3-2021

Privacy-preserving federated deep learning with irregular users

Guowen XU
University of Electronic Science and Technology of China

Hongwei LI
University of Electronic Science and Technology of China

Yun ZHANG
University of Electronic Science and Technology of China

Shengmin XU
Singapore Management University, smxu@smu.edu.sg

Jianting NING
Singapore Management University, jtning@smu.edu.sg

See next page for additional authors

Citation

XU, Guowen; LI, Hongwei; ZHANG, Yun; XU, Shengmin; NING, Jianting; and DENG, Robert H.. Privacy-preserving federated deep learning with irregular users. (2021). *IEEE Transactions on Dependable and Secure Computing*. 19, (2), 1364-1381.
Available at: https://ink.library.smu.edu.sg/sis_research/5181

Privacy-Preserving Federated Deep Learning with Irregular Users

Guowen Xu, *Student Member, IEEE,* Hongwei Li (Corresponding author), *Senior Member, IEEE*, Yun Zhang, *Student Member, IEEE,* Shengmin Xu, Jianting Ning, *Member, IEEE*, and Robert H. Deng, *Fellow, IEEE*

Abstract—Federated deep learning has been widely used in various fields. To protect data privacy, many privacy-preserving approaches have been designed and implemented in various scenarios. However, existing works rarely consider a fundamental issue that the data shared by certain users (called *irregular users*) may be of low quality. Obviously, in a federated training process, data shared by many *irregular users* may impair the training accuracy, or worse, lead to the uselessness of the final model. In this paper, we propose PPFDL, a Privacy-Preserving Federated Deep Learning framework with *irregular users*. In specific, we design a novel solution to reduce the negative impact of *irregular users* on the training accuracy, which guarantees that the training results are mainly calculated from the contribution of high-quality data. Meanwhile, we exploit Yao's garbled circuits and additively homomorphic cryptosystems to ensure the confidentiality of all user-related information. Moreover, PPFDL is also robust to users dropping out during the whole implementation. This means that each user can be offline at any subprocess of training, as long as the remaining online users can still complete the training task. Extensive experiments demonstrate the superior performance of PPFDL in terms of training accuracy, computation, and communication overheads.

Index Terms—Privacy Protection, Federated Learning, Cloud Computing.

1 Introduction

Deep learning, as one of the most promising technologies, has been widely used in various aspects such as image classification, automatic driving, and smart healthcare. For example, intelligent image recognition systems have been widely deployed in public places such as airports and train stations. It has been demonstrated far more precision than humans in identifying suspected terrorists and detecting prohibited items. Based on patient-based medical data, deep learning-based regression techniques can aid in the diagnosis and prevention of certain diseases (e.g., hereditary and infectious diseases). It is clear that deep learning-based services are slowly changing our lives from travel, social, economy and many other aspects.

While deep learning has enormous potential benefits, traditional centralized deep learning usually requires a data processing center (e.g., the cloud server) to collect a large amount of users' data, and train Deep Neural Networks (DNNs) on the dataset before releasing the service interface. However, users' data may be sensitive or contain private information, e.g., personal medical records, location information, and social relationships. To gather this information on a third-party service provider (such as the cloud) will inevitably lead to considerable privacy breaches [1], [2]. Recently, federated deep learning (also known as distributed or decentralized deep learning) is proposed to alleviate privacy breaches in the process of training neural networks, as it supports neural network optimization by only sharing parameters between users and servers, rather than sharing users' raw data. Because of this nature, federated deep learning has received widespread attention from industry and academia, and various distributed learning architectures [3], [4], [5] have been proposed to serve specific scenarios.

Although federated learning eases concerns about privacy leaks to some extent, recent works [6], [7] have shown that an adversary, such as the cloud server, can still recover target data (e.g., data tabs, memberships, etc) by exploiting the shared gradients and global parameters. Moreover, in addition to privacy protection, how to ensure the high quality of data during training is another fundamental issue. In real-life scenarios, the quality of the raw data held by each user is usually uneven. Users with superior expertise or terminal devices usually generate high-quality data while others may hold low-quality data. In this paper, we call these users with low-quality data as *irregular users* (refer to Section 2.1 for more details). Obviously, in the federated training process, data shared by *irregular users* may impair the training accuracy, or worse, lead to the uselessness of the final model. For practicality and improving training accuracy, it is also essential to design strategies to eliminate

- *Guowen Xu and Yun Zhang are with the school of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China. Guowen Xu is also with the Cyberspace Security Research Center, Peng Cheng Laboratory, Shenzhen, 518000, China. (e-mail: guowen.xu@foxmail.com; yunzhang505@foxmail.com).*
- *Hongwei Li is with the school of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China. Hongwei Li is also with the Cyberspace Security Research Center, Peng Cheng Laboratory, Shenzhen, 518000, China (e-mail: hongweili@uestc.edu.cn)*
- *Shengmin Xu and Jianting Ning are with the School of Information Systems, Singapore Management University, 178902 Singapore (e-mail: smxu@smu.edu.sg; jtning88@gmail.com)*
- *Robert H. Deng is with the School of Information Systems, Singapore Management University, 178902 Singapore (e-mail:robertdeng@smu.edu.sg)*

the impact of low-quality data on the training process.

To address the privacy leakage in the federated training process, many remarkable solutions have been proposed and applied to various scenarios [3], [4], [5], [8], [9]. In summary, existing privacy-preserving federated deep learning approaches are mostly evolved from three underlying techniques: *Differential Privacy* [3], [8], *Homomorphic Encryption* [10], [11] and *Secure Multi-Party Computation (SMC)* [12], [13]. However, all of the above solutions do not consider the fundamental issue of *irregular users*. i.e., all of them assume that the data held by each user is high-quality and similar to each other. To our best knowledge, only *Zhao et al.* [14] proposed SecProbe, the first privacy-preserving approach while reducing the impact of *irregular users* on training accuracy. The authors exploited differential privacy based technologies to perturb the objective function of target DNNs, and adopted a custom data detection mechanism [15] to ensure a good balance between accuracy and security. However, the results [6], [16] have shown that current mechanisms for differentially private deep learning rarely offer acceptable utility-privacy trade-offs for complex learning tasks: the adversary can still easily recover the user's sensitive data if the model is acceptable for accuracy. Moreover, to make the learning process fair and non-discriminative, the information of " data quality" of each user (called user's reliability) should be also kept confidential, and be not derived by the server and any users during the training process. However, in SecProbe, each user's reliability is accessible to the server.

To address the above issues, in this paper, we propose PPFDL, an efficient and Privacy-Preserving Federated Deep Learning framework while maintaining a high data utility. In summary, the contributions of PPFDL can be summarized as follows:

- We propose an efficient and privacy-preserving data aggregation framework in the federated training process, by highly integrating the technologies of additive homomorphism and Yao's garbled circuits. The framework protects the privacy of all user-related information, including the gradient and reliability of each user, as well as the aggregation results.
- We design a novel strategy (called $\mathtt{Meth}_{\mathtt{IU}}$) to reduce the negative impact of irregular users on the training accuracy. Since most of the computations in $\mathtt{Meth}_{\mathtt{IU}}$ are done by the servers, $\mathtt{Meth}_{\mathtt{IU}}$ is very friendly for end-users with limited computing power. Moreover, for practicality, our PPFDL is also robust to users dropping out during the entire training process for various unpredictable reasons.
- We give a comprehensive security analysis to prove the high security of our PPFDL. Besides, extensive experiments conducted on real-world data demonstrate the high performance of PPFDL in terms of training accuracy, computation, and communication overheads.

