When Homomorphic Encryption Marries Secret Sharing: Secure Large-Scale Sparse Logistic Regression and Applications in Risk Control*

Chaochao Chen[1], Jun Zhou[1], Li Wang[1], Xibin Wu[1], Wenjing Fang[1], Jin Tan[1], Lei Wang[1], Alex X. Liu[1], Hao Wang[2], Cheng Hong[3]

[1]Ant Group, [2]Shandong Normal University, [3]Alibaba Group

{chaochao.ccc, jun.zhoujun, raymond.wangl, xibin.wxb, bean.fwj, tanjin.tj, shensi.wl, alexliu}@antgroup.com

wanghao@sdnu.edu.cn, vince.hc@alibaba-inc.com

ABSTRACT

Logistic Regression (LR) is the most widely used machine learning model in industry for its efficiency, robustness, and interpretability. Due to the problem of data isolation and the requirement of high model performance, many applications in industry call for building a secure and efficient LR model for multiple parties. Most existing work uses either Homomorphic Encryption (HE) or Secret Sharing (SS) to build secure LR. HE based methods can deal with high-dimensional sparse features, but they incur potential security risks. SS based methods have provable security, but they have efficiency issue under high-dimensional sparse features. In this paper, we first present CAESAR, which combines HE and SS to build secure large-scale sparse logistic regression model and achieves both efficiency and security. We then present the distributed implementation of CAESAR for scalability requirement. We have deployed CAESAR in a risk control task and conducted comprehensive experiments. Our experimental results show that CAESAR improves the state-of-the-art model by around 130 times.

CCS CONCEPTS

• **Security and privacy** → **Privacy protections**; **Usability in security and privacy**; • **Computing methodologies** → **Machine learning**.

KEYWORDS

Homomorphic encryption; secret sharing; multi-party computation; large-scale; logistic regression

---

*Jun Zhou is the corresponding author.

1 INTRODUCTION

Logistic Regression (LR) and other machine learning models have been popularly deployed in various applications by different kinds of companies, e.g., advertisement in e-commerce companies [50], disease detection in hospitals [10], and fraud detection in financial companies [48]. In reality, there are also increasingly potential gains if different organizations could collaboratively combine their data for data mining and machine learning. For example, health data from different hospitals can be used together to facilitate more accurate diagnosis, while financial companies can collaborate to train more effective fraud-detection engines. Unfortunately, this cannot be done in practice due to competition and regulation reasons. That is, data are isolated by different parties, which is also known as the "isolated data island" problem [45].

To solve this problem, the concept of privacy-preserving or secure machine learning is introduced. Its main goal is to combine data from multiple parties to improve model performance while protecting data holders from any possibility of information disclosure. The security and privacy concerns, together with the desire of data combination, pose an important challenge for both academia and industry. To date, Homomorphic Encryption (HE) [35] and secure Multi-Party Computation (MPC) [46] are two popularly used techniques to solve the above challenge. For example, many HE-based methods have been proposed to train LR and other machine learning models using participants' encrypted data, as a centralized manner [3, 4, 9, 16, 19, 22, 25, 26, 28, 44, 47]. Although these data are encrypted, they can be abused by the centralized data holder, which raises potential information leakage risk. There are also several HE-based methods which build machine learning models in a decentralized manner [23, 43], i.e., data are still held by participants during model training procedure. These methods process features by participants themselves and therefore can handle high-dimensional sparse features. Moreover, participants only need to communicate encrypted reduced information during model training, and therefore the communication cost is low. However, models are exposed as plaintext to server or participants during each iteration in the training procedure, which can be used to infer extra private information [31], even in semi-honest security setting. In a nutshell, these models are non-provably secure.

Besides HE-based methods, in literature, many MPC—mostly Secret Sharing (SS) [39]—based protocols, are proposed for secure

The original **sparse** feature x

f1	f2	f3	f4	f5
	2		3	

A share of **dense** feature $\langle x \rangle_1$

f1	f2	f3	f4	f5
3	3	2	1	1

The other share of **dense** feature $\langle x \rangle_2$

f1	f2	f3	f4	f5
1	3	2	2	3

Figure 1: Sparse feature becomes dense features after using secret sharing in $\mathbb{Z}_4$.

machine learning models, e.g., linear regression [18], logistic regression [7, 40], tree based model [17], neural network [37, 41, 49], recommender systems [8], and general machine learning algorithms [15, 30, 32, 33]. Besides academic research, large companies are also devoted to develop secure and privacy preserving machine learning systems. For example, Facebook opensourced CrypTen[1] and Visa developed ABY[3] [32]. Comparing with HE based methods, SS based methods have provable security. However, since their idea is secretly sharing the participants' data (both features and labels) on several servers for secure computation, the features become dense after using secret sharing even they are sparse beforehand. Figure 1 shows an example, where the original feature is a 5-dimensional sparse vector. While after secretly share it to two parties, it becomes two dense vectors. Therefore, they cannot handle sparse features efficiently and have high communication complexity when dataset gets large, as we will analyze in Section 4.4.

1.1 Gaps Between Research and Practice

We analyze the gaps between research and practice from the following three aspects. (1) *Number of participants*. In research, most existing works usually assume that any number of parties can jointly build privacy-preserving machine learning models [15, 23, 32, 33], and few of them are customized to only two-party setting. However, in practice, it is always two parties, as we will introduce in Section 6. This is because it usually involves more commercial interests and regulations when there are three or more parties. (2) *Data partition*. Many existing researches are suitable for both data vertically partition and data horizontally partition settings. In practice, when participants are large business companies, they usually have the same batch of samples but different features, i.e., data vertically partition. There are limited researches focus on this setting and leverage its characteristic. (3) *Feature sparsity*. Existing studies usually ignore that features are usually high-dimensional and sparse in practice, which is usually caused by missing feature values or feature engineering such as one-hot. Therefore, how to build secure large-scale LR under data vertically partition is a challenging and valuable task for industrial applications.

1.2 Our Contributions

New protocols by marrying HE and SS. We present CAESAR, a seCure lArge-scalE SpArse logistic Regression model by marrying HE and SS. To guarantee security, we secretly share model parameters to both parties, rather than reveal them during model training [23, 43]. To handle large-scale sparse data when calculating predictions and gradients, we propose a secure sparse matrix multiplication protocol based on HE and SS, which is the key to scalable

secure LR. By combining HE and SS, CAESAR has the advantages of both efficiency and provable security.

Distributed implementation. Our designed implementation framework is comprised of a coordinator and two distributed clusters on both participants' sides. The *coordinator* controls the start and terminal of the clusters. In each *cluster*, we borrow the idea of parameter server [29, 50] and distribute data and model on *servers* who learn model parameters following our proposed CAESAR. Meanwhile, each server delegates the most time-consuming encryption operations to multiple *workers* for distributed encryption. By vertically distribute data and model on servers, and delegate encryption operations on workers, our implementation can scale to large-scale dataset.

Real-world deployment and applications. We deployed CAESAR in a risk control task in Ant Financial (Ant, for short), and conducted comprehensive experiments. The results show that CAESAR significantly outperforms the state-of-the-art secure LR model, i.e., SecureML, especially under the situation where network bandwidth is the bottleneck, e.g., limited communication ability between participants or high-dimensional sparse features. Taking our real-world risk control dataset, which has around 1M samples and 100K features (3:7 vertically partitioned), as an example, it takes 7.72 hours for CAESAR to finish an epoch when bandwidth is 10Mbps and batch size is 4,096. In contrast, SecureML needs to take 1,005 hours under the same setting—a **130x speedup**. To the best of our knowledge, CAESAR is the first secure LR model that can handle such large-scale datasets efficiently.

2 RELATED WORK

In this section, we review literatures on Homomorphic Encryption (HE) based privacy-preserving Machine Learning (ML) and Multi-Party Computation (MPC) based secure ML.

2.1 HE based Privacy-Preserving ML

Most existing HE based privacy-preserving ML models belong to centralized modelling. That is, private data are first encrypted by participants and then outsourced to a server who trains ML models as a centralized manner using HE techniques. Various privacy-preserving ML models are built under this setting, including least square [16], logistic regression [3, 9, 22, 28], and neural network [4, 19, 25, 44, 47]. However, this kind of approach suffers from data abuse problem, since the server can do whatever computations with these encrypted data in hand. The data abuse problem may further raise potential risk of data leakage.

There are also some researches focus on training privacy-preserving ML models using HE under a decentralized manner. That is, the private data are still held by participants during model training. For example, Wu et al. proposed a secure logistic regression model for two parties, assuming that one party has features and the other party has labels [42]. Other researches proposed to build linear regression [21] and logistic regression [43] under horizontally partitioned data. As we described in Section 1, features partitioned vertically is the most common setting in practice. Thus, the above methods are difficult to apply into real-world applications. The most similar work to ours is vertically Federated Logistic Regression (FLR) [23]. FLR trains logistic regression in a decentralized

manner, i.e., both private features and labels are held by participants during model training. Besides, participants only need to communicate compressed encrypted information with each other and a third-party server, thus its communication cost is low. However, it assumes there is a third-party that does not collude with any participants, which may not exist in practice. Moreover, model parameters are revealed to server or participants in plaintext during each iteration in the training procedure, which can be used to infer extra information and cause information leakage in semi-honest security setting [31]. In contrast, in this paper, we propose to marry HE and SS to build secure logistic regression.

2.2 MPC based Secure ML

Besides HE, MPC is also popularly used to build secure ML systems in literature. First of all, there are some general-purpose MPC protocols such as VIFF [12] and SPDZ [13] that can be used to build secure logistic regression model [11]. Secondly, there are also MPC protocols for specific ML algorithms, e.g., garbled circuit and HE based neural network [27], garbled circuit based linear regression [18], logistic regression [40], and neural network [2, 37].

In recent years, there is a trend of building general ML systems using Secret Sharing (SS). For example, Demmler et al. proposed ABY [15], which combines Arithmetic sharing (A), Boolean sharing (B), and Yao's sharing (Y) for general ML. Mohassel and Zhang proposed SecureML [33], which optimized ABY for vectorized scenario so as to compute multiplication of shared matrices and vectors. Later on, Mohassel and Rindal proposed ABY³ [32], which extended ABY and SecureML to a three-server mode setting and thus has better efficiency. Li et al. proposed PrivPy [30] that works under four-server mode. The above approaches are provably secure and their basic idea is secretly sharing the data/model among multi-parties/servers. Under thus circumstances, these systems cannot scale to high-dimensional data even when these data are sparse, since secret sharing will make the sparse data dense.

Recently, Phillipp et al. proposed ROOM [38] to solve the data sparsity problem in machine learning. However, it needs to reveal data sparseness which may cause potential information leakage. For example, when a dataset contains only binary features (0 or 1), a dense sample directly reflects that its features are all ones if using ROOM. Besides, when it involves matrix multiplication, ROOM still needs cryptographical techniques to generate Beaver triples [5], which limits its efficiency for training. In this paper, we propose to combine HE and SS to improve the communication efficiency of the existing MPC based secure logistic regression, which can not only protect data sparseness, but also avoid the time-consuming Beaver triples generation procedure.

3 PRELIMINARIES

In this section, we briefly describe the setting and threat model of our proposal, and present some background knowledge.

3.1 Data Vertically Partitioned by Two-Parties

In this work, we consider secure protocols for two parties who want to build secure logistic regression together. Moreover, existing works on secure logistic regression mainly focus on two cases based on how data are partitioned between participants, i.e., *horizontally*

data partitioning that denotes each party has a subset of the samples with the same features, and *vertically data partitioning* which means each party has the same samples but different features [21]. In this paper, we focus on vertically partitioning setting, since it is more common in industry. It becomes more general as long as one of the participants is a large company who has hundreds of millions of customers.

Note that, in practice, when participants collaboratively build secure logistic regression under vertically data partitioning setting, the first step is matching sample IDs between participants. *Private Set Intersection* technique [36] is commonly used to get matched IDs privately. We omit its details in this paper and only focus on the machine learning part.

3.2 Threat Model

We consider the standard *semi-honest model*, the same as the existing methods [30, 33], where a probabilistic polynomial-time adversary with semi-honest behaviors is considered. In this security model, the adversary may corrupt and control one party (referred as to *the corrupted party*), and try to obtain information about the input of the other party (referred as to *the honest party*). During the protocol executed by the honest party, the adversary will follow the protocol specifically, but may attempt to obtain additional information about the honest party's input by analyzing the corrupted party's *view*, i.e., the transcripts it receives during the protocol execution. The detailed definition of the semi-honest security model can be found in Appendix A.

3.3 Additive Secret Sharing

We use the classic additive secret sharing scheme under our two party ($\mathcal{A}$ and $\mathcal{B}$) setting [6, 39]. Let $\phi = 2^l$ be a large integer, x and y be non-negative integers and $0 < x, y \ll \phi$, and $\mathbb{Z}_\phi$ be the group of integers module ϕ. Assuming that $\mathcal{A}$ wants to **share** a secret x with $\mathcal{B}$, $\mathcal{A}$ first randomly samples an integer r in $\mathbb{Z}_\phi$ as a share $\langle x \rangle_2$ and sends it to $\mathcal{B}$, and then calculates $x - r \mod \phi$ as the other share $\langle x \rangle_1$ and keeps it itself. To this end, $\mathcal{A}$ has $\langle x \rangle_1$ and $\mathcal{B}$ has $\langle x \rangle_2$ such that $\langle x \rangle_1$ and $\langle x \rangle_2$ are randomly distributed integers in $\mathbb{Z}_\phi$ and $x = \langle x \rangle_1 + \langle x \rangle_2 \mod \phi$. Similarlly, assume $\mathcal{B}$ has a secret y and after shares it with $\mathcal{A}$, $\mathcal{A}$ has $\langle y \rangle_1$ and $\mathcal{B}$ has $\langle y \rangle_2$.

Addition. Suppose $\mathcal{A}$ and $\mathcal{B}$ want to secretly calculate $x + y$ in secret sharing, $\mathcal{A}$ computes $\langle z \rangle_1 = \langle x \rangle_1 + \langle y \rangle_1 \mod \phi$ and $\mathcal{B}$ computes $\langle z \rangle_2 = \langle x \rangle_2 + \langle y \rangle_2 \mod \phi$ and each of them gets a share of the addition result. To **reconstruct** a secret, one party just needs to send its share to the other party, and then reconstruct can be done by $z = \langle z \rangle_1 + \langle z \rangle_2 \mod \phi$.

Multiplication. Most existing secret sharing multiplication protocol are based on Beaver's triplet technique [5]. Specifically, to multiply two secretly shared values $\langle x \rangle$ and $\langle y \rangle$ between two parties, they need a shared triple (Beaver's triplet) $\langle u \rangle$, $\langle v \rangle$, and $\langle w \rangle$, where u, v are uniformly random values in $\mathbb{Z}_\phi$ and $w = u \cdot v \mod \phi$. They then make communication and local computations, and finally each of the two parties gets $\langle z \rangle_1$ and $\langle z \rangle_2$, respectively, such that $\langle x \rangle \cdot \langle y \rangle = \langle z \rangle_1 + \langle z \rangle_2$.

Supporting real numbers and vectors. We use fix-point representation to map real numbers to $\mathbb{Z}_\phi$ [30, 33]. Assume $x \in [-p, p]$ is a real number where $0 < p \ll \phi/2$, it can be represented by

$\lfloor 10^c x \rfloor$ if $x \geq 0$ and $\lfloor 10^c x \rfloor + \phi$ if $x < 0$, where c determines the precision of the represented real number, i.e., the fractional part has c bits at most. After this, it can be easily vectorized to support matrices, e.g., SS based secure matrix multiplication in [14].

3.4 Additive Homomorphic Encryption

Additive HE methods, e.g., Okamoto-Uchiyama encryption (OU) [34] and Paillier [35], are popularly used in machine learning algorithms [3], as described in Section 2.1. The use of additive HE mainly has the following steps [1]:

- **Key generation.** One participant generates the public and secret key pair (pk, sk) and publicly distributes pk to the other participant.
- **Encryption.** Given a plaintext x, it is encrypted using pk and a random r, i.e., $[\![x]\!] = \mathbf{Enc}(pk; x, r)$, where $[\![x]\!]$ denotes the ciphertext and r makes sure the ciphertexts are different in multiple encryptions even when the plaintexts are the same.
- **Homomorphic operation.** Given two plaintexts (x and y) and their corresponding ciphertexts ($[\![x]\!]$ and $[\![y]\!]$), there are three types of operations for additive HE, i.e., **OP1:** $[\![x + y]\!] = x + [\![y]\!]$, **OP2:** $[\![x + y]\!] = [\![x]\!] + [\![y]\!]$, and **OP3:** $[\![x \cdot y]\!] = x \cdot [\![y]\!]$. Note that we overload '+' as the homomorphic addition operation.
- **Decryption.** Given a ciphertext $[\![x]\!]$, it is decrypted using sk, i.e., $x = \mathbf{Dec}(sk; [\![x]\!])$.

Similar as secret sharing, the above additive HE only works on a finite field. One can use similar fix-point representation approach to support real numbers in a group of integers module ψ, i.e., $\mathbb{Z}_\psi$. Additive HE operations on matrix work similarly [23].

3.5 Logistic Regression Overview

We briefly describe the key components of logistic regression as follows.

Model and loss. Let $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ be the training dataset, where n is the sample size, $\mathbf{x}_i \in \mathbb{R}^{1 \times d}$ is the feature of i-th sample with d denoting the feature size, and y_i is its corresponding label. Logistic regression aims to learn a model $\mathbf{w}$ so as to minimize the loss $\mathcal{L} = \sum_{i=1}^n l(y_i, \hat{y}_i)$, where $l(y_i, \hat{y}_i) = -y \cdot log(\hat{y}_i) - (1-y) \cdot log(1 - \hat{y}_i)$ with $\hat{y}_i = 1/(1 + e^{-\mathbf{x}_i \cdot \mathbf{w}})$. Note that without loss of generality, we employ bold capital letters (e.g., $\mathbf{X}$) to denote matrices and use bold lowercase letters (e.g., $\mathbf{w}$) to indicate vectors.

Mini-batch gradient descent. The logistic regression model can be learnt efficiently by minimizing the loss using mini-batch gradient descent. That is, instead of selecting one sample or all samples of training data per iteration, a batch of samples are selected and $\mathbf{w}$ is updated by averaging the partial derivatives of the samples in the current batch. Let $\mathbf{B}$ be the current batch, $|\mathbf{B}|$ be the batch size, $\mathbf{X}_B$ and $\mathbf{Y}_B$ be the features and labels in the current batch, then the model updating can be expressed in a vectorized form:

$$\mathbf{w} \leftarrow \mathbf{w} - \frac{\alpha}{|\mathbf{B}|} \cdot \frac{\partial \mathcal{L}}{\partial \mathbf{w}}, \tag{1}$$

where α is the learning rate that controls the moving magnitude and $\partial \mathcal{L} / \partial \mathbf{w} = (\hat{\mathbf{Y}}_B - \mathbf{Y}_B)^T \cdot \mathbf{X}_B$ is the total gradient of the loss with respect to the model in current batch, and we omit regularization terms for conciseness. From it, we can see that model updation involves many matrix multiplication operations. In practice, features
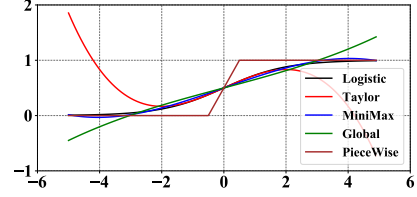

Figure 2: Approximation results of different methods.

are always high-dimensional and sparse, as we described in Section 1. Therefore, sparse matrix multiplication is the key to large scale machine learning. The above mini-batch gradient descent not only benefits from fast convergence, but also enjoys good computation speed by using vectorization libraries.

Sigmoid approximation. The non-linear sigmoid function in logistic regression is not cryptographically friendly. Existing researches have proposed different approximation methods for this, which include Taylor expansion [23, 28], Minimax approximation [9], Global approximation [28], and Piece-wise approximation [33]. Figure 2 shows the approximation results of these methods, where we set the polynomial degree to 3. In this paper, we choose the Minimax approximation method, considering its best performance.

4 CAESAR: SECURE LARGE-SCALE SPARSE LOGISTIC REGRESSION

In this section, we first describe our motivation. We then propose a secure sparse matrix multiplication protocol by combining HE and SS. Finally, we present the secure large-scale logistic regression algorithm and its advantages over the state-of-the-art methods.

4.1 Motivation

In practice, security and efficiency are the two main obstacles to deploy secure machine learning models. As described in Section 2.1, under data vertically partitioned setting, although existing HE based models have good communication efficiency since they can naturally handle high-dimensional sparse feature situation, extra information is leaked during model training procedure which causes security risk. Therefore, it is dangerous to deploy it in real-world applications. In contrast, although existing SS based methods are provable security and computaionally effecient, they cannot handle high-dimensional sparse feature situation, as is shown in Figure 1, which makes them difficulty to be applied in large scale data setting. This is because, in practice, participants are usually large companies who have rich computation resources, and distributed computation is easily implemented inside each participant. However, the network bandwidth between participants are usually quite limited, which makes communication cost be the bottleneck. Therefore, decreasing communication cost is the key to large scale secure machine learning models. To combine the advantages of HE (efficiency) and SS (security), we propose CAESAR, a seCure lArge-scalE SpArse logistic Regression model.

4.2 Secure Sparse Matrix Multiplication Protocol by Combining HE and SS

As we described in Section 3.5, secure sparse matrix multiplication is the key to secure large-scale logistic regression.

Protocol 1: Secure Sparse Matrix Multiplication

Input: A sparse matrix $\mathbf{X}$ hold by $\mathcal{A}$, a matrix $\mathbf{Y}$ hold by $\mathcal{B}$, HE key pair for $\mathcal{A}$ ($\{pk_a, sk_a\}$), HE key pair for $\mathcal{B}$ ($\{pk_b, sk_b\}$)

Output: $\mathbf{Z}_1$ for $\mathcal{A}$ and $\mathbf{Z}_2$ for $\mathcal{B}$ thus that $\mathbf{Z}_1 + \mathbf{Z}_2 = \mathbf{X} \cdot \mathbf{Y}$

1 $\mathcal{B}$ encrypts $\mathbf{Y}$ with pk_b and sends $[\![\mathbf{Y}]\!]_b$ to $\mathcal{A}$
2 $\mathcal{A}$ calculates $[\![\mathbf{Z}]\!]_b = \mathbf{X} \cdot [\![\mathbf{Y}]\!]_b$
3 $\mathcal{A}$ secretly shares $[\![\mathbf{Z}]\!]_b$ using Protocol 2, and after that $\mathcal{A}$ gets $\mathbf{Z}_1$ and $\mathcal{B}$ gets $\mathbf{Z}_2$
4 **return** $\mathbf{Z}_1$ for $\mathcal{A}$ and $\mathbf{Z}_2$ for $\mathcal{B}$

Protocol 2: Secret Sharing in Homomorphically Encrypted Field

Input: Homomorphically encrypted matrix $[\![\mathbf{Z}]\!]_b$ for $\mathcal{A}$, HE key pair for $\mathcal{B}$ ($\{pk_b, sk_b\}$)

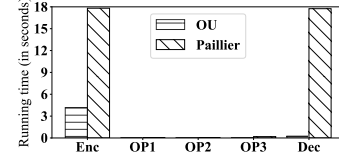**Output:** $\langle\mathbf{Z}\rangle_1$ for $\mathcal{A}$ and $\langle\mathbf{Z}\rangle_2$ for $\mathcal{B}$

1 $\mathcal{A}$ locally generates share $\langle\mathbf{Z}\rangle_1$ from $\mathbb{Z}_\phi$
2 $\mathcal{A}$ calculates $[\![\langle\mathbf{Z}\rangle_2]\!]_b = [\![\mathbf{Z}]\!]_b - \langle\mathbf{Z}\rangle_1 \mod \psi$ and sends $[\![\langle\mathbf{Z}\rangle_2]\!]_b$ to $\mathcal{B}$
3 $\mathcal{B}$ decrypts $[\![\langle\mathbf{Z}\rangle_2]\!]_b$ and gets $\langle\mathbf{Z}\rangle_2$
4 **return** $\langle\mathbf{Z}\rangle_1$ for $\mathcal{A}$ and $\langle\mathbf{Z}\rangle_2$ for $\mathcal{B}$

Notations. Before present our proposal, we first define some notations. Recall in Section 3.1 that we target the setting where data are vertically partitioned by two parties. We use $\mathcal{A}$ and $\mathcal{B}$ to denote the two parties. Correspondingly, we use $\mathbf{X}_a$ and $\mathbf{X}_b$ to denote the features of $\mathcal{A}$ and $\mathcal{B}$, and assume $\mathbf{Y}$ are the labels hold by $\mathcal{B}$. Let $\{pk_a, sk_a\}$ and $\{pk_b, sk_b\}$ be the HE key pairs of $\mathcal{A}$ and $\mathcal{B}$, respectively. Let $[\![x]\!]_a$ and $[\![x]\!]_b$ be the ciphertext of x that are encrypted by using pk_a and pk_b.
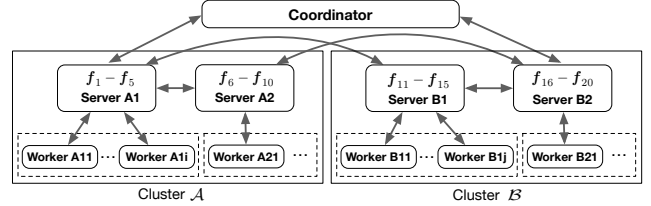
Secure sparse matrix multiplication. We then present a secure sparse matrix multiplication protocol in **Protocol 1**. Given a sparse matrix $\mathbf{X}$ hold by $\mathcal{A}$ and a dense matrix $\mathbf{Y}$ hold by $\mathcal{B}$ we aim to securely calculate $\mathbf{X} \cdot \mathbf{Y}$ without revealing the value of $\mathbf{X}$ and $\mathbf{Y}$. In Protocol 1, Line 1 shows that $\mathcal{B}$ encrypts $\mathbf{Y}$ and sends it to $\mathcal{A}$. Line 2 is the ciphertext multiplication using additive HE, which can be significnatly speed up by parallelization as long as $\mathbf{X}$ is sparse. Line 3 shows how to generate secrets under homomorphically encrypted field, as shown in Protocol 2 [21]. Compared with the existing SS based secure matrix multiplication (Section 3.3), the communication cost of Protocol 1 will be much cheaper when $\mathbf{Y}$ is smaller than $\mathbf{X}$, which is common in machine learning models, e.g., the model parameter vector is smaller than the feature matrix in logistic regression model. We present the security proof of Protocol 1 and Protocol 2 in Appendix B.

4.3 Secure Large-Scale Logistic Regression

With secure sparse matrix multiplication protocol in hand, we now present CAESAR, a seCure lArge-scalE SpArse logistic Regression model, in Algorithm 1, where we omit mod (refers to Section 3.3), mini-batch (refers to Section 3.5), and regularizations for conciseness. The basic idea is that $\mathcal{A}$ and $\mathcal{B}$ secretly **share** the models between two parties (Line 5 and Line 6), so that models are always

Figure 3: Running time comparison of different computation types for OU and Paillier.

Figure 4: Implementation framework of CAESAR.

secret shares during model training, and finally **reconstruct** models after training (Line 30-Line 33). Meanwhile, $\mathcal{A}$ and $\mathcal{B}$ keep their private features and labels. During model training, we calculate the shares of $\mathbf{X} \cdot \mathbf{w}$ using Protocol 1 (Line 10-13), and approximate the logistic function using Minimax approximation (Line 14 and Line 15), as described in Section 3.5. After it, $\mathcal{B}$ calculates the prediction in ciphertext and secretly shares it using Protocol 2 (Line 16). $\mathcal{A}$ and $\mathcal{B}$ then calculate the shares of error (Line 18 and Line 19) and calculate the shares of the gradients using Protocol 1 and Protocol 2 (Line 21-24). They finally update the shares of their models using gradient descent (Line 26-27). During model training, private features and labels are kept by participants themselves, while models and gradients are either *secretly shared* or *homomorphically encrypted*. Algorithm 1 is exactly a classic secure two-party computation if one takes it as a function.

4.4 Advantages of CAESAR

We now analyze the advantages of CAESAR over existing secret sharing based secure logistic regression protocols [15, 33]. It can be seen from Algorithm 1 that, for CAESAR, in each mini-batch (iteration), the communication complexity between $\mathcal{A}$ and $\mathcal{B}$ is $O(7|\mathbf{B}| + 2d)$. That is, $O(7n + 2nd/|\mathbf{B}|)$ for passing the dataset once (one epoch), where $|\mathbf{B}|$ is batch size, n is sample size, and d is the feature size of both parties. In comparison, for the existing secret sharing based secure protocols, e.g., SecureML [33] and ABY [15], the communication complexity between $\mathcal{A}$ and $\mathcal{B}$ is $O(4nd)$ for each epoch during the online phase, ignoring the time-consuming offline circuit and Beaver's triplet generation phase [5]. Our protocol has much less communication costs than existing secure LR protocols, especially when it refers to large scale datasets.

5 IMPLEMENTATION

In this section, we first describe the motivation of our implementation, and then present the detailed implementation of CAESAR.

5.1 Motivation

Our proposed CAESAR has overwhelming advantages against existing secret sharing based protocols in terms of communication

Algorithm 1: CAESAR: seCure lArge-scalE SpArse logistic Regression

---

Input: features for party $\mathcal{A}$ ($\mathbf{X}_a$), features for party $\mathcal{B}$ ($\mathbf{X}_b$), labels for $\mathcal{B}$ ($\mathbf{y}$), HE key pair for $\mathcal{A}$ ($\{pk_a, sk_a\}$), HE key pair for $\mathcal{B}$
    ($\{pk_b, sk_b\}$), max iteration number (T), and polynomial coefficients (q_0, q_1, q_2)

Output: models for party $\mathcal{A}$ ($\mathbf{w}_a$) and models for party $\mathcal{B}$ ($\mathbf{w}_b$)

1   <u>**Initialization:**</u>

2   $\mathcal{A}$ and $\mathcal{B}$ initialize their logistic regression models, i.e., $\mathbf{w}_a$ and $\mathbf{w}_b$, respectively

3   $\mathcal{A}$ and $\mathcal{B}$ exchange their public key pk_a and pk_b

4   **Secretly share models:**

5   $\mathcal{A}$ locally generates shares $\langle \mathbf{w}_a \rangle_1$ and $\langle \mathbf{w}_a \rangle_2$, keeps $\langle \mathbf{w}_a \rangle_1$, and sends $\langle \mathbf{w}_a \rangle_2$ to $\mathcal{B}$

6   $\mathcal{B}$ locally generates shares $\langle \mathbf{w}_b \rangle_1$ and $\langle \mathbf{w}_b \rangle_2$, keeps $\langle \mathbf{w}_b \rangle_2$, and sends $\langle \mathbf{w}_b \rangle_1$ to $\mathcal{A}$

7   **Training model:**

8   **for** $t = 1$ *to* T **do**

9   **Calculate prediction:**

10      $\mathcal{A}$ calculates $\langle \mathbf{z}_a \rangle_1 = \mathbf{X}_a \cdot \langle \mathbf{w}_a \rangle_1$

11      $\mathcal{A}$ and $\mathcal{B}$ securely calculate $\langle \mathbf{z}_a \rangle_2 = \mathbf{X}_a \cdot \langle \mathbf{w}_a \rangle_2$ using Protocol 1, and after that $\mathcal{A}$ gets $\langle\langle \mathbf{z}_a \rangle_2 \rangle_1$ and $\mathcal{B}$ gets the result $\langle\langle \mathbf{z}_a \rangle_2 \rangle_2$

12      $\mathcal{B}$ calculates $\langle \mathbf{z}_b \rangle_2 = \mathbf{X}_b \cdot \langle \mathbf{w}_b \rangle_2$

13      $\mathcal{A}$ and $\mathcal{B}$ securely calculate $\langle \mathbf{z}_b \rangle_1 = \mathbf{X}_b \cdot \langle \mathbf{w}_b \rangle_1$ using Protocol 1, and after that $\mathcal{A}$ gets $\langle\langle \mathbf{z}_b \rangle_1 \rangle_1$ and $\mathcal{B}$ gets the result $\langle\langle \mathbf{z}_b \rangle_1 \rangle_2$

14      $\mathcal{A}$ calculates $\langle \mathbf{z} \rangle_1 = \langle \mathbf{z}_a \rangle_1 + \langle\langle \mathbf{z}_a \rangle_2 \rangle_1 + \langle\langle \mathbf{z}_b \rangle_1 \rangle_1$, $\langle \mathbf{z} \rangle_1^2$, and $\langle \mathbf{z} \rangle_1^3$ and sends ciphertext $[\![\langle \mathbf{z} \rangle_1]\!]_a$, $[\![\langle \mathbf{z} \rangle_1^2]\!]_a$, and $[\![\langle \mathbf{z} \rangle_1^3]\!]_a$ to $\mathcal{B}$

15      $\mathcal{B}$ calculates $\langle \mathbf{z} \rangle_2 = \langle \mathbf{z}_b \rangle_2 + \langle\langle \mathbf{z}_a \rangle_2 \rangle_2 + \langle\langle \mathbf{z}_b \rangle_1 \rangle_2$, $[\![\mathbf{z}]\!]_a = [\![\langle \mathbf{z} \rangle_1]\!]_a + \langle \mathbf{z} \rangle_2$, and
        $[\![\mathbf{z}^3]\!]_a = [\![\langle \mathbf{z} \rangle_1^3]\!]_a + 3[\![\langle \mathbf{z} \rangle_1^2]\!]_a \odot \langle \mathbf{z} \rangle_2 + 3[\![\langle \mathbf{z} \rangle_1]\!]_a \odot \langle \mathbf{z} \rangle_2^2 + \langle \mathbf{z} \rangle_2^3$

16      $\mathcal{B}$ calculates $[\![\hat{\mathbf{y}}]\!]_a = q_0 + q_1 [\![\mathbf{z}]\!]_a + q_2 [\![\mathbf{z}^3]\!]_a$, $[\![\mathbf{e}]\!]_a = [\![\hat{\mathbf{y}}]\!]_a - \mathbf{y}$, and secretly shares $[\![\hat{\mathbf{y}}]\!]_a$ using Protocol 2, and after that $\mathcal{A}$ gets $\langle \hat{\mathbf{y}} \rangle_1$
        and $\mathcal{B}$ gets $\langle \hat{\mathbf{y}} \rangle_2$

17   **Calculate shared error:**

18      $\mathcal{A}$ calculates error $\langle \mathbf{e} \rangle_1 = \langle \hat{\mathbf{y}} \rangle_1$

19      $\mathcal{B}$ calculates error $\langle \mathbf{e} \rangle_2 = \langle \hat{\mathbf{y}} \rangle_2 - \mathbf{y}$

20   **Calculate gradients:**

21      $\mathcal{B}$ locally calculates $[\![\mathbf{e}]\!]_a^T = [\![\hat{\mathbf{y}}]\!]_a - \mathbf{y}$ and $[\![\mathbf{g}_b]\!]_a = [\![\mathbf{e}]\!]_a^T \cdot \mathbf{X}_b$

22      $\mathcal{B}$ secretly shares $[\![\mathbf{g}_b]\!]_a$ using Protocol 2, and after that $\mathcal{A}$ gets $\langle \mathbf{g}_b \rangle_1$ and $\mathcal{B}$ gets $\langle \mathbf{g}_b \rangle_2$

23      $\mathcal{A}$ calculates $\langle \mathbf{g}_a \rangle_1 = \langle \mathbf{e} \rangle_1^T \cdot \mathbf{X}_a$

24      $\mathcal{A}$ and B securely calculate $\langle \mathbf{g}_a \rangle_2 = \langle \mathbf{e} \rangle_2^T \cdot \mathbf{X}_A$ using Protocol 1, and after that $\mathcal{A}$ gets $\langle\langle \mathbf{g}_a \rangle_2 \rangle_1$ and $\mathcal{B}$ gets $\langle\langle \mathbf{g}_a \rangle_2 \rangle_2$

25   **Update model:**

26      $\mathcal{A}$ updates $\langle \mathbf{w}_a \rangle_1$ and $\langle \mathbf{w}_b \rangle_1$ by $\langle \mathbf{w}_a \rangle_1 \leftarrow \langle \mathbf{w}_a \rangle_1 - \alpha \cdot (\langle \mathbf{g}_a \rangle_1 + \langle\langle \mathbf{g}_a \rangle_2 \rangle_1)$ and $\langle \mathbf{w}_b \rangle_1 \leftarrow \langle \mathbf{w}_b \rangle_1 - \alpha \cdot \langle \mathbf{g}_b \rangle_1$

27      $\mathcal{B}$ updates $\langle \mathbf{w}_a \rangle_2$ and $\langle \mathbf{w}_b \rangle_2$ by $\langle \mathbf{w}_a \rangle_2 \leftarrow \langle \mathbf{w}_a \rangle_2 - \alpha \cdot \langle\langle \mathbf{g}_a \rangle_2 \rangle_2$ and $\langle \mathbf{w}_b \rangle_2 \leftarrow \langle \mathbf{w}_b \rangle_2 - \alpha \cdot \langle \mathbf{g}_b \rangle_2$

28   **end**

29   **Reconstructing models:**

30   $\mathcal{A}$ sends $\langle \mathbf{w}_b \rangle_1$ to $\mathcal{B}$

31   $\mathcal{B}$ sends $\langle \mathbf{w}_a \rangle_2$ to $\mathcal{A}$

32   $\mathcal{A}$ reconstructs $\mathbf{w}_a = \langle \mathbf{w}_a \rangle_1 + \langle \mathbf{w}_a \rangle_2$

33   $\mathcal{B}$ reconstructs $\mathbf{w}_b = \langle \mathbf{w}_b \rangle_1 + \langle \mathbf{w}_b \rangle_2$

34   **return** models for party $\mathcal{A}$ ($\mathbf{w}_a$) and models for party $\mathcal{B}$ ($\mathbf{w}_b$)

---

efficiency, as we have analyzed in Section 4.3. Although CAESAR involves additional HE operations, this can be solved by distributed computations. To help design reasonable distributed computation framework, we first analyze the running time of different computations of additive HE. We choose two additive HE methods, i.e., OU [34] and Paillier [35]. They have five types of computations, i.e., encryption (**Enc**), decryption (**Dec**), and three types of homomorphic operations: **OP1:** $[\![x + y]\!] = x + [\![y]\!]$, **OP2:** $[\![x + y]\!] = [\![x]\!] + [\![y]\!]$, and **OP3:** $[\![x \cdot y]\!] = x \cdot [\![y]\!]$. We run these computations 1,000 times and report their running time in Figure 3. From it, we can find that

OU has better performance than Paillier, and **Enc** is the most time-consuming computation type. Therefore, improving the encryption efficiency is the key to distributed implementation.

5.2 Distributed Implementation

Overview. Overall, our designed implementation framework of CAESAR is comprised of a coordinator and two clusters on both participants' side, as shown in Figure 4. The coordinator controls the start and terminal of the clusters based on a certain condition, e.g., the number of iterations. Each cluster is also a distributed

learning system, which consists of **servers** and **workers** and is maintained by participant ($\mathcal{A}$ or $\mathcal{B}$) itself.

Vertically distribute data and model on servers. To support high-dimensional features and the corresponding model parameters, we borrow the idea of distributed data and model from parameter server [29, 50]. Specifically, each cluster has a group of **servers** who split features and models vertically, as shown in Figure 4, where each server has 5 features for example. The label could be held by either $\mathcal{A}$ or $\mathcal{B}$, and we omit it in Figure 4 for conciseness. Note that $\mathcal{A}$ and $\mathcal{B}$ should have the same number of servers, so that each server pair, e.g., Server **A1** and Server **B1**, learn the model parameters using Algorithm 1. In each mini-batch, all the server pairs of $\mathcal{A}$ and $\mathcal{B}$ first calculate partial predictions in parallel, and then, servers in each cluster communicate with each other to get the whole predictions.

Distribute encryption operations on workers. During model training for the server pairs, when it involves encryption operation, each server distributes its plaintext data to *workers* for distributed encryption. After workers finish encryption, they send the ciphertext data to the corresponding server for successive computations. Take Line 14 in Algorithm 1 for example, each server of $\mathcal{A}$ (e.g., Server **A1**) sends partial plaintext data, i.e., $\langle \mathbf{z} \rangle_1$, $\langle \mathbf{z} \rangle_1^2$, and $\langle \mathbf{z} \rangle_1^3$, to its workers (Worker **A11** to Worker**A1i**) for encryption, and then these workers send $[\![\langle \mathbf{z} \rangle_1]\!]_a$, $[\![\langle \mathbf{z} \rangle_1^2]\!]_a$, and $[\![\langle \mathbf{z} \rangle_1^3]\!]_a$ back to Server **A1**. The communication cost in each cluster is cheap, since they are in a local area network. Moreover, two clusters communicate shared or encrypted information with each other to finish model learning, following Algorithm 1.

By vertically distributing data and model on servers, and distributing encryption operations on workers, our implementation can scale to large datasets.

6 DEPLOYMENT AND APPLICATIONS

In this section, we deploy CAESAR into a risk control task, and conduct comprehensive expreiments on it to study the effectiveness of CAESAR.

6.1 Experiment Setup

Scenario. Ant provides online payment services, whose customers include both individual users and large merchants. Users can make online transactions to merchants through Ant, and to protect user's property, controlling transaction risk is rather important to both Ant and its large merchants. Under such scenario, rich features, including user feature in Ant, transaction (context) features in Ant, and user feature in merchant, are the key to build intelligent risk control models. However, due to the data isolation problem, these data cannot be shared with each other directly. To build a more intelligent risk control system, we deploy CAESAR for Ant and the merchant to collaboratively build a secure logistic regression model due to the requirements of robustness and interpretability.

Datasets. We use the real-world dataset in the above scenario, where Ant has 30,100 features and label and the merchant has 70,100 features, and the feature sparsity degree is about 0.02%. Among them, Ant mainly has transaction features (e.g., transaction amount) and partial user feature (e.g., user age), while the merchant has

Table 1: Comparison results

Metric	AUC	KS	F1	Recall@0.9precision
Ant-LR	0.9862	0.9018	0.5350	0.2635
SecureML	0.9914	0.9415	0.6167	0.3598
CAESAR	**0.9914**	**0.9415**	**0.6167**	**0.3598**

the other partial user behavior features (e.g., visiting count). The sparse features mainly come from the incomplete user profiles or feature engineering such as one-hot. The whole dataset has 1,236,681 samples and among which there are 1,208,569 positive (normal) samples and 28,112 negative (risky) samples. We split the dataset into two parts based on the timeline, the 80% earlier happened transactions are taken as training dataset while the later 20% ones are taken as test dataset.

Metrics. We adopt four metrics to evaluate model performance for risk control task, i.e., (1) Area Under the ROC Curve (AUC), (2) Kolmogorov-Smirnov (KS) statistic, (3) F1 value, and (4) recall of the 90% precision (Recall@0.9precision), i.e., the recall value when the classification model reaches 90% precision. These four metrics evaluate a classification model from different aspects, the first three metrics are commonly used in literature, and the last metric is commonly used in industry under imbalanced tasks such as risk control and fraud detection. For all the metrics, the bigger values indicate better model performance.

Comparison methods. To test the *effectiveness* of CAESAR, we compare it with the plaintext logistic regression model using Ant's features only, and the existing SS based secure logistic regression, i.e., SecureML [33]. To test the *efficiency* of CAESAR, we compare it with SecureML [33]. SecureML is based on secret sharing, and thus cannot handle high-dimensional sparse features, as we have described in Figure 1. Note that, we cannot empirically compare the performance of CAESAR with the plaintext logistic regression using mixed plaintext data, since the data are isolated.

Hyper-parameters. We choose OU as the HE method, and set key length to 2,048. We set $l = 64$ and set ψ to be a large number with longer than 1,365 bits (2/3 of the key length). We fix the server number to 1 since it is suitable for our dataset, and vary the number of workers and bandwidth to study their effects on CAESAR.

6.2 Comparison Results

Effectiveness. We first compare CAESAR with plaintext logistic regression using Ant's data only (Ant-LR) to test its effectiveness. Traditionally, Ant can build logistic regression model using its plaintext features only. With secure logistic regression models, i.e., SecureML and CAESAR, Ant can build better logistic regression model together with the merchant, without compromising their private data. We summarize their comparison results in Table 1. From it, we can observe that Ant-LR can already achieve satisfying performance. However, we also find that SecureML and CAESAR have the same performance, i.e., they consistently achieve much better performance than Ant-LR. Take Recall@0.9precision—the most practical metric in industry—for example, CAESAR increases the recall rate of Ant-LR as high as 36.55% while remaining the same precision (90%). This means that CAESAR captures 36.55% more risky transactions than the traditional Ant-LR model while keeping the same accuracy,
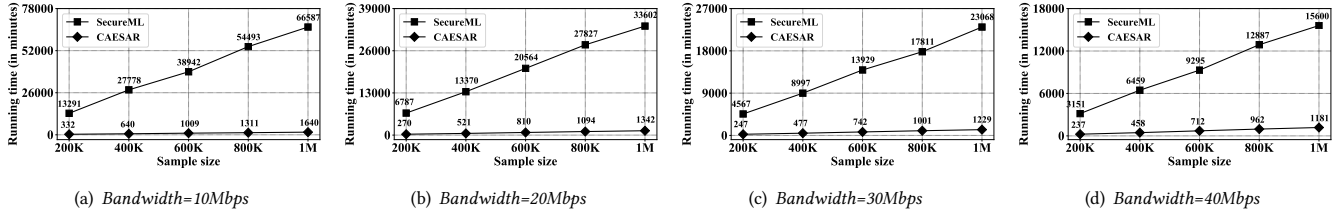
(a) *Bandwidth=10Mbps* (b) *Bandwidth=20Mbps* (c) *Bandwidth=30Mbps* (d) *Bandwidth=40Mbps*

Figure 5: Running time (per epoch) comparison with respect to sample size by varying bandwidth (batch size = 1,024).

(a) *Bandwidth=10Mbps* (b) *Bandwidth=20Mbps* (c) *Bandwidth=30Mbps* (d) *Bandwidth=40Mbps*

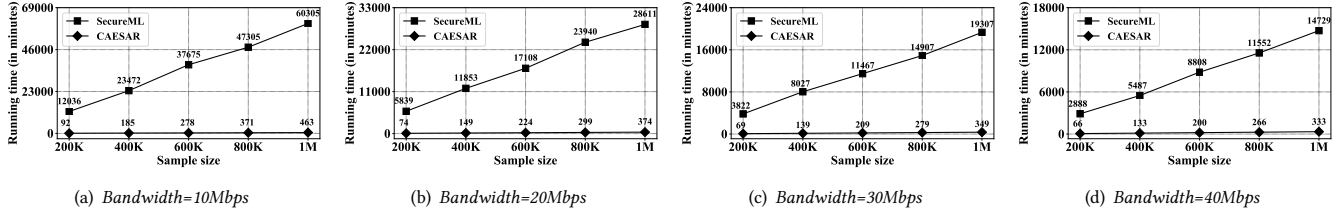**Figure 6: Running time (per epoch) comparison with respect to sample size by varying bandwidth (batch size = 4,096).**

(a) *Effect of worker number* (b) *Effect of feature number* (c) *Effect of batch size* (d) *Effect of network bandwidth*
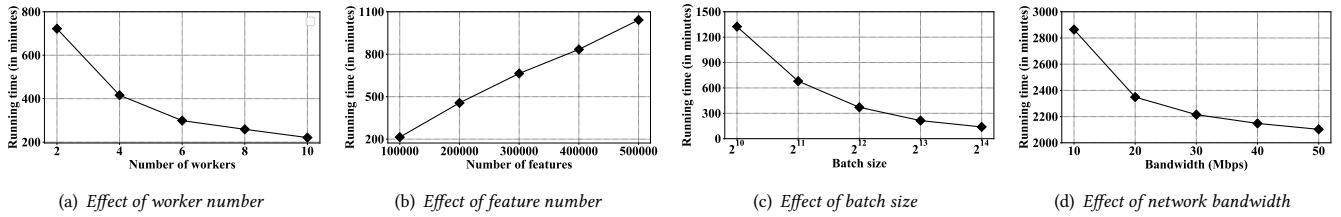
Figure 7: Effects of parameters on CAESAR.

which is quite a high improvement in risk control task. The result is easy to explain: more valuable features will naturally improve risk control ability, which indicates the effectiveness of CAESAR.

Efficiency. We compare CAESAR with SecureML to test its efficiency. To do this, we set the worker number to 10, use all the features, and vary network bandwidth to study the running time of both CAESAR and SecureML for passing the data once (one epoch). We show the results in Figure 5 and Figure 6, where we set batch size to 1,024 and 4,096, respectively. From them, we observe that, CAESAR consistently achieves better efficiency than SecureML, especially when the network bandwidth is limited. Specifically, take Figure 5 for example, CAESAR improves the running speed of SecureML by 40x, 25x, 18x, 13x, in average, when the network bandwidth are 10Mbps, 20Mbps, 30Mbps, and 40Mbps, respectively. Moreover, we also find that, increasing batch size will also improve the speedup of CAESAR against SecureML. Take bandwidth=10Mbps for example, CAESAR improves the running speed of SecureML by 40x and 130x when batch size are 1,024 and 4,096, respectively. One should also notice that, for SecureML, we only count the online running time, and it will take much longer time if we count the offline Beaver's triples generation time.

6.3 Parameter Analysis

To further study the efficiency of CAESAR, we change the number of workers, the number of features, batch size, and bandwidth, and report the running time of CAESAR per epoch.

Effect of worker number. We first use all the features, fix batch size to 8,192, and bandwidth to 32Mbps to study the effect of worker number on CAESAR. From Figure 7 (a), we can find that worker number significantly affects the efficiency of CAESAR. With more workers, CAESAR can scale to large datasets, which indicates the scalability of our distributed implementation.

Effect of feature number. We then fix worker number to 10, batch size to 8,192, and bandwidth to 32Mbps to study the effect of feature number on CAESAR. We can find from Figure 7 (b) that CAESAR scales linearly with feature size, the same as we have analyzed in Section 4.3, which proves the scalability of CAESAR.

Effect of batch size. Next, we fix worker number to 10, use all the features, and set bandwidth to 32Mbps to study the effect of batch size on CAESAR. From Figure 7 (c), we find that the running time of CAESAR decreases when batch size increases. This is because in each epoch, the communication complexity between $\mathcal{A}$ and $\mathcal{B}$ is $O(7n + 2nd/|\mathbf{B}|)$, as we analyzed in Section 4.3, and therefore, increasing batch size will decrease the running time of each epoch.

Effect of network bandwidth. Finally, we fix worker number to 10, use all the features, and set batch size to 8,192 to study the effect of bandwidth on CAESAR. We observe from Figure 7 (d) that the running time of CAESAR also decreases with the increases of bandwidth, which is consistent with common sense. However, when bandwidth is large enough, the running time of CAESAR tends to be stable, this is because computation time, instead of communication time, becomes the new bottleneck.

7 CONCLUSION AND FUTURE WORK

In this paper, to solve the efficiency and security problem of the existing secure and privacy-preserving logistic regression models, we propose CAESAR, which combines homomorphic encryption and secret sharing to build seCure lArge-scalE SpArse logistic Regression model. We then implemented CAESAR distributedly across different parties. Finally, we deployed CAESAR into a risk control task and conducted experiments on it. In future, we plan to customize CAESAR for more machine learning models and deploy them for more applications.

REFERENCES

[1] Abbas Acar, Hidayet Aksu, A Selcuk Uluagac, and Mauro Conti. 2018. A survey on homomorphic encryption schemes: Theory and implementation. *CSUR* 51, 4 (2018), 79.

[2] Nitin Agrawal, Ali Shahin Shamsabadi, Matt J Kusner, and Adrià Gascón. 2019. QUOTIENT: Two-Party Secure Neural Network Training and Prediction. In *CCS*. ACM, 1231–1247.

[3] Yoshinori Aono, Takuya Hayashi, Le Trieu Phong, and Lihua Wang. 2016. Scalable and secure logistic regression via homomorphic encryption. In *CODASPY*. ACM, 142–144.

[4] Ahmad Al Badawi, Jin Chao, Jie Lin, Chan Fook Mun, Sim Jun Jie, Benjamin Hong Meng Tan, Xiao Nan, Khin Mi Mi Aung, and Vijay Ramaseshan Chandrasekhar. 2018. The AlexNet moment for homomorphic encryption: HCNN, the first homomorphic CNN on encrypted data with GPUs. *arXiv preprint arXiv:1811.00778* (2018).

[5] Donald Beaver. 1991. Efficient multiparty protocols using circuit randomization. In *Cryptology*. Springer, 420–432.

[6] Elette Boyle, Niv Gilboa, and Yuval Ishai. 2015. Function secret sharing. In *Eurocrypt*. Springer, 337–367.

[7] Chaochao Chen, Liang Li, Wenjing Fang, Jun Zhou, Li Wang, Lei Wang, Shuang Yang, Alex Liu, and Hao Wang. 2020. Secret Sharing based Secure Regressions with Applications. *arXiv preprint arXiv:2004.04898* (2020).

[8] Chaochao Chen, Liang Li, Bingzhe Wu, Cheng Hong, Li Wang, and Jun Zhou. 2020. Secure social recommendation based on secret sharing. In *ECAI*. 506–512.

[9] 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), 81.

[10] Jonathan H Chen and Steven M Asch. 2017. Machine learning and prediction in medicine—beyond the peak of inflated expectations. *The New England journal of medicine* 376, 26 (2017), 2507.

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

[12] Ivan Damgård, Martin Geisler, Mikkel Krøigaard, and Jesper Buus Nielsen. 2009. Asynchronous multiparty computation: Theory and implementation. In *International Workshop on Public Key Cryptography*. Springer, 160–179.

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

[14] Martine De Cock, Rafael Dowsley, Caleb Horst, Raj Katti, Anderson CA Nascimento, Wing-Sea Poon, and Stacey Truex. 2017. Efficient and private scoring of decision trees, support vector machines and logistic regression models based on pre-computation. *TDSC* 16, 2 (2017), 217–230.

[15] Daniel Demmler, Thomas Schneider, and Michael Zohner. 2015. ABY-A Framework for Efficient Mixed-Protocol Secure Two-Party Computation.. In *NDSS*.

[16] Pedro M Esperança, Louis JM Aslett, and Chris C Holmes. 2017. Encrypted accelerated least squares regression. *arXiv preprint arXiv:1703.00839* (2017).

[17] Wenjing Fang, Chaochao Chen, Jin Tan, Chaofan Yu, Yufei Lu, Li Wang, Lei Wang, Jun Zhou, et al. 2020. A Hybrid-Domain Framework for Secure Gradient Tree Boosting. *arXiv preprint arXiv:2005.08479* (2020).

[18] Adrià Gascón, Phillipp Schoppmann, Borja Balle, Mariana Raykova, Jack Doerner, Samee Zahur, and David Evans. 2017. Privacy-preserving distributed linear regression on high-dimensional data. *PETs*, 345–364.

[19] 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 *ICML*. 201–210.

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

[21] Rob Hall, Stephen E Fienberg, and Yuval Nardi. 2011. Secure multiple linear regression based on homomorphic encryption. *Journal of Official Statistics* 27, 4 (2011), 669.

[22] Kyoohyung Han, Seungwan Hong, Jung Hee Cheon, and Daejun Park. 2019. Logistic Regression on Homomorphic Encrypted Data at Scale. In *IAAI*. 9466–9471. https://doi.org/10.1609/aaai.v33i01.33019466

[23] 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).

[24] Carmit Hazay and Yehuda Lindell. 2010. *Efficient secure two-party protocols: Techniques and constructions.* Springer Science & Business Media.

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

[26] Yichen Jiang, Jenny Hamer, Chenghong Wang, Xiaoqian Jiang, Miran Kim, Yongsoo Song, Yuhou Xia, Noman Mohammed, Md Nazmus Sadat, and Shuang Wang. 2018. SecureLR: Secure logistic regression model via a hybrid cryptographic protocol. *IEEE/ACM TCBB* 16, 1 (2018), 113–123.

[27] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. 2018. {GAZELLE}: A Low Latency Framework for Secure Neural Network Inference. In *USENIX Security*. 1651–1669.

[28] Miran Kim, Yongsoo Song, Shuang Wang, Yuhou Xia, and Xiaoqian Jiang. 2018. Secure logistic regression based on homomorphic encryption: Design and evaluation. *JMIR medical informatics* 6, 2 (2018), e19.

[29] Mu Li, David G Andersen, Jun Woo Park, Alexander J Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J Shekita, and Bor-Yiing Su. 2014. Scaling distributed machine learning with the parameter server. In *USENIX Security*. 583–598.

[30] 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).

[31] Zhaorui Li, Zhicong Huang, Chaochao Chen, and Cheng Hong. 2019. Quantification of the Leakage in Federated Learning. *arXiv preprint arXiv:1910.05467* (2019).

[32] Payman Mohassel and Peter Rindal. 2018. ABY 3: a mixed protocol framework for machine learning. In *CCS*. ACM, 35–52.

[33] Payman Mohassel and Yupeng Zhang. 2017. Secureml: A system for scalable privacy-preserving machine learning. In *S&P*. IEEE, 19–38.

[34] Tatsuaki Okamoto and Shigenori Uchiyama. 1998. A new public-key cryptosystem as secure as factoring. In *Eurocrypt*. Springer, 308–318.

[35] Pascal Paillier. 1999. Public-key cryptosystems based on composite degree residuosity classes. In *Eurocrypt*. Springer, 223–238.

[36] Benny Pinkas, Thomas Schneider, and Michael Zohner. 2014. Faster Private Set Intersection Based on {OT} Extension. In *USENIX Security*. 797–812.

[37] Bita Darvish Rouhani, M Sadegh Riazi, and Farinaz Koushanfar. 2018. Deepsecure: Scalable provably-secure deep learning. In *DAC*. ACM, 2.

[38] Phillipp Schoppmann, Adrià Gascón, Mariana Raykova, and Benny Pinkas. 2019. Make some room for the zeros: Data sparsity in secure distributed machine learning. In *CCS*. 1335–1350.

[39] Adi Shamir. 1979. How to share a secret. *Commun. ACM* 22, 11 (1979), 612–613.

[40] Haoyi Shi, Chao Jiang, Wenrui Dai, Xiaoqian Jiang, Yuzhe Tang, Lucila Ohno-Machado, and Shuang Wang. 2016. Secure multi-pArty computation grid LOgistic REgression (SMAC-GLORE). *BMC medical informatics and decision making* 16, 3 (2016), 89.

[41] Sameer Wagh, Divya Gupta, and Nishanth Chandran. 2018. SecureNN: Efficient and Private Neural Network Training. *IACR Cryptology ePrint Archive* 2018 (2018), 442.

[42] Shuang Wu, Tadanori Teruya, Junpei Kawamoto, Jun Sakuma, and Hiroaki Kikuchi. 2013. Privacy-preservation for stochastic gradient descent application to secure logistic regression. In *The 27th Annual Conference of the Japanese Society for Artificial Intelligence*, Vol. 27. 1–4.

[43] Wei Xie, Yang Wang, Steven M Boker, and Donald E Brown. 2016. Privlogit: Efficient privacy-preserving logistic regression by tailoring numerical optimizers. *arXiv preprint arXiv:1611.01170* (2016).

[44] Runhua Xu, James BD Joshi, and Chao Li. 2019. CryptoNN: Training Neural Networks over Encrypted Data. *arXiv preprint arXiv:1904.07303* (2019).

[45] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. 2019. Federated machine learning: Concept and applications. *TIST* 10, 2 (2019), 12.

[46] Andrew C Yao. 1982. Protocols for secure computations. In *FOCS*. IEEE, 160–164.

[47] Jiawei Yuan and Shucheng Yu. 2013. Privacy preserving back-propagation neural network learning made practical with cloud computing. *TPDS* 25, 1 (2013), 212–221.

[48] Ya-Lin Zhang, Jun Zhou, Wenhao Zheng, Ji Feng, Longfei Li, Ziqi Liu, Ming Li, Zhiqiang Zhang, Chaochao Chen, Xiaolong Li, et al. 2019. Distributed Deep Forest and its Application to Automatic Detection of Cash-Out Fraud. *TIST* 10, 5 (2019), 55.

[49] Longfei Zheng, Chaochao Chen, Yingting Liu, Bingzhe Wu, Xibin Wu, Li Wang, Lei Wang, Jun Zhou, and Shuang Yang. 2020. Industrial scale privacy preserving deep neural network. *arXiv preprint arXiv:2003.05198* (2020).

[50] Jun Zhou, Xiaolong Li, Peilin Zhao, Chaochao Chen, Longfei Li, Xinxing Yang, Qing Cui, Jin Yu, Xu Chen, Yi Ding, et al. 2017. Kunpeng: Parameter server based distributed learning systems and its applications in alibaba and ant financial. In *SIGKDD*. ACM, 1693–1702.

A SECURITY DEFINITION

The two-party primitives considered in this paper pertain to the category of secure two-party computation. Specifically, secure two-party computation is a two-party random process. It maps pairs of inputs (one from each party) to pairs of outputs (one for each party), while preserving several security properties, such as correctness, privacy, and independence of inputs [24]. This random process is called *functionality*. Formally, denote a two-output functionality $f = (f_1, f_2)$ as $f : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}^* \times \{0,1\}^*$. For a pair of inputs (x, y), where x is from party P_1 and y is from party P_2, the output pair $(f_1(x, y), f_2(x, y))$ is a random variable. $f_1(x, y)$ is the output for P_1, and $f_2(x, y)$ is for P_2. During this process, neither party should learn anything more than its prescribed output.

Definition 1 (*Security in semi-honest model* [20]). Let $f = (f_1, f_2)$ be a *deterministic* functionality and π be a two-party protocol for computing f. Given the security parameter κ, and a pair of inputs (x, y) (where x is from P_1 and y is from P_2), the view of P_i ($i = 1, 2$) in the protocol π is denoted as $\text{view}_i^\pi(x, y, \kappa) = (w, r_i, m_i^1, \cdots, m_i^t)$, where $w \in \{x, y\}$, r_i is the randomness used by P_i, and m_i^j is the j-th message received by P_i; the output of P_i is denoted as $\text{output}_i^\pi(x, y, \kappa)$, and the joint output of the two parties is $\text{output}^\pi(x, y, \kappa) = (\text{output}_1^\pi(x, y, \kappa)), \text{output}_2^\pi(x, y, \kappa))$. We say that π securely computes f in semi-honest model if

- There exist probabilistic polynomial-time simulators $\mathcal{S}_1$ and $\mathcal{S}_2$, such that

$$\{\mathcal{S}_1(1^\kappa, x, f_1(x, y))\}_{x,y,\kappa} \cong \{\text{view}_1^\pi(x, y, \kappa)\}_{x,y,\kappa},$$
$$\{\mathcal{S}_2(1^\kappa, y, f_2(x, y))\}_{x,y,\kappa} \cong \{\text{view}_2^\pi(x, y, \kappa)\}_{x,y,\kappa}.$$

- The joint output and the functionality output satisfy

$$\{\text{output}^\pi(x, y, \kappa)\}_{x,y,\kappa} \cong \{f(x, y)\}_{x,y,\kappa},$$

where $x, y \in \{0, 1\}^*$, and $\cong$ denotes computationally indistinguishablity.

B SECURITY PROOF

B.1 Secret Sharing in Homomorphically Encrypted Field

Functionality of Secret Sharing in Homomorphically Encrypted Field $\mathcal{F}_{\text{SSHEF}}$

Inputs:
- $\mathcal{A}$ inputs a homomorphically encrypted matrix $[[\mathbf{Z}]]_b$ under $\mathcal{B}$'s public key pk_b;
- $\mathcal{B}$ inputs its secret key sk_b.

Outputs:
- $\mathcal{A}$ outputs a share $\langle \mathbf{Z} \rangle_1$;
- $\mathcal{B}$ outputs a share $\langle \mathbf{Z} \rangle_2$, where $\langle \mathbf{Z} \rangle_1 + \langle \mathbf{Z} \rangle_2 = \mathbf{Z}$.

Security Proof. We show that **Protocol 2** is secure against semi-honest adversaries. Formally, we have the following theorem.

Theorem 1. *Assume that the additively homomorphic crypto system $\Pi = (KeyGen, Enc, Dec)$ is indistinguishable under chosen-plaintext attacks. Then, **Protocol 2** (denoted as π_2) is secure in semi-honest model, as in **Definition 1**.*

PROOF. We begin by proving the correctness of **Protocol 2**, *i.e.,* we prove that $\langle \mathbf{Z} \rangle_1 + \langle \mathbf{Z} \rangle_2$ is equal to $\mathbf{Z}$. According to the protocol execution, $\mathcal{A}$ computes $[[\langle \mathbf{Z} \rangle_2]]_b = [[\mathbf{Z}]]_b - \langle \mathbf{Z} \rangle_1$. From the additive homomorphism of the crypto system Π, we can directly obtain the fact that the decryption of $[[\langle \mathbf{Z} \rangle_2]]_b$ is equal to $\mathbf{Z} - \langle \mathbf{Z} \rangle_1$.

Therefore, it holds that $\langle \mathbf{Z} \rangle_1 + \langle \mathbf{Z} \rangle_2 = \mathbf{Z}$. This proves the correctness of **Protocol 2**.

We now prove that we can construct two simulators $\mathcal{S}_\mathcal{A}$ and $\mathcal{S}_\mathcal{B}$, such that

$$\{\mathcal{S}_\mathcal{A}(1^\kappa, [[\mathbf{Z}]]_b, \langle \mathbf{Z} \rangle_1)\}_{[[\mathbf{Z}]]_b, sk_b, \kappa} \cong \{\text{view}_\mathcal{A}^{\pi_2}([[\mathbf{Z}]]_b, sk_b, \kappa)\}_{[[\mathbf{Z}]]_b, sk_b, \kappa}, \quad (2)$$

$$\{\mathcal{S}_\mathcal{B}(1^\kappa, sk_b, \langle \mathbf{Z} \rangle_2)\}_{[[\mathbf{Z}]]_b, sk_b, \kappa} \cong \{\text{view}_\mathcal{B}^{\pi_2}([[\mathbf{Z}]]_b, sk_b, \kappa)\}_{[[\mathbf{Z}]]_b, sk_b, \kappa}, \quad (3)$$

where $\text{view}_\mathcal{A}^{\pi_2}$ and $\text{view}_\mathcal{B}^{\pi_2}$ denotes the views of $\mathcal{A}$ and $\mathcal{B}$, respectively.

We prove the above equations for a corrupted $\mathcal{A}$ and a corrupted $\mathcal{B}$, respectively.

Corrupted $\mathcal{A}$. In this case, we construct a probabilistic polynomial-time simulator $\mathcal{S}_\mathcal{A}$ that, when given the security parameter κ, $\mathcal{A}$'s input $[[\mathbf{Z}]]_b$ and output $\langle \mathbf{Z} \rangle_1$, can simulate the view of $\mathcal{A}$ in the protocol execution. To this end, we first analyze $\mathcal{A}$'s view $\text{view}_\mathcal{A}^{\pi_2}([[\mathbf{Z}]]_b, sk_b, \kappa)$ in **Protocol 2**. In **Protocol 2**, $\mathcal{A}$ does not receive any messages from $\mathcal{B}$. Therefore, $\text{view}_\mathcal{A}^{\pi_2}([[\mathbf{Z}]]_b, sk_b, \kappa)$ consists of $\mathcal{A}$'s input $[[\mathbf{Z}]]_b$ and the randomness $r_\mathcal{A}$.

Given κ, $[[\mathbf{Z}]]_b$, and $\langle \mathbf{Z} \rangle_1$, $\mathcal{S}_\mathcal{A}$ simply generates a simulation of $\text{view}_\mathcal{A}^{\pi_2}([[\mathbf{Z}]]_b, sk_b, \kappa)$ by outputting $([[\mathbf{Z}]]_b, r_\mathcal{A})$. Therefore, we have the following two equations:

$$\text{view}_\mathcal{A}^{\pi_2}([[\mathbf{Z}]]_b, sk_b, \kappa) = ([[\mathbf{Z}]]_b, r_\mathcal{A}),$$
$$\mathcal{S}_\mathcal{A}(1^\kappa, [[\mathbf{Z}]]_b, \langle \mathbf{Z} \rangle_1) = ([[\mathbf{Z}]]_b, r_\mathcal{A}).$$

We note that the probability distributions of $\mathcal{A}$'s view and $\mathcal{S}_\mathcal{A}$'s output are identical. We thereby claim that Equation (1) holds.

This completes the proof in the case of corrupted $\mathcal{A}$.

Corrupted $\mathcal{B}$. In this case, we construct a probabilistic polynomial-time simulator $\mathcal{S}_\mathcal{B}$, when given the security parameter κ, $\mathcal{B}$'s input sk_b and output $\langle \mathbf{Z} \rangle_2$, can simulate the view of $\mathcal{B}$ in the protocol execution. To this end, we first analyze $\mathcal{B}$'s view $\text{view}_\mathcal{B}^{\pi_2}([[\mathbf{Z}]]_b, sk_b, \kappa)$ in **Protocol 2**. The only message obtained by $\mathcal{B}$ is the ciphertext $[[\langle \mathbf{Z} \rangle_2]]_b$. Therefore, $\text{view}_\mathcal{B}^{\pi_2}([[\mathbf{Z}]]_b, sk_b, \kappa)$ consists of $\mathcal{B}$'s input sk_b, the randomness $r_\mathcal{B}$, and the ciphertext $[[\langle \mathbf{Z} \rangle_2]]_b$.

Given κ, sk_b, and $\langle \mathbf{Z} \rangle_2$, $\mathcal{S}_\mathcal{B}$ generates a simulation of $\text{view}_\mathcal{B}^{\pi_2}([[\mathbf{Z}]]_b, sk_b, \kappa)$ as follows. It encrypts $\langle \mathbf{Z} \rangle_2$ with $\mathcal{B}$'s public key pk_b, and obtains $[[\langle \mathbf{Z} \rangle_2]]_b'$. Then, it generates $(sk_b, r_\mathcal{B}, [[\langle \mathbf{Z} \rangle_2]]_b')$ as the output. Therefore, we have the following two equations:

$$\text{view}_\mathcal{B}^{\pi_2}([[\mathbf{Z}]]_b, sk_b, \kappa) = (sk_b, r_\mathcal{B}, [[\langle \mathbf{Z} \rangle_2]]_b),$$
$$\mathcal{S}_\mathcal{B}(1^\kappa, sk_b, \langle \mathbf{Z} \rangle_2) = (sk_b, r_\mathcal{B}, [[\langle \mathbf{Z} \rangle_2]]_b').$$

We note that both $[[\langle \mathbf{Z} \rangle_2]]_b$ and $[[\langle \mathbf{Z} \rangle_2]]_b'$ are the ciphertexts of $\langle \mathbf{Z} \rangle_2$, and they look the same to $\mathcal{B}$. Therefore, the probability distributions of $\mathcal{B}$'s view and $\mathcal{S}_\mathcal{B}$'s output are identical. We thereby claim that Equation (2) holds.

This completes the proof in the case of corrupted $\mathcal{B}$.

In summary, **Protocol 2** securely computes $\mathcal{F}_{\text{SSHEF}}$ in semi-honest model.

□

B.2 Secure Sparse Matrix Multiplication

Functionality of Secure Sparse Matrix Multiplication $\mathcal{F}_{\mathsf{SSMM}}$

Inputs:
- $\mathcal{A}$ inputs a sparse matrix $\mathbf{X}$;
- $\mathcal{B}$ inputs a matrix $\mathbf{Y}$.

Outputs:
- $\mathcal{A}$ outputs a share $\mathbf{Z}_1$;
- $\mathcal{B}$ outputs a share $\mathbf{Z}_2$, where $\mathbf{Z}_1 + \mathbf{Z}_2 = \mathbf{X} \cdot \mathbf{Y}$.

Security Proof. We show that **Protocol 1** is secure against semi-honest adversaries. Formally, we have the following theorem.

Theorem 2. *Assume that the additively homomorphic crypto system* $\Pi = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *is indistinguishable under chosen-plaintext attacks. Then,* ***Protocol 1*** *(denoted as* π_1*) is secure against semi-honest adversaries in* $\mathcal{F}_{\mathsf{SSHEF}}$ *model, as in* ***Definition 1***.

PROOF. We begin by proving the correctness of **Protocol 1**, *i.e.,* we prove that $\mathbf{Z}_1 + \mathbf{Z}_2$ is equal to $\mathbf{X} \cdot \mathbf{Y}$. According to the protocol execution, $\mathcal{A}$ computes $[[\mathbf{Z}]]_b = \mathbf{X} \cdot [[\mathbf{Y}]]_b$. From the additive homomorphism of the cryptosystem Π, we know that $[[\mathbf{Z}]]_b = [[\mathbf{X} \cdot \mathbf{Y}]]_b$, *i.e.,* $\mathbf{Z} = \mathbf{X} \cdot \mathbf{Y}$. After invoking $\mathcal{F}_{\mathsf{SSHEF}}$, $\mathcal{A}$ and $\mathcal{B}$ obtain the shares $\mathbf{Z}_1$ and $\mathbf{Z}_2$ satisfying $\mathbf{Z}_1 + \mathbf{Z}_2 = \mathbf{Z}$.

Therefore, it holds that $\mathbf{Z}_1 + \mathbf{Z}_2 = \mathbf{X} \cdot \mathbf{Y}$. This proves the correctness of **Protocol 1**.

We now prove that we can construct two simulators $\mathcal{S}_{\mathcal{A}}$ and $\mathcal{S}_{\mathcal{B}}$, such that

$$\{\mathcal{S}_{\mathcal{A}}(1^{\kappa}, \mathbf{X}, \mathbf{Z}_1)\}_{\mathbf{X},\mathbf{Y},\kappa} \cong \{\mathsf{view}_{\mathcal{A}}^{\pi_1}(\mathbf{X}, \mathbf{Y}, \kappa)\}_{\mathbf{X},\mathbf{Y},\kappa}, \quad (4)$$

$$\{\mathcal{S}_{\mathcal{B}}(1^{\kappa}, \mathbf{Y}, \mathbf{Z}_2)\}_{\mathbf{X},\mathbf{Y},\kappa} \cong \{\mathsf{view}_{\mathcal{B}}^{\pi_1}(\mathbf{X}, \mathbf{Y}, \kappa)\}_{\mathbf{X},\mathbf{Y},\kappa}, \quad (5)$$

where $\mathsf{view}_{\mathcal{A}}^{\pi_1}$ and $\mathsf{view}_{\mathcal{B}}^{\pi_1}$ denotes the views of $\mathcal{A}$ and $\mathcal{B}$, respectively.

We prove the above equations for a corrupted $\mathcal{A}$ and a corrupted $\mathcal{B}$, respectively.

Corrupted $\mathcal{A}$. In this case, we construct a probabilistic polynomial-time simulator $\mathcal{S}_{\mathcal{A}}$ that, when given the security parameter κ, $\mathcal{A}$'s input $\mathbf{X}$ and output $\mathbf{Z}_1$, can simulate the view of $\mathcal{A}$ in the protocol execution. To this end, we first analyze $\mathcal{A}$'s view $\mathsf{view}_{\mathcal{A}}^{\pi_1}(\mathbf{X}, \mathbf{Y}, \kappa)$ in **Protocol 1**. In **Protocol 1**, the messages obtained by $\mathcal{A}$ are consisted of two parts. One is the ciphertext $[[\mathbf{Y}]]_b$; one is from the functionality $\mathcal{F}_{\mathsf{SSHEF}}$, *i.e.,* $\mathbf{Z}_1$. Therefore, $\mathsf{view}_{\mathcal{A}}^{\pi_1}(\mathbf{X}, \mathbf{Y}, \kappa)$ consists of $\mathcal{A}$'s input $\mathbf{X}$, the randomness $r_{\mathcal{A}}$, the ciphertext $[[\mathbf{Y}]]_b$, and $\mathbf{Z}_1$.

Given κ, $\mathbf{X}$, and $\mathbf{Z}_1$, $\mathcal{S}_{\mathcal{A}}$ generates a simulation of $\mathsf{view}_{\mathcal{A}}^{\pi_1}(\mathbf{X}, \mathbf{Y}, \kappa)$ as follows.

- $\mathcal{S}_{\mathcal{A}}$ randomly selects a matrix $\mathbf{Y}'$, encrypts it with pk_b and obtains $[[\mathbf{Y}']]_b$.
- $\mathcal{S}_{\mathcal{A}}$ simulates the functionality $\mathcal{F}_{\mathsf{SSHEF}}$ and takes $\mathbf{Z}_1$ as the output for $\mathcal{A}$ in $\mathcal{F}_{\mathsf{SSHEF}}$.
- $\mathcal{S}_{\mathcal{A}}$ generates a simulation of $\mathsf{view}_{\mathcal{A}}^{\pi_1}(\mathbf{X}, \mathbf{Y}, \kappa)$ by outputting $(\mathbf{X}, r_{\mathcal{A}}, [[\mathbf{Y}']]_b, \mathbf{Z}_1)$.

Therefore, we have the following two equations:

$$\mathsf{view}_{\mathcal{A}}^{\pi_1}(\mathbf{X}, \mathbf{Y}, \kappa) = (\mathbf{X}, r_{\mathcal{A}}, [[\mathbf{Y}]]_b, \mathbf{Z}_1),$$

$$\mathcal{S}_{\mathcal{A}}(1^{\kappa}, \mathbf{X}, \mathbf{Z}_1) = (\mathbf{X}, r_{\mathcal{A}}, [[\mathbf{Y}']]_b, \mathbf{Z}_1).$$

We note that as the additively homomorphic cryptosystem Π is indistinguishable under chosen-plaintext attacks, the probability distributions of $\mathcal{A}$'s view and $\mathcal{S}_{\mathcal{A}}$'s output are computationally indistinguishable. We thereby claim that Equation (3) holds.

This completes the proof in the case of corrupted $\mathcal{A}$.

Corrupted $\mathcal{B}$. In this case, we construct a probabilistic polynomial-time simulator $\mathcal{S}_{\mathcal{B}}$, when given the security parameter κ, $\mathcal{B}$'s input $\mathbf{Y}$ and output $\mathbf{Z}_2$, can simulate the view of $\mathcal{B}$ in the protocol execution. To this end, we first analyze $\mathcal{B}$'s view $\mathsf{view}_{\mathcal{B}}^{\pi_1}(\mathbf{X}, \mathbf{Y}, \kappa)$ in **Protocol 1**. The only message $\mathcal{B}$ receives is from the functionality $\mathcal{F}_{\mathsf{SSHEF}}$, *i.e.,* $\mathbf{Z}_2$. Therefore, $\mathsf{view}_{\mathcal{B}}^{\pi_1}(\mathbf{X}, \mathbf{Y}, \kappa)$ consists of $\mathcal{B}$'s input $\mathbf{Y}$, the randomness $r_{\mathcal{A}}$, and $\mathbf{Z}_2$.

Given κ, $\mathbf{Y}$, and $\mathbf{Z}_2$, $\mathcal{S}_{\mathcal{B}}$ simulates the functionality $\mathcal{F}_{\mathsf{SSHEF}}$ and takes $\mathbf{Z}_2$ as the output for $\mathcal{B}$ in $\mathcal{F}_{\mathsf{SSHEF}}$. Then $\mathcal{S}_{\mathcal{B}}$ generates a simulation of $\mathsf{view}_{\mathcal{B}}^{\pi_1}(\mathbf{X}, \mathbf{Y}, \kappa)$ by simply outputting $(\mathbf{Y}, r_{\mathcal{A}}, \mathbf{Z}_2)$. Therefore, we have the following two equations:

$$\mathsf{view}_{\mathcal{B}}^{\pi_1}(\mathbf{X}, \mathbf{Y}, \kappa) = (\mathbf{Y}, r_{\mathcal{B}}, \mathbf{Z}_2),$$

$$\mathcal{S}_{\mathcal{B}}(1^{\kappa}, \mathbf{X}, \mathbf{Z}_1) = (\mathbf{Y}, r_{\mathcal{B}}, \mathbf{Z}_2).$$

We note that the probability distributions of $\mathcal{B}$'s view and $\mathcal{S}_{\mathcal{B}}$'s output are identical. We thereby claim that Equation (4) holds.

This completes the proof in the case of corrupted $\mathcal{B}$.

In summary, **Protocol 1** securely computes $\mathcal{F}_{\mathsf{SSMM}}$ against semi-honest adversaries in $\mathcal{F}_{\mathsf{SSHEF}}$ model.

□