite solution security and provable security, respectively. In *ISS-VLR*, we first establish rigorous security constraints to make the exposure of linear-functional outputs satisfy infinite solution security. Then, we propose *RM-SGD* to efficiently perform backward updating while preserving privacy. In *PS-VLR*, we employ SS to perform model training and inference and utilize matrix triples to reduce communication overhead. Moreover, we propose *RS-MVM* to improve the efficiency of matrix triple generation. We further propose *HY-SIG* to effectively and efficiently approximate the sigmoid function under SS. Experimental results demonstrate the effectiveness and efficiency of the proposed protocols.

REFERENCES

[1] 2021. PALISADE Lattice Cryptography Library (release 1.11.3). https://palisade-crypto.org/.

[2] Yoshinori Aono, Takuya Hayashi, Le Trieu Phong, and Lihua Wang. 2016. Scalable and secure logistic regression via homomorphic encryption. In *ACM Conference on Data and Application Security and Privacy*. 142–144.

[3] Donald Beaver. 1991. Efficient multiparty protocols using circuit randomization. In *International Cryptology Conference*. 420–432.

[4] Michael JA Berry and Gordon S Linoff. 2004. *Data mining techniques: for marketing, sales, and customer relationship management*. John Wiley & Sons.

[5] Carl R Boyd, Mary Ann Tolson, and Wayne S Copes. 1987. Evaluating trauma care: the TRISS method. Trauma Score and the Injury Severity Score. *Trauma Score and the Injury Severity Score* 27, 4 (1987), 370–378.

[6] Melissa Chase and Peihan Miao. 2020. Private set intersection in the internet setting from lightweight oblivious PRF. In *International Cryptology Conference*. 34–63.

[7] Chaochao Chen, Jun Zhou, Li Wang, Xibin Wu, Wenjing Fang, Jin Tan, Lei Wang, Alex X Liu, Hao Wang, and Cheng Hong. 2021. When homomorphic encryption marries secret sharing: Secure large-scale sparse logistic regression and applications in risk control. In *ACM Conference on Knowledge Discovery & Data Mining*. 2652–2662.

[8] Hao Chen, Ran Gilad-Bachrach, Kyoohyung Han, Zhicong Huang, Amir Jalali, Kim Laine, and Kristin Lauter. 2018. Logistic regression over encrypted data from fully homomorphic encryption. *BMC medical genomics* 11, 4 (2018), 3–12.

[9] Hao Chen, Miran Kim, Ilya Razenshteyn, Dragos Rotaru, Yongsoo Song, and Sameer Wagh. 2020. Maliciously secure matrix multiplication with applications to private deep learning. In *Annual International Conference on the Theory and Application of Cryptology and Information Security*. 31–59.

[10] Hao Chen, Kim Laine, and Peter Rindal. 2017. Fast private set intersection from homomorphic encryption. In *ACM Conference on Computer and Communications Security*. 1243–1255.

[11] Valerie Chen, Valerio Pastro, and Mariana Raykova. 2019. Secure computation for machine learning with SPDZ. *arXiv preprint arXiv:1901.00329* (2019).

[12] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. 2017. Homomorphic encryption for arithmetic of approximate numbers. In *Annual International Conference on the Theory and Application of Cryptology and Information Security*. 409–437.

[13] Ronald Cramer, Ivan Damgård, Daniel Escudero, Peter Scholl, and Chaoping Xing. 2018. SPDZ_{2k}: efficient MPC mod 2^k for dishonest majority. In *International Cryptology Conference*. 769–798.

[14] Ivan Damgård, Daniel Escudero, Tore Frederiksen, Marcel Keller, Peter Scholl, and Nikolaj Volgushev. 2019. New primitives for actively-secure MPC over rings with applications to private machine learning. In *IEEE Symposium on Security and Privacy*. 1102–1120.

[15] Ivan Damgård, Valerio Pastro, Nigel Smart, and Sarah Zakarias. 2012. Multiparty computation from somewhat homomorphic encryption. In *Annual Cryptology Conference*. 643–662.

[16] Trung Dang, Om Thakkar, Swaroop Ramaswamy, Rajiv Mathews, Peter Chin, and Françoise Beaufays. 2021. Revealing and Protecting Labels in Distributed Training. *Advances in Neural Information Processing Systems* 34 (2021).

[17] CSIRO's Data61. 2013. Python Paillier Library. https://github.com/data61/python-paillier.

[18] David Donoho, Hossein Kakavand, and James Mammen. 2006. The simplest solution to an underdetermined system of linear equations. In *IEEE International Symposium on Information Theory*. 1924–1928.

[19] Dheeru Dua and Casey Graff. 2017. UCI Machine Learning Repository. http://archive.ics.uci.edu/ml

[20] Xiaoyu Fan, Kun Chen, Guosai Wang, Mingchun Zhuang, Yi Li, and Wei Xu. 2022. NFGen: Automatic Non-linear Function Evaluation Code Generator for General-purpose MPC Platforms. In *ACM Conference on Computer and Communications Security*. 995–1008.

[21] Fangcheng Fu, Huanran Xue, Yong Cheng, Yangyu Tao, and Bin Cui. 2022. BlindFL: Vertical Federated Machine Learning without Peeking into Your Data. In *ACM International Conference on Management of Data*. 1316–1330.

[22] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. 2016. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *ACM International Conference on Machine Learning*. 201–210.

[23] Oded Goldreich. 2009. *Foundations of cryptography: volume 2, basic applications*. Cambridge university press.

[24] Kyoohyung Han, Seungwan Hong, Jung Hee Cheon, and Daejun Park. 2019. Logistic regression on homomorphic encrypted data at scale. In *AAAI Conference on Artificial Intelligence*. 9466–9471.

[25] Kyoohyung Han, Jinhyuck Jeong, Jung Hoon Sohn, and Yongha Son. 2020. Efficient privacy preserving logistic regression inference and training. *Cryptology ePrint Archive* (2020).

[26] Stephen Hardy, Wilko Henecka, Hamish Ivey-Law, Richard Nock, Giorgio Patrini, Guillaume Smith, and Brian Thorne. 2017. Private federated learning on vertically partitioned data via entity resolution and additively homomorphic encryption. *arXiv preprint arXiv:1711.10677* (2017).

[27] Ehsan Hesamifard, Hassan Takabi, and Mehdi Ghasemi. 2017. Cryptodl: Deep neural networks over encrypted data. *arXiv preprint arXiv:1711.05189* (2017).

[28] Zhicong Huang, Wen-jie Lu, Cheng Hong, and Jiansheng Ding. 2022. Cheetah: Lean and Fast Secure {Two-Party} Deep Neural Network Inference. In *USENIX Security Symposium*. 809–826.

[29] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. 2018. GAZELLE: A low latency framework for secure neural network inference. In *USENIX Security Symposium*. 1651–1669.

[30] Marcel Keller, Emmanuela Orsini, and Peter Scholl. 2016. MASCOT: faster malicious arithmetic secure computation with oblivious transfer. In *ACM Conference on Computer and Communications Security*. 830–842.

[31] Marcel Keller, Valerio Pastro, and Dragos Rotaru. 2018. Overdrive: making SPDZ great again. In *International Conference on the Theory and Applications of Cryptographic Techniques*. 158–189.

[32] Miran Kim, Yongsoo Song, Shuang Wang, Yuhou Xia, Xiaoqian Jiang, et al. 2018. Secure logistic regression based on homomorphic encryption: Design and evaluation. *JMIR Medical Informatics* 6, 2 (2018), e8805.

[33] Rohit Kumar, Sneha Manjunath Naik, Vani D Naik, Smita Shiralli, VG Sunil, and Moula Husain. 2015. Predicting clicks: CTR estimation of advertisements using logistic regression classifier. In *IEEE International Advance Computing Conference*. 1134–1138.

[34] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

[35] Yi Li, Yitao Duan, Yu Yu, Shuoyao Zhao, and Wei Xu. 2018. PrivPy: Enabling Scalable and General Privacy-Preserving Machine Learning. *arXiv preprint arXiv:1801.10117* (2018).

[36] Zhaorui Li, Zhicong Huang, Chaochao Chen, and Cheng Hong. 2019. Quantification of the leakage in federated learning. *arXiv preprint arXiv:1910.05467* (2019).

[37] Yehuda Lindell. 2017. How to simulate it–a tutorial on the simulation proof technique. *Tutorials on the Foundations of Cryptography* (2017), 277–346.

[38] Azalia Mirhoseini, Ahmad-Reza Sadeghi, and Farinaz Koushanfar. 2016. Cryptoml: Secure outsourcing of big data machine learning applications. In *IEEE International Symposium on Hardware Oriented Security and Trust*. 149–154.

[39] Payman Mohassel and Peter Rindal. 2018. ABY3: A mixed protocol framework for machine learning. In *ACM Conference on Computer and Communications Security*. 35–52.

[40] Payman Mohassel and Yupeng Zhang. 2017. Secureml: A system for scalable privacy-preserving machine learning. In *IEEE Symposium on Security and Privacy*. 19–38.

[41] SEAL 2023. Microsoft SEAL (release 4.1). https://github.com/Microsoft/SEAL. Microsoft Research, Redmond, WA.

[42] Sameer Wagh, Divya Gupta, and Nishanth Chandran. 2019. SecureNN: 3-Party Secure Computation for Neural Network Training. *Proceedings on Privacy Enhancing Technologies* 2019, 3 (2019), 26–49.

[43] Runhua Xu, Nathalie Baracaldo, Yi Zhou, Ali Anwar, James Joshi, and Heiko Ludwig. 2021. FedV: Privacy-Preserving Federated Learning over Vertically Partitioned Data. In *ACM Workshop on Artificial Intelligence and Security*. 181–192.

[44] Runhua Xu, James BD Joshi, and Chao Li. 2019. Cryptonn: Training neural networks over encrypted data. In *IEEE International Conference on Distributed Computing Systems*. 1199–1209.

[45] Kai Yang, Tao Fan, Tianjian Chen, Yuanming Shi, and Qiang Yang. 2019. A quasi-newton method based vertical federated learning framework for logistic regression. *arXiv preprint arXiv:1912.00513* (2019).

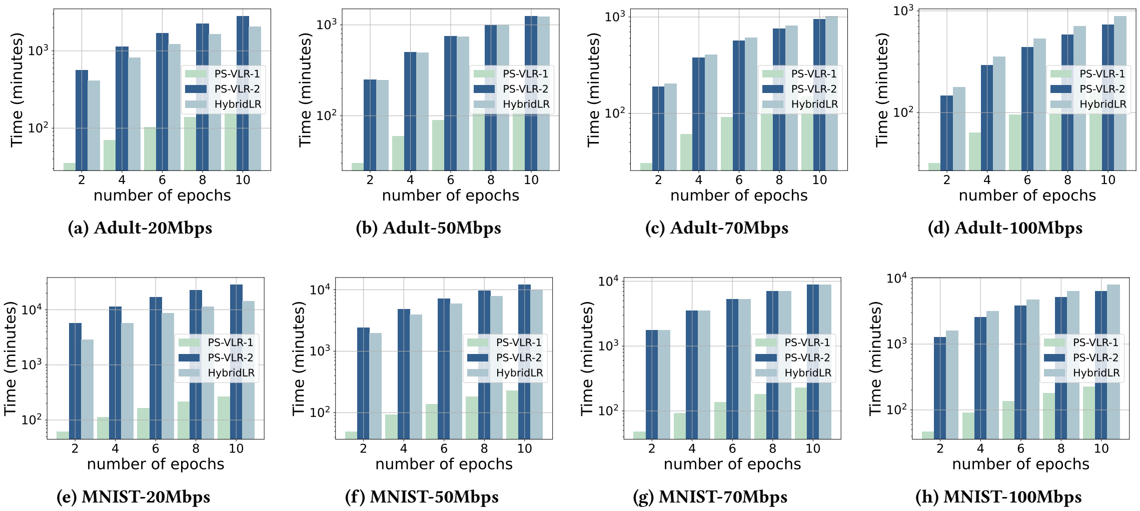
[46] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. 2019. Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology* 10, 2 (2019), 1–19.

[47] Shengwen Yang, Bing Ren, Xuhui Zhou, and Liping Liu. 2019. Parallel distributed logistic regression for vertical federated learning without third-party coordinator. *arXiv preprint arXiv:1911.09824* (2019).

[48] Qingsong Zhang, Bin Gu, Cheng Deng, and Heng Huang. 2021. Secure bilevel asynchronous vertical federated learning with backward updating. *arXiv preprint arXiv:2103.00958* (2021).

[49] Yanjun Zhang, Guangdong Bai, Xue Li, Caitlin Curtis, Chen Chen, and Ryan KL Ko. 2020. Privcoll: Practical privacy-preserving collaborative machine learning. In *European Symposium on Research in Computer Security*. 399–418.

[50] Ligeng Zhu, Zhijian Liu, and Song Han. 2019. Deep leakage from gradients. *Advances in Neural Information Processing Systems* 32 (2019).



(a) Adult-20Mbps

(b) Adult-50Mbps

(c) Adult-70Mbps

(d) Adult-100Mbps

(e) MNIST-20Mbps

(f) MNIST-50Mbps

(g) MNIST-70Mbps

(h) MNIST-100Mbps

Figure 5: Evaluation of the training efficiency of *PS-VLR* and HybridLR under various numbers of epochs

A SUPPLEMENTARY EXPERIMENTS

A.1 Comparsion with HybridLR on the overall training time

Taking the running time of sharing input data into consideration, we evaluate the training efficiency of *PS-VLR* and HybridLR on Adult and MNIST under different bandwidths and numbers of epochs. Figure 5 shows the comparison results, where *PS-VLR*-1 reports the running time of *PS-VLR* including sharing input data and model training, and *PS-VLR*-2 reports the running time of *PS-VLR* including sharing input data, model training, and offline triple generation. We can observe that if the offline triple generation is not considered, *PS-VLR* is more efficient than HybridLR in all settings; otherwise, *PS-VLR* is more efficient than HybridLR when the bandwidth is high (e.g., ≥ 70 Mbps). We can also observe that the running time of sharing input data is much smaller than that of offline triple generation.

Table 7: The running time (s) of *PS-VLR* and HybridLR under various feature sparsities

Sparsity	0.5	0.9	0.99	0.999
HybridLR	148.781	132.587	129.898	128.591
PS-VLR-training	0.374	0.374	0.374	0.374
PS-VLR-offline	71.998	71.998	71.998	71.998

A.2 Effect of feature sparsity

We compare *PS-VLR* with HybridLR in terms of the running time of model training and offline triple generation for one iteration under different sparsities of input features. Specifically, we randomly generate a number of batches with 64 samples, 10,000 features, and different sparsities, and measure the running time of model training and offline triple generation on these batches. For comparing fairness, the running time of the evaluation of the sigmoid function is not recorded. The bandwidth in this experiment is fixed to 70 Mbps. Table 7 illustrates the comparison results. We can observe that the running time of HybridLR decreases as the sparsity grows, while that of *PS-VLR* remains constant. The reason lies in the fact that HybridLR does not secretly share the input data and can exploit sparse computation to improve computational efficiency.

B SECURITY ANALYSIS

B.1 Proof for ISS-VLR

Theorem 3.3. *If the number of epochs e satisfies $e < m_c^*$, where m_c^* is the minimum value of $\{m_{c,i} \mid i \in [2, p]\}$, and $m_{c,i}$ is the number of continuous features held by P_i, we say that Protocol 1 satisfies infinite solution security against a semi-honest adversary who can statically corrupt up to p-1 out of p parties.*

Proof. We start by declaring the private information in Protocol 1, which includes the feature set, label set, residuals, gradients, and model parameters. Then, we discuss whether the intermediate results received by the active party P_1 and the passive party P_i ($i \in [2, p]$) satisfy infinite solution security, respectively.

Active party. Let n be the number of samples, m_i be the number of features held by P_i, $m_{c,i}$ be the number of continuous features held by P_i, b be the batch size, T be the number of iterations, and e be the number of epochs. The intermediate results that P_1 can receive from P_i during training include $\left\{\theta_i^t X_i^t, K_i^t G_i^t, K_i^t \theta_i^{t+1}\right\}_{t=1}^{T}$. Since the systems of equations constructed by the above intermediate results contain common unknown variables, we can solve them jointly. As shown in Lemma B.1, we require that either $m_i \geq \frac{1+\sqrt{1+4b}}{2}$ or $e < \frac{(nm_{c,i}+m_i)b}{(b-m_i^2+m_i)n}$ to make the joint system of equations have infinitely many solutions. Meanwhile, according to Theorem 3.1, e is also required to satisfy $e < m_{c,i}$ to make the system of equations constructed by $\left\{\theta_i^t X_i^t\right\}_{t=1}^{T}$ have infinitely many solutions. Therefore, when $0 < m_i < \frac{1+\sqrt{1+4b}}{2}$, we should determine the quantitative relationship between $u = \frac{(nm_{c,i}+m_i)b}{(b-m_i^2+m_i)n}$ and $v = m_{c,i}$ for getting the least upper bound of e. To achieve this goal, we calculate $u - v$ and determine the sign of it. Specifically, $u - v$ is expressed as follows:

$$u - v = \frac{(nm_{c,i} + m_i)b}{(b - m_i^2 + m_i)n} - m_{c,i} = \frac{nm_{c,i}(m_i^2 - m_i) + m_i b}{(b - m_i^2 + m_i)n}. \tag{9}$$

We can observe from Equation (9) that the denominator and the numerator are both constantly greater than 0. Therefore, if $0 < m_i < \frac{1+\sqrt{1+4b}}{2}$, u is constantly greater than v, implying that $e < m_{c,i}$. Considering that e is only required to satisfy $e < m_{c,i}$ when $m_i \geq \frac{1+\sqrt{1+4b}}{2}$, we can conclude that if $e < m_{c,i}$ is satisfied, all intermediate results sent from P_i to P_1 satisfy infinite solution security. Since P_1 should interact with all passive parties simultaneously, in order for all intermediate results received by P_1 to satisfy infinite solution security, e is required to be less than the number of continuous features held by any passive party, i.e., $e < m_c^*$, where m_c^* is the minimum value of $\{m_{c,i}\}_{i=2}^{p}$.

Passive party. The intermediate results received by P_i include $\{\sigma^t r^t, \alpha\varphi_i^t K_i^t G_i^t + \mu_i^t, \varphi_i^{t+1}\theta_i^{t+1}\}_{t=1}^{T}$. The random masks contained in these intermediate results ensure that the number of unknown variables is great than the number of equations in the system of equations constructed by each intermediate result. Therefore, these intermediate results satisfy infinite solution security.

In summary, if $e < m_c^*$, Protocol 1 satisfies infinite solution security.

□

Lemma B.1. *Given a nonlinear system of equations $\mathcal{S}$ consisting of $h_1 = \{\theta_i^t X_i^t\}_{t=1}^{T}$, $h_2 = \{K_i^t G_i^t\}_{t=1}^{T}$, and $h_3 = \{K_i^t \theta_i^{t+1}\}_{t=1}^{T}$, if the number of features held by P_i satisfies $m_i \geq \frac{1+\sqrt{1+4b}}{2}$ or the number of epochs satisfies $e < \frac{(nm_{c,i}+m_i)b}{(b-m_i^2+m_i)n}$, where n is the number of samples, $m_{c,i}$ is the number of continuous features held by P_i, and b is the batch size, $\mathcal{S}$ has infinitely many solutions.*

Proof. Let n be the number of samples, m_i be the number of features held by P_i, $F_{c,i}$ be the set of continuous features held by P_i, ψ be a function used to count the number of intervals in F_c, $m_{c,i}$ be the size of $F_{c,i}$, $m_{d,i}$ be the number of discrete features held by P_i, b be the batch size, T be the number of iterations, and e be the number of epochs. Since $K_i^t\theta_i^{t+1} = K_i^t\theta_i^t - \alpha K_i^t G_i^t$, we merge h_2 and h_3 into $h_4 = \{K_i^t\theta_i^t\}_{t=1}^{T}$, and $\mathcal{S}$ only consists of h_1 and h_4. To ensure that $\mathcal{S}$ has infinitely many solutions, we should derive a constraint to make the number of variables greater than the number of equations. Specifically, the unknown variables in $\mathcal{S}$ include input features X_i, initial model parameters θ_i^1, random masks $\{K_i^t\}_{t=1}^{T}$, and the artificial variables introduced by considering the value ranges of input features. The total number of unknown variables is $nm_i + m_i + m_i^2 T + n\phi(F_{c,i})$. For the equations in $\mathcal{S}$, they contain not only the equations formed by h_1 and h_4, but also the equations introduced by considering the value ranges of input features. The total number of equations is $ne + m_i T + nm_{d,i} + n\phi(F_{c,i})$. Let the number of unknown variables be greater than the number of equations, we can get an inequality as follows:

$$\begin{aligned} &nm_i + m_i + m_i^2 T + n\phi(F_{c,i}) - (ne + m_i T + nm_{d,i} + n\phi(F_{c,i})) \\ &= nm_{c,i} + m_i + (m_i^2 - m_i - b)T > 0. \end{aligned} \tag{10}$$

If $m_i^2 - m_i - b \geq 0$, i.e., $m_i \geq \frac{1+\sqrt{1+4b}}{2}$, Equation (10) holds constant. Otherwise, the number of iterations T is required to satisfy $T < \frac{nm_{c,i}+m_i}{b-m_i^2+m_i}$ for guaranteeing that Equation (10) holds. Since $T = \frac{ne}{b}$, it can be further deduced that $e < \frac{(nm_{c,i}+m_i)b}{(b-m_i^2+m_i)n}$. In summary, if $m_i \geq \frac{1+\sqrt{1+4b}}{2}$ or $e < \frac{(nm_{c,i}+m_i)b}{(b-m_i^2+m_i)n}$ is satisfied, $\mathcal{S}$ has infinitely many solutions.

□

B.2 Proof for PS-VLR

Figure 6: Ideal functionality for model training $\mathcal{F}_{Train}$

Parameters: Active party P_1 and passive parties P_2, ..., P_p
Inputs: P_i inputs features X_i for $i \in [1, p]$ and P_1 inputs labels Y
Outputs: A set of secretly shared model parameters $\{\langle\theta_j\rangle_i^A\}_{j=1}^{p}$ to P_i for $i \in [1, p]$

Figure 7: Ideal functionality for matrix triple generation $\mathcal{F}_{Triple}$

Parameters: Active party P_1 and passive parties P_2, ..., P_p
Inputs: P_1 inputs the number of rows N_r, the number of columns N_c, and the number of matrix triples k
Outputs: A set of secretly shared matrix triples $\{\langle U_j\rangle_i^A, \langle V_j\rangle_i^A, \langle Z_j\rangle_i^A\}_{j=1}^{k}$ to P_i for $i \in [1, p]$

Theorem 4.1. *Protocol 2 and Protocol 3 securely compute $\mathcal{F}_{Train}$ and $\mathcal{F}_{Triple}$ against a semi-honest adversary who can statically corrupt up to p-1 out of p parties, respectively.*

Proof. An adversary in our threat model can corrupt any subset of parties. Let I be the set of corrupted parties, we construct a simulator $\mathcal{S}$ to generate the view of the adversary in the ideal world. $\mathcal{S}$ submits the corrupted parties' input data to the ideal functionalities $\mathcal{F}_{Train}$ and $\mathcal{F}_{Triple}$, and receives the final outputs. We briefly argue that the views generated by $\mathcal{S}$ are indistinguishable from the views generated by the adversary in the real world. This immediately follows from the security of the SS and threshold FHE, as well as the uniformly random generation of matrices/vectors (U, V, and c) during matrix triple generation.

□