The remainder of this paper is organized as follows. In Section 2, we describe the problem statement and preliminaries. In Section 3 and Section 4, we give the details of PPFDL and analyze its security. Next, performance evaluation and related works are discussed in Section 5 and Section 6, respectively. Finally, Section 7 concludes the paper.

2 PROBLEM STATEMENT AND PRELIMINARIES

2.1 System Model

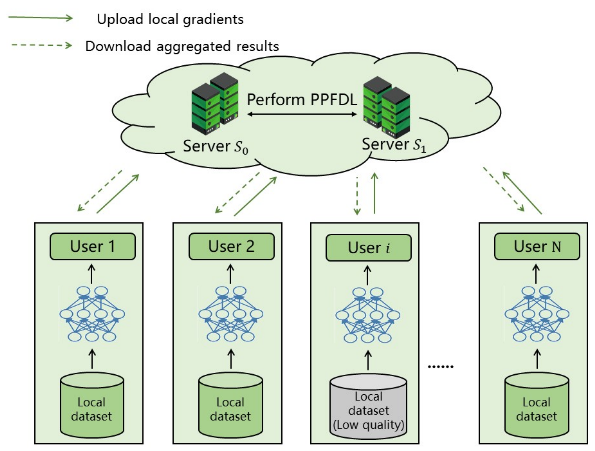


Fig. 1: System Model

As shown in Fig. 1, we consider two generic entities in our system model, i.e., *servers* and *users*, who work together to achieve privacy-preserving federated training. Specifically, all participating users first agree on a unified DNN. Then, in each epoch, every user trains the DNN with its local dataset, and computes the gradient corresponding to each training sample. To speed up convergence and improve the accuracy of training, each user encrypts its local gradients and submits them to the cloud. Next, two non-collusive servers (i.e., $\boldsymbol{S_0}$ and $\boldsymbol{S_1}$) interactively execute PPFDL to obtain ciphertexts of aggregated results (i.e., the aggregated value of gradients), and return the results to all users. In the end, each user decrypts the ciphertext and updates the parameters of the local DNN. To obtain a satisfactory network structure, the two servers and all users iteratively perform the above operations until the DNN satisfies the predefined optimization condition.

Remark: We consider two servers $\boldsymbol{S_0}$ and $\boldsymbol{S_1}$ in our system, where $\boldsymbol{S_0}$ serves as a platform that receives encrypted data from users, while $\boldsymbol{S_1}$ provides computation assistance to $\boldsymbol{S_1}$ for executing PPFDL. Such server-assisted setting has been formalized and widely used in outsourced computing scenarios [17], [18], [19]. It has also been utilized in previous works on privacy-preserving machine learning [11], [19], [20]. Two important advantages of this setting are that (i) users can alleviate the pressure on local resources by outsourcing complex computing to the two servers. In PPFDL, in addition to supporting privacy-preserving deep learning, we introduce a method called $\mathtt{Meth}_{\mathtt{IU}}$ (shown in Section 2.6) to reduce the effect of low-quality data in the federated training process. This will inevitably result in unbearable computing overhead if all of the operations are performed by edge devices. Indeed, previous works also designed privately deep learning method [10], [12] under a single server setting. However, with the fact of long-term multiple interactions of training a neural network, frequent interactions between users and the server will inevitably result in huge communication overhead. Besides, in the interaction process, how to deal with users offline due to abnormal conditions is also a problem to be considered. In our two servers setting, each user is only required to send its encrypted gradient set to the server. Most of ciphertext processes are done by two servers interacting with each other. As a result, this setting is very friendly for end-users with limited computing power, and can withstand some users dropping out for unpredictable reasons, such as network problems and machine failures. (ii) the two non-collusive servers setting can be considered as a secure two-party calculation (2PC), which can benefit from a combination of efficient techniques for boolean computation such as garbled circuits and OT-extension [11]. In Section 5, we also conduct experiments to confirm the superiority of our proposed model in terms of computation and communication overhead.

In this paper, we consider an important practical problem of *irregular users* (e.g., user i shown in Fig. 1) in the training process. Concretely, different from previous literature assuming that all users are *regular*, that is, the data held by each user is high-quality and similar to each other, we assume that there are a small group of *irregular users*. This means that a portion of data held by these *irregular users* is not always as accurate as others, and thus the gradients they upload may impair the accuracy of the training results. In real life, it is very common to have *irregular users* in a federated deep learning system. Considering the scenario where multiple hospitals hope to learn a unified neural network model in a distributed manner, thereby using this model to predict the incidence of cancer. There may be a non-negligible gap of data quality among different hospitals, since the data quality provided by the chief doctor in the top hospital with advanced equipment is often higher than that from community hospitals. Actually, each user may have partially "poor" data as more and more data is collected into local data sets. There are too many possibilities (i.e., recording errors, stale data, etc) to generate these low-quality data in the data generation and storage procedures. As a result, this may harm the accuracy of the model if the low-quality data of these *irregular users* is involved in the training process. In this paper, we design a novel strategy to reduce the negative impact of irregular users on the accuracy of training results. Moreover, our strategy is computationally friendly to the users since most of computations are done by servers. In Section 3 and Section 4 we will describe how our solution can accomplish the above requirements.

2.2 Threat Model and Privacy Requirements

In PPFDL, we consider that the two servers are the main adversaries since they hold all the ciphertext of the user's local gradients. In our threat model, we assume that the two servers are *honest-but-curious*, which means that each server honestly abides by the pre-agreed procedures to complete its mission. However, it may also try to compromise users' data privacy by utilizing mastered prior knowledge. Besides, we assume that there is no collusion between the two servers $\boldsymbol{S_0}$ and $\boldsymbol{S_1}$. Such an assumption has been widely used in outsourced computing scenarios [11], [17], [21], such as the state-of-the-art work SecureML [11], which also uses two non-collusive servers to implement the privacy-preserving DNNs training. Besides, since the user is the requester of the cloud service, every user is considered to be trustworthy and will not collude with servers.

Under the above threat model, we formulate the privacy requirements as follows.

- *Confidentiality of user's local gradients*: An adversary, such as the cloud server, may recover users' sensitive information such as data tabs and memberships by exploiting the shared gradients and global parameters. To protect users' privacy, each user's local gradient should be encrypted before being sent to the server.
- *Privacy protection of user's reliability and aggregated result*: To make the learning process fair and non-discriminative, each user's reliability, that is, the information of " data quality" of the user, should be kept confidential and be not derived by servers and any users during the training process. On the other hand, the aggregated results can be regarded as valuable intellectual properties, which are generated with a lot of resources, and even contain some user's proprietary information. Hence, the aggregation results should be confidential to adversaries (such as the cloud) except for the users participating in the training.

2.3 Neural Network and Federated Deep Learning

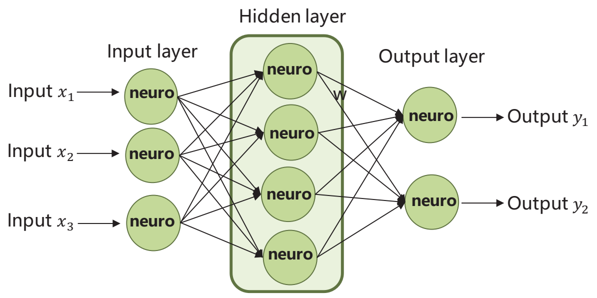


Fig. 2: An example of a fully connected neural network

Deep Neural Networks (DNNs), as the underlying structure of deep learning-based techniques, have spawned a variety of structures that serve differnet scenarios. At a high-level, a traditional neural network usually consists of one input layer, one or more hidden layers, and one output layer. Fig. 2 shows an example of a simple fully connected neural network, which contains one input layer, one hidden layer, and one output layer. *Fully connected* means that the neurons (i.e., cycles in Fig. 2) between two adjacent layers are connected by parameters (denoted as $\boldsymbol{\omega}$ in this paper). As described in Fig. 2, given an input $\boldsymbol{x} = \{x_1, x_2, x_3\}$, the output of the DNN can be denoted as $f(\boldsymbol{x}, \boldsymbol{\omega}) = \hat{\boldsymbol{y}} = \{y_1, y_2\}$, where f is the output function of the DNN. Hence, the goal of training a DNN is to find the optimal parameters $\boldsymbol{\omega}$ that accurately reflect the relationship between $\boldsymbol{x}$ and $\boldsymbol{y}$, and ultimately make the DNN output $\hat{\boldsymbol{y}}$ infinitely close to the real label $\boldsymbol{y}$.

Specifically, given a training set $\mathcal{D} = \{(\boldsymbol{x_i}, \boldsymbol{y_i}); i = 1, 2, \cdots, M\}$, we first define a loss on the training set as $\mathcal{L}_f(\mathcal{D}, \boldsymbol{\omega}) = \frac{1}{|\mathcal{D}|}\sum_{\langle \boldsymbol{x_i}, \boldsymbol{y_i}\rangle \in \mathcal{D}} \mathcal{L}_f(\boldsymbol{x_i}, \boldsymbol{y_i}, \boldsymbol{\omega})$, where $\mathcal{L}_f(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\omega}) = l(\boldsymbol{y}, f(\boldsymbol{x}, \boldsymbol{\omega}))$ for a loss function l, e.g., $l(\boldsymbol{y}, f(\boldsymbol{x}, \boldsymbol{\omega})) = ||\boldsymbol{y} - f(\boldsymbol{x}, \boldsymbol{\omega})||_2$, where $||\cdot||_2$ is the l_2 norm of a vector. Then, to minimize the function $\mathcal{L}_f(\mathcal{D}, \boldsymbol{\omega})$, we use a minibatch stochastic gradient descent rule [12], [22] to iteratively find the optimal parameters $\boldsymbol{\omega}$ as below,

$$\boldsymbol{\omega}^{j+1} \leftarrow \boldsymbol{\omega}^j - \eta \nabla \mathcal{L}_f(\mathcal{D}^j, \boldsymbol{\omega}^j) \tag{1}$$

where $\boldsymbol{\omega}^j$ represents the parameters after j-th iteration. $\mathcal{D}^j \subseteq \mathcal{D}$ is a subset of training set $\mathcal{D}$. η represents the learning rate, and $\nabla \mathcal{L}_f(\mathcal{D}^j, \boldsymbol{\omega}^j)$ denotes the partial derivative of the parameter $\boldsymbol{\omega}^j$ on the set $\mathcal{D}^j$.

In the federated training, assuming that each user $n \in N$ possesses a local data set $\mathcal{D}_n$, then the whole training data set $\mathcal{D}$ can be denoted as $\mathcal{D} = \cup_{n\in N}\mathcal{D}_n$. To run the stochastic gradient descent algorithm, at j-th iteration, each user n first randomly selects a subset $\mathcal{D}_n^j$. Next, each user n computes $\boldsymbol{x}_n^j = |\mathcal{D}_n^j| \nabla \mathcal{L}_f(\mathcal{D}_n^j, \boldsymbol{\omega}^j)$. As a result, the minibatch loss gradient $\nabla \mathcal{L}_f(\mathcal{D}^j, \boldsymbol{\omega}^j)$ can be rewritten as

$$\nabla \mathcal{L}_f(\mathcal{D}^j, \boldsymbol{\omega}^j) = \frac{1}{|\mathcal{D}^j|}\sum_{n\in N} \boldsymbol{x}_n^j \tag{2}$$

where $\mathcal{D}^j = \cup_{n\in N}\mathcal{D}_n^j$. Then, each user shares just the concatenated vector $[|\mathcal{D}_n^j|]||\boldsymbol{x}_n^j$ to the cloud server, from which the cloud server can update the parameters (i.e., compute the Eqn.(1)) as

$$\boldsymbol{\omega}^{j+1} \leftarrow \boldsymbol{\omega}^j - \eta \frac{\sum_{n\in N}\boldsymbol{x}_n^j}{\sum_{n\in N}|\mathcal{D}_n^j|} \tag{3}$$

In the end, the cloud server returns $\boldsymbol{\omega}^{j+1}$ to all users for updating the local network. The server and users iteratively perform the above operations until the DNN's output reaches the predetermined precision range.

Remark: Based on Eqn.(3), we can see that in the federated training, the main task of the cloud server is to aggregate all of $\boldsymbol{x}_n^j$ (For simplicity, we call $\boldsymbol{x}_n^j$ as the local gradients of user n at j-th iteration), i.e., compute aggregated value $\frac{\sum_{n\in N}\boldsymbol{x}_n^j}{\sum_{n\in N}|\mathcal{D}_n^j|}$. Then, each user can update its local DNN based on Eqn.(3). However, the above process does not consider that the quality of data held by different users may be different, i.e., it assumes that the data held by all users is of high quality or similar distribution. In Section 2.6, we will introduce a novel method to calculate the reliability of different users, thereby reducing the proportion of data provided by *irregular users* in the aggregation process. Based on this, In Section 3 and Section 4, we will provide two methods to compute the above aggregation in a secure way, which guarantees that each user's local gradients are protected while maintaining a high data utility.

2.4 Yao's Garbled Circuits

Yao's garbled circuit enables two parties $\mathcal{A}$ and $\mathcal{B}$ to privately calculate any polynomial function $g(a, b)$ without revealing their secrets [23], where $\mathcal{A}$ and $\mathcal{B}$ hold secrets a and b, respectively. At a high level, the Yao's garbled circuit protocol works as follows: Firstly, assume that the party $\mathcal{A}$ is the generator, who creates the garbled circuit GC of the targeted function $g(\cdot, \cdot)$, and then sends GC along with the garbled value $\hat{a}$ of input a to the party $\mathcal{B}$ (we call the party $\mathcal{B}$ as evaluator). Next, the evaluator $\mathcal{B}$ runs a 1-out-of-2 oblivious transfer (OT_2^1) [24] protocol with $\mathcal{A}$ to receive its garbled value $\hat{b}$ of input b. Once obtaining the garbled value $\hat{a}$ and $\hat{b}$, party $\mathcal{B}$ executes the GC and gets the result $g(a, b)$. For more details of Yao's garbled circuit, please refer to literature [23], [24].

2.5 Additively Homomorphic Cryptosystem

In general, a public-key encryption scheme can be denoted as $\mathcal{HE} = (KenGen, Enc, Dec)$, where $KenGen$ generates the key pair used in $\mathcal{HE}$, Enc and Dec denote the encryption and decryption algorithms, respectively. Then, we say $\mathcal{HE}$ is additively homomorphic if it satisfies the following properties: Firstly, given two plaintexts x_1 and x_2 and their ciphertexts $Enc_{pk}(x_1)$ and $Enc_{pk}(x_2)$ with the same public key pk, $\mathcal{HE}$ holds that $Enc_{pk}(x_1) \cdot Enc_{pk}(x_2) = Enc_{pk}(x_1 + x_2)$. That is, given two ciphertexts of x_1 and x_2, $\mathcal{HE}$ can directly compute the ciphertext of the sum of the two plaintexts x_1 and x_2, without requiring the secret key sk corresponding to public key pk. Secondly, given a plaintext x_1 and a constant t, $\mathcal{HE}$ holds that $Enc_{pk}(x_1)^t = Enc_{pk}(t \cdot x_1)$. That is, with the ciphertext $Enc_{pk}(x_1)$ and the constant t, $\mathcal{HE}$ can directly compute the ciphertext of the multiplication of x_1 and t.

2.6 Method of Handling Irregular Users

We introduce our method called $\mathtt{Meth_{IU}}$ to reduce the effect of low-quality data in the federated training process. In this paper, we only consider the federated learning scenario where each user holds independent and identically distributed (IID) samples. Specifically, Let N be the number of users participating in the federated training, where each user holds M different types of gradients after performing gradient descent algorithm (refer to Section 2.3) under the local data set. In the federated training, each user i will upload its local gradients to the cloud, and receive the aggregated results to update the local DNN. Without loss of generality, we use the symbol x_n^m to represent the m-th gradient submitted by the user n, and the aggregated result for gradient m is denoted as x_*^m.

With these notations, we now describe the technical details of $\mathtt{Meth_{IU}}$. $\mathtt{Meth_{IU}}$ is derived from the traditional truth discovery algorithm [25]. Truth discovery has shown excellent performance in estimating real data from numerous heterogeneous data and widely applied in various scenarios, such as healthcare, transportation and crowdsourcing systems [26]. Specifically, $\mathtt{Meth_{IU}}$ begins with the initialization of the aggregated values for each type of gradient m, i.e., the cloud server first initializes x_*^m and sends them to each user, where x_*^m can be computed through the experience of the server or a public auxiliary database. Let $\mathcal{T}_i$ be the reliability of user i, that is, the information of "data quality" of the user, $\mathtt{Meth_{IU}}$ is divided into **`User's Reliability Update`** and **`Aggregated Value Update`** two parts as below.

2.6.1 *User's Reliability Update*

Given the estimated aggregated value x_*^m, each user's reliability $\mathcal{T}_i$ can be updated as follows:

$$\mathcal{T}_i = \frac{\mathcal{C}}{\sum_{m=1}^{m=M} d(x_i^m, x_*^m)} \tag{4}$$

where $\mathcal{C} = \chi^2_{(1-\alpha/2,|M|)}$, is a coefficient to scale up the user's reliability. χ represents the Chi-squared distribution and α is the corresponding significance level. Note that the coefficient $\mathcal{C} = \chi^2_{(1-\alpha/2,|M|)}$ can be considered as a public parameter once given the value of α and the number of gradients. $d(\cdot)$ is the function to measure the distance between x_i^m and x_*^m. Based on the results of existing works [27], [28], if each user holds IID samples, gradients of the same type generated from disparate data may differ in scalars, however, if these data are good for training, they are always consistent with a high probability in the direction of the vector. In practice, we require to always keep the direction of the local gradient x_i^m consistent with the global gradient x_*^m, so as to ensure the convergence of training in optimization. Hence, we calculate $d(x_i^m, x_*^m) = (x_i^m - x_*^m)^2$ if x_i^m and x_*^m are consistent on the sign of the value, otherwise, $d(x_i^m, x_*^m)$ is set to a large positive integer (described in Section 3).

Remark: Based on Eqn.(4), for any two gradients with same sign of the value, we can see that if a gradient provided by a user is closer to aggregated values, this user will be assigned higher reliability. This is reasonable. The state-of-the-art work [29] has shown that data with similar quality has statistically similar properties on gradients, i.e., gradients calculated under them are very similar in value. However, for scenarios where each user holds non-IID samples, it may be problematic to directly use $\mathtt{Meth_{IU}}$ to reduce the proportion of gradients of users who are inconsistent with the global gradient in the aggregation process. Since each user collect data based on its local environment and usage pattern, it is inevitable that there will be some differences in the distribution of data collected between different users. In this context, using to punish irregular users may lead to overfitting of the model in accuracy. However, to the best of our knowledge, all existing privacy-preserving federated learning are designed based on the assumption that users hold IID samples [3], [7], [10], [11], [14]. Although some federated learning approaches with non-IID data have been proposed [30], [31], [32], it is unclear whether these schemes can be easily converted into cipher text form. We leave them as one of the future works.

2.6.2 *Aggregated Value Update*

Given the reliability $\mathcal{T}_i$ of each user, the aggregated value for each type of gradients is updated as follows:

$$x_*^m = \frac{\sum_{i=1}^{i=N} \mathcal{T}_i x_i^m}{\sum_{i=1}^{i=N} \mathcal{T}_i} \tag{5}$$

Based on Eqn.(5), we can see that if a user is assigned a high reliability, the gradients of this user will be counted more in the aggregation process. This ensures that the aggregated value is calculated primarily based on the data submitted by regular users.

Remark: In practical scenarios, we can treat user i as an irregular user if $\frac{\mathcal{T}_i - \min_{j\in[1,n]} \mathcal{T}_j}{\max_{s\in[1,n]} \mathcal{T}_s - \min_{j\in[1,n]} \mathcal{T}_j} < \delta$ under the current data set, where δ is the threshold negotiated by users. Such a setting is also adopted in works [27], [29]. Please note that we do not remove gradients generated by irregular users through the above guideline, but only reduce their contribution during the training process. One reason we do this is that the reliability information of users in the PPFDL is kept confidential to all parties (including the users themselves), which is mainly to ensure the non-discrimination of training. Therefore, each participant cannot know the reliability of every user, which also makes it impossible to remove the data of irregular users. On the other hand, as described in Section 2.1, it is inevitable that a part of low-quality data (also considered random noises) involved in realistic neural network training. Based on such facts, i.e., models trained in rare benign events may appear to overfit in actual predictions [22], [33], [34], in our PPFDL, we do not remove gradients generated by irregular users, but only ensures that the aggregated value is calculated primarily based on the data submitted by regular users. In Section 5, we will demonstrate the superiority of this approach in terms of prediction accuracy.

In the next two sections, we will present our Privacy-Preserving Federated Deep Learning (PPFDL) approach, which consists of two schemes, called $\mathtt{PPFDL_{bsc}}$ and $\mathtt{PPFDL_{imd}}$. The goal of PPFDL is to perform $\mathtt{Meth_{IU}}$ over encrypted data, thereby achieving the expected privacy-preserving level while maintaining a high data utility.

3 BASIC SCHEME

In this section, we present the technical details of $\mathtt{PPFDL_{bsc}}$, which enables servers and users to conduct the encrypted $\mathtt{Meth_{IU}}$ over ciphertext domain, and achieve the expected privacy requirements, i.e., privacy protection of user's local gradients and reliability, and the aggregated results. However, the communication complexity between $\boldsymbol{S_0}$ and $\boldsymbol{S_1}$ is not optimal. In Section 4, we give an improved scheme $\mathtt{PPFDL_{imd}}$ to show how to address this problem.

3.1 Overview

$\mathtt{PPFDL_{bsc}}$ resorts to a hybrid approach to building a bridge between additively homomorphic encryption and garbled circuits. Specifically, we use additively homomorphic encryption such as Paillier cryptosystem [35] to perform all the aggregations required in the $\mathtt{Meth_{IU}}$. After that, the garbled circuit is applied to perform encrypted division operation. In addition, multiplication is also required in the $\mathtt{Meth_{IU}}$ (e.g., computing squared distance between local gradients and aggregated values in Eqn.(4), and the product of user's reliability and aggregate value in Eqn.(5)). As a result, we present an encrypted multiplication protocol by simply using homomorphic encryption primitives. However, we will point out that this protocol is not optimal because it incurs high communication complexity between $\boldsymbol{S_0}$ and $\boldsymbol{S_1}$. This prompts us to propose an improved protocol (shown in Section 4) with optimal communication complexity. Note that we do not prefer garbled circuits to construct the secure multiplication protocol. The main reason is that garbled circuits always require excessive gates even to calculate a simple multiplication function.

3.2 Building Blocks

Before describing the technical details of $\mathtt{PPFDL_{bsc}}$, we first present two building blocks used in our construction.

3.2.1 Secure Division Protocol

We first describe how to construct a secure division protocol (called `SecDiv`) based on garbled circuits, whose goal is to have the two servers (i.e., $\boldsymbol{S_0}$ and $\boldsymbol{S_1}$) get the result of the division without knowing each other's secrets. Specifically, assume that server $\boldsymbol{S_0}$ has $Enc_{pk_1}(x_1)$ and $Enc_{pk_1}(x_2)$, while $\boldsymbol{S_1}$ has the secret key sk_1. The goal of $\boldsymbol{S_0}$ is to obtain $Enc_{pk_1}(x_1/x_2)$ without knowing x_1, x_2 and x_1/x_2. The specific steps of `SecDiv` are as follows: Firstly, $\boldsymbol{S_0}$ obscures $Enc_{pk_1}(x_1)$ and $Enc_{pk_1}(x_2)$ by randomly adding noises in the ciphertexts, and sends the obscured values x_1' and x_2' to $\boldsymbol{S_1}$. $\boldsymbol{S_1}$ then constructs a garbled circuit `GC` and sends part of the garbled inputs to $\boldsymbol{S_0}$. Next, $\boldsymbol{S_0}$ runs an oblivious transfer (OT) protocol with $\boldsymbol{S_0}$ to get the additional garbled values. Finally, $\boldsymbol{S_0}$ evaluates the `GC` to get the desired results. Fig. 3 shows the details of `SecDiv`.

Proposition 1: The `SecDiv` *holds the security property that* $\boldsymbol{S_0}$ *and* $\boldsymbol{S_1}$ *learn nothing about the values of* x_1, x_2 *and* x_1/x_2.

Proof: As shown in Fig. 3, for $\boldsymbol{S_1}$, we can see that it only observes the obscured values d_1 and d_2 (i.e., $d_1 = x_1 + h_1$, $d_2 = x_2 + h_2$), which is indistinguishable from random values as long as we randomly select h_1 and h_2 in a large domain. For $\boldsymbol{S_0}$, it can observe the encrypted values $Enc_{pk_1}(x_1)$, $Enc_{pk_1}(x_2)$ and $Enc_{pk_1}(r)$ under the Paillier homomorphic scheme, and the information exposed during the garbled circuit process. The security of the garbled circuit has been formally proven in [36], which shows that no information will be revealed in execution except for the final output. The final output produced at $\boldsymbol{S_0}$ is also a obscured value $x_1/x_2 - r$, which is also indistinguishable from random values. Therefore, the view of each server can be simulated so that both two servers learn nothing about the values of x_1, x_2 and x_1/x_2.

Remark: The standard integer division function described above can be easily implemented through existing advanced program frameworks [37], [38], [39]. In this paper, we adopt the expressive ObliVM-lang programming framework [38] to perform `SecDiv`. We know that there are many solutions for optimizing garbled circuits, and the principles they follow are also not identical. However, In this paper, we mainly focus on the secure implementation of federated training. We emphasize that optimizing garbled circuits is an orthogonal direction that is beyond the scope of this paper.

```
Input: $S_0$ holds $Enc_{pk_1}(x_1)$ and $Enc_{pk_1}(x_2)$; $S_1$ has the
       secret key $sk_1$.
Output: $S_0$ obtains $Enc_{pk_1}(x_1/x_2)$.
Procedure:
$S_0$:
    1. Randomly select two integers $h_1$ and $h_2$, and calculate
       $x_1' \leftarrow Enc_{pk_1}(x_1) \cdot Enc_{pk_1}(h_1)$ and
       $x_2' \leftarrow Enc_{pk_1}(x_2) \cdot Enc_{pk_1}(h_2)$.
    2. Send $x_1'$ and $x_2'$ to $S_1$.
$S_1$:
    3. $d_1 = Dec_{sk_1}(x_1')$ and $d_2 = Dec_{sk_1}(x_2')$
    4. Construct a garbled circuit GC taking as input the
       garbled values of $d_1$, $d_2$, $h_1$, $h_2$ and $r$, where $r$ is a
       random integer selected by $S_1$.
    5. Send GC and the garbled values of $d_1$, $d_2$, $r$ and
       $Enc_{pk_1}(r)$ to $S_0$.
$S_0, S_1$:
    6. $S_0$ runs an OT protocol with $S_1$ to get the garbled
       values of $h_1$ and $h_2$.
$S_0$:
    7. Run the GC to get $x_1/x_2 - r$.
    8. Calculate $Enc_{pk_1}(x_1/x_2) = Enc_{pk_1}(x_1/x_2 - r) \cdot$
       $Enc_{pk_1}(r)$.
```

Fig. 3: `SecDiv`: Secure division protocol

3.2.2 Secure Multiplication Protocol

To support the multiplication over the encrypted data, we present an encrypted multiplication protocol by simply using homomorphic encryption primitives. Fig. 4 shows technical details of the protocol (called `SecMul`), which enables $\boldsymbol{S_0}$ holding $Enc_{pk_1}(x_1)$ and $Enc_{pk_1}(x_2)$ to calculate $Enc_{pk_1}(x_1 \cdot x_2)$ without knowing x_1 and x_2.

Proposition 2: The `SecMul` *holds the security property that* $\boldsymbol{S_0}$ *and* $\boldsymbol{S_1}$ *learn nothing about the values of* x_1, x_2 *and* $x_1 \cdot x_2$.

Proof: As shown in Fig. 4, for $\boldsymbol{S_0}$, we can see that it only observes the encrypted values $Enc_{pk_1}(x_1)$, $Enc_{pk_1}(x_2)$ and $Enc_{pk_1}(d)$ under semantically-secure homomorphic cryptosystem, from which $\boldsymbol{S_0}$ cannot get any useful information if it does not have the secret key. For $\boldsymbol{S_1}$, it only observes the obscured values of d_1 and d_2, which is also indistinguishable from random values. Therefore, the `SecMul` holds the security property that $\boldsymbol{S_0}$ and $\boldsymbol{S_1}$ learn nothing about the values of x_1, x_2 and $x_1 \cdot x_2$.

```
Input: $S_0$ holds $Enc_{pk_1}(x_1)$ and $Enc_{pk_1}(x_2)$; $S_1$ has the
       secret key $sk_1$.
Output: $S_0$ obtains $Enc_{pk_1}(x_1 \cdot x_2)$.
Procedure:
$S_0$:
     1. Randomly select two integers $h_1$ and $h_2$.
     2. $x_1' \leftarrow Enc_{pk_1}(x_1) \cdot Enc_{pk_1}(h_1)$.
     3. $x_2' \leftarrow Enc_{pk_1}(x_2) \cdot Enc_{pk_1}(h_2)$.
     4. Send $x_1'$ and $x_2'$ to $S_1$.
$S_1$:
     5. $d_1 = Dec_{sk_1}(x_1')$ and $d_2 = Dec_{sk_1}(x_2')$.
     6. $d = d_1 \cdot d_2$.
     7. $d' \leftarrow Enc_{pk_1}(d)$.
     8. Send $d'$ to $S_0$.
$S_0$:
     9. $s \leftarrow d' \cdot Enc_{pk_1}(x_1)^{-h_1} \cdot Enc_{pk_1}(x_2)^{-h_2}$.
    10. $Enc_{pk_1}(x_1 \cdot x_2) \leftarrow s \cdot Enc_{pk_1}(-h_1 \cdot h_2)$.
```

Fig. 4: `SecMul`: Secure multiplication protocol

3.3 Construction of $\mathtt{PPFDL}_{\mathtt{bsc}}$

We now describe the workflow of $\mathtt{PPFDL}_{\mathtt{bsc}}$. As described in Fig. 5, it contains two phases: system setup and encrypted $\mathtt{Meth}_{\mathtt{IU}}$.

3.3.1 System Setup

We require a trusted Third Party (TA) to generate a pair of asymmetric key (pk_1, sk_1) of Paillier cryptosystem for server $\boldsymbol{S_1}$, where pk_1 is the public key, and secret key sk_1 is kept by server $\boldsymbol{S_1}$. Similarly, TA also generates a pair of shared key (pk_u, sk_u) of Paillier cryptosystem for each user.

3.3.2 Encrypted $\mathtt{Meth_{IU}}$

Each user i encrypts its gradients $x_i^m, m \in [1, M]$ as $Enc_{pk_1}(x_i^m)$ and sends them to server $\boldsymbol{S_0}$. We know that Paillier cryptosystem works on integer domains due to the inherent requirement of the cryptographic primitive. To deal with it, we use a large rounding factor $\mathcal{L}$ to scale each x_i^m to an integer $\mathcal{L} \cdot x_i^m$ whenever needed. Then, for each negative integer x_i^m, we replace it with its inverse in the homomorphic encryption system, so that $d(x_i^m, x_*^m)$ will be set to a large integer when the signs of the two are inconsistent. This is a common trick widely adopted in works such as [17], [40], [41], [42]. For simplicity, we omit the above operation and default to all $\widehat{x}_n$ being integers in the following description. After receiving the encrypted local gradients from users, the server $\boldsymbol{S_0}$ and $\boldsymbol{S_1}$ work together to execute the encrypted $\mathtt{Meth_{IU}}$. Encrypted $\mathtt{Meth_{IU}}$ consists of four parts: (i) encrypted aggregated value initialization, (ii) encrypted user's reliability update and (iii) encrypted aggregated value update, (iv) weighted federated parameter update. The technical details of each component is described as follows.

(a) Encrypted aggregated value initialization: Before iteratively executing the encrypted $\mathtt{Meth_{IU}}$, the value of x_*^m used in the first iteration needs to be initialized. Based on the guideline of [25], x_*^m can be initialized as the mean of the sum of all users' m-th gradient. This can be computed as follows: Firstly, the server $\boldsymbol{S_0}$ calculates $Enc_{pk_1}(\sum_{i=1}^{i=N} x_i^m) = \prod_{i=1}^{i=N} Enc_{pk_1}(x_i^m)$ based on the homomorphic addition property. After that, each x_*^m can be initialized as $Enc_{pk_1}(x_{*(0)}^m) = Enc_{pk_1}(\sum_{i=1}^{i=N} x_i^m)^{-N}$.

(b) Encrypted user's reliability update: Given the encrypted gradient $Enc_{pk_1}(x_i^m)$ and $Enc_{pk_1}(x_*^m)$, the secure reliability update for each user i is described steps 1-6 in Fig. 5. In the final step, the encrypted user's reliability of each user is produced on the $\boldsymbol{S_0}$ side. Please note that if this is the first iteration, the initialized encrypted aggregated value $Enc_{pk_1}(x_*^m) = Enc_{pk_1}(x_{*(0)}^m)$.

(c) Encrypted aggregated value update: After obtaining the $Enc_{pk_1}(\mathcal{T}_i)$ for each user i, the encrypted aggregated value $Enc_{pk_1}(x_*^m)$ for each gradient m can be updated in the encrypted domain accordingly. The detailed description is presented steps 7-10 in Fig. 5. In the final step, the encrypted aggregated value $Enc_{pk_1}(x_*^m)$ for each gradient m is produced on the $\boldsymbol{S_0}$ side. Afterwards, $\boldsymbol{S_0}$ returns the encrypted aggregated results to each user for the next iteration.

(d) Weighted federated parameter update: As shown in Fig. 5, $\boldsymbol{S_0}$ and $\boldsymbol{S_1}$ perform steps 3 to 10 iteratively until the preset convergence conditions are met. Then, $\boldsymbol{S_0}$ returns $Enc_{pk_u}(x_*^m)$ to all users. Then, each user decrypts $Enc_{pk_u}(x_*^m)$ to get the aggregated result x_*^m for each type of gradient. Next, based on the Eqn.(3), where x_*^m is the weighted value of $\frac{\sum_{n \in N} \boldsymbol{x}_n^j}{\sum_{n \in N |\mathcal{D}_n^j|}}$, each user can update parameters of the local network. Finally, each user returns to step 1 for next iteration.

Remark: For the convenience of description, our $\mathtt{PPFDL_{bsc}}$ defaults to no user dropping out during the running process. However, it is robust to users dropping out during the whole implementation. First, during the process of executing encrypted $\mathtt{Meth_{IU}}$, each user i is only required to upload its encrypted local gradients. As a result, users can go offline at this stage once they submit their data to the server. On the other hand, although $\mathtt{PPFDL_{bsc}}$ is an iterated process where every user needs to update their local parameters based on the current global parameters in each iteration, we can observe that each user encrypts its gradients with the same public key. It will have no effect on the subsequent ciphertext operation (e.g., execute `SecMul` and `SecDiv`) even if some users fail to upload their data. This means that servers can still perform $\mathtt{PPFDL_{bsc}}$ and update parameters on the data of the remaining active users.

Please note that in `PPFDL`, users who are offline due to abnormal conditions may be normal users or irregular users. This is sure to reduce the training accuracy if the majority of users offline are normal users and vice versa. However, in a real network, it is difficult to determine the proportion of users offline for different types of users in advance. For convenience, we do not consider the effect of user offline on training accuracy. On the other hand, in the last iteration of encrypted aggregated value update, to ensure that the encrypted aggregated value is confidential to two servers and can be decrypted by each user, the public key pk_u shared by all users is used instead of the $\boldsymbol{S_1}$'s public key pk_1 in the fifth step of `SecDiv` (see Fig. 3). That is, $\boldsymbol{S_1}$ first sends the $Enc_{pk_u}(r)$ to $\boldsymbol{S_0}$. Then, $\boldsymbol{S_0}$ and $\boldsymbol{S_1}$ jointly run the OT protocol and `GC` to obtain the encrypted aggregated value $Enc_{pk_u}(x_*^m)$ for each $m \in [1, M]$. It should be explained that some additive blinding/masking techniques [12], [43] can also be used in the above process for converting one ciphertext to another ciphertext. However, to our best knowledge, it is infeasible to perform division (i.e., `SecDiv` shown in Fig. 3) under converted ciphertexts unless resorting to fully homomorphic encryption technology [44], [45]. This contradicts our assumption of additive homomorphic encryption in this paper.

3.4 Security Guarantees

Proposition 3: The $\mathtt{PPFDL_{bsc}}$ holds the security property that $\boldsymbol{S_0}$ and $\boldsymbol{S_1}$ learn nothing about each user-related private data, i.e., the gradient and reliability of each user, as well as the aggregation results.

Proof: We prove the theorem by a standard hybrid argument [12], [46]. Specifically, given a security parameter k, two non-collusive servers (i.e., $\boldsymbol{S_0}$ and $\boldsymbol{S_1}$), and a set $\mathcal{N}$ users (denoted as $1, 2, \cdots, N$), let $\mathcal{C} = \{\boldsymbol{S_0}, \boldsymbol{S_1}\}$, we use the random variable $\boldsymbol{Real}_{\boldsymbol{\mathcal{C}}}^{\mathcal{N},k}$ to represent the jointed view of $\mathcal{C}$ in the protocol $\mathtt{PPFDL_{bsc}}$. Then, we prove that there is a *PPT* simulator $\mathcal{SIM}$ such that for all k and $\mathcal{C}$, the output of $\mathcal{SIM}$ (i.e., $\boldsymbol{SIM}_{\boldsymbol{\mathcal{C}}}^{\mathcal{N},k}$) is computationally indistinguishable from the output of $\boldsymbol{Real}_{\boldsymbol{\mathcal{C}}}^{\mathcal{N},k}$.

To achieve this, we will define a simulator $\mathcal{SIM}$ by a series of (polynomial many) modifications to the random variable $\boldsymbol{Real}_{\boldsymbol{\mathcal{C}}}^{\mathcal{N},k}$, so that these two variables are ultimately

Implementation of $\texttt{PPFDL}_{\texttt{bsc}}$

Setup: Given the security parameter k, the trusted Third Party (TA) generates a pair of asymmetric key (pk_1, sk_1) of Paillier cryptosystem for server $\boldsymbol{S_1}$, where pk_1 is the public key, and secret key sk_1 is kept by server $\boldsymbol{S_1}$. Similarly, TA also generates a pair of shared key (pk_u, sk_u) of Paillier cryptosystem for each user.

Encrypted $\texttt{Meth}_{\texttt{IU}}$:

$\underline{user_i}$:

1. Each user i runs the unified neural network with a minibatch stochastic gradient descent rule to obtain the gradients in the current data set.
2. Each user i encrypts its gradients $x_i^m, m \in [1, M]$ as $Enc_{pk_1}(x_i^m)$ and sends them to the server $\boldsymbol{S_0}$.

$\underline{\boldsymbol{S_0}}$:

3. For each $m \in [1, M]$, $\boldsymbol{S_0}$ computes $Enc_{pk_1}(\mu_i^m) = Enc_{pk_1}(x_i^m) \cdot Enc_{pk_1}(x_*^m)^{-1}$.

$\underline{\boldsymbol{S_0}, \boldsymbol{S_1}}$:

4. $\boldsymbol{S_0}$ executes the $\texttt{SecMul}$ protocol with $\boldsymbol{S_1}$ to get $Enc_{pk_1}((\mu_i^m)^2)$ for each $m \in [1, M]$.

$\underline{\boldsymbol{S_0}}$:

5. $\boldsymbol{S_0}$ computes $Enc_{pk_1}(\mu_i) = Enc_{pk_1}(\sum_{m=1}^{m=M}(\mu_i^m)^2)$.

$\underline{\boldsymbol{S_0}, \boldsymbol{S_1}}$:

6. $\boldsymbol{S_0}$ executes the $\texttt{SecDiv}$ protocol with $\boldsymbol{S_1}$ to get $Enc_{pk_1}(\mathcal{T}_i)$.

$\underline{\boldsymbol{S_0}, \boldsymbol{S_1}}$:

7. $\boldsymbol{S_0}$ executes the $\texttt{SecMul}$ protocol with $\boldsymbol{S_1}$ to get $Enc_{pk_1}(\mathcal{T}_i \cdot x_i^m)$ for each $i \in [1, N]$.

$\underline{\boldsymbol{S_0}}$:

8. $\boldsymbol{S_0}$ computes $Enc_{pk_1}(\sum_{i=1}^{i=N} \mathcal{T}_i \cdot x_i^m) = \prod_{i=1}^{i=N} Enc_{pk_1}(\mathcal{T}_i \cdot x_i^m)$.
9. $\boldsymbol{S_0}$ computes $Enc_{pk_1}(\sum_{i=1}^{i=N} \mathcal{T}_i) = \prod_{i=1}^{i=N} Enc_{pk_1}(\mathcal{T}_i)$.

$\underline{\boldsymbol{S_0}, \boldsymbol{S_1}}$:

10. $\boldsymbol{S_0}$ executes the $\texttt{SecDiv}$ protocol with $\boldsymbol{S_1}$ to get $Enc_{pk_1}(x_*^m)$.

$\underline{\boldsymbol{S_0}, \boldsymbol{S_1}}$:

11. $\boldsymbol{S_0}$ and $\boldsymbol{S_1}$ perform steps 3 to 10 iteratively until the preset convergence conditions are met. Then, $\boldsymbol{S_0}$ returns $Enc_{pk_u}(x_*^m)$ to all users.

$\underline{user_i}$:

12. Each user i decrypts $Enc_{pk_u}(x_*^m)$ and updates parameters of the local network, return to step 1.

Fig. 5: Detailed description of $\texttt{PPFDL}_{\texttt{bsc}}$

computationally indistinguishable. The detailed proof is shown as follows.

$\mathbf{hyb_1}$ We initialize a random variable whose distribution is exactly the same as $\boldsymbol{Real}$, the joint view of the parties $\mathcal{C}$ in a real execution of the protocol.

$\mathbf{hyb_2}$ In this hybrid, we change the behavior of each user $i \in \mathcal{N}$, so that each user i encrypts randomly selected numbers γ_i^m with key pk_1 under Paillier cryptosystem, instead of encrypting the original gradient x_i^m. The two non-collusive servers setting and the property of Paillier cryptosystem [35] guarantee the indistinguishability of this hybrid from the previous one.

$\mathbf{hyb_3}$ In this hybrid, $\boldsymbol{S_0}$ computes $Enc_{pk_1}(\beta_i^m) = Enc_{pk_1}(\gamma_i^m) \cdot Enc_{pk_1}(x_*^m)^{-1}$. Reviewing step 3 in protocol $\texttt{PPFDL}_{\texttt{bsc}}$, the security of Paillier cryptosystem guarantees the indistinguishability between real $Enc_{pk_1}(\mu_i^m)$ and simulated $Enc_{pk_1}(\beta_i^m)$. Therefore, this hybrid is indistinguishable from the previous one.

$\mathbf{hyb_4}$ In this hybrid, we change the input of $\texttt{SecMul}$ protocol executed by $\boldsymbol{S_0}$ and $\boldsymbol{S_1}$ with $Enc_{pk_1}(\beta_i^m)$ instead of $Enc_{pk_1}((\mu_i^m))$. The security of $\texttt{SecMul}$ (described in Proposition 2) guarantees $\boldsymbol{S_0}$ and $\boldsymbol{S_1}$ learn nothing about the values of β_i^m. Hence, this hybrid is indistinguishable from the previous one.

$\mathbf{hyb_5}$ In this hybrid, we simulate $\boldsymbol{S_0}$ to compute $Enc_{pk_1}(\beta_i) = Enc_{pk_1}(\sum_{m=1}^{m=M}(\beta_i^m)^2)$, instead of computing $Enc_{pk_1}(\mu_i)$. The two non-collusive servers setting and the security of Paillier cryptosystem guarantee the indistinguishability between real $Enc_{pk_1}(\mu_i)$ and simulated $Enc_{pk_1}(\beta_i)$. Therefore, this hybrid is indistinguishable from the previous one.

$\mathbf{hyb_6}$ In this hybrid, we change the input of $\texttt{SecDiv}$ protocol executed by $\boldsymbol{S_0}$ and $\boldsymbol{S_1}$ with $Enc_{pk_1}(\beta_i)$ instead of $Enc_{pk_1}(\mu_i)$. The security of $\texttt{SecDiv}$ (described in Proposition 1) guarantees $\boldsymbol{S_0}$ and $\boldsymbol{S_1}$ learn nothing about the values of β_i, as well as simulated each user i' reliability $\mathcal{G}_i$. Hence, this hybrid is indistinguishable from the previous one.

$\mathbf{hyb_7}$ This hybrid is similar to $\mathbf{hyb_4}$, we change the input of $\texttt{SecMul}$ protocol executed by $\boldsymbol{S_0}$ and $\boldsymbol{S_1}$ with $Enc_{pk_1}(\mathcal{G}_i)$ and $Enc_{pk_1}(\gamma_i^m)$ instead of $Enc_{pk_1}(\mathcal{T}_i)$ and $Enc_{pk_1}(x_i^m)$. The security of $\texttt{SecMul}$ (described in Proposition 2) guarantees the indistinguishability between real $Enc_{pk_1}(\mathcal{T}_i \cdot x_i^m)$ and simulated $Enc_{pk_1}(\mathcal{G}_i \cdot \gamma_i^m)$. Hence, this hybrid is indistinguishable from the previous one.

$\mathbf{hyb_8}$ In this hybrid, we simulate $\boldsymbol{S_0}$ to compute $\prod_{i=1}^{i=N} Enc_{pk_1}(\mathcal{G}_i \cdot \gamma_i^m)$ and $\prod_{i=1}^{i=N} Enc_{pk_1}(\mathcal{G}_i)$ instead of computing $\prod_{i=1}^{i=N} Enc_{pk_1}(\mathcal{T}_i \cdot x_i^m)$ and $\prod_{i=1}^{i=N} Enc_{pk_1}(\mathcal{T}_i)$. The two non-collusive servers setting and the security of Paillier cryptosystem guarantee the indistinguishability between real results and simulated results. Therefore, this hybrid is indistinguishable from the previous one.

$\mathbf{hyb_9}$ In this hybrid, we change the input of $\texttt{SecDiv}$ protocol executed by $\boldsymbol{S_0}$ and $\boldsymbol{S_1}$ with $\prod_{i=1}^{i=N} Enc_{pk_u}(\mathcal{G}_i \cdot \gamma_i^m)$ and $\prod_{i=1}^{i=N} Enc_{pk_u}(\mathcal{G}_i)$ instead of $\prod_{i=1}^{i=N} Enc_{pk_u}(\mathcal{T}_i \cdot x_i^m)$ and $\prod_{i=1}^{i=N} Enc_{pk_u}(\mathcal{T}_i)$. The security of $\texttt{SecDiv}$ guarantees the indistinguishability between real aggregated results $Enc_{pk_u}(x_*^m)$ and simulated results $Enc_{pk_u}(\gamma_*^m)$. Hence, this hybrid is indistinguishable from the previous one.

As described above, with the security properties of `SecDiv`, `SecMul`, and Paillier cryptosystem, we prove that there is a *PPT* simulator $\mathcal{SIM}$ such that the output of $\mathcal{SIM}$ (i.e., $\boldsymbol{SIM}_{\mathcal{C}}^{\mathcal{N},k}$) is computationally indistinguishable from the output of $\boldsymbol{Real}_{\mathcal{C}}^{\mathcal{N},k}$. Furthermore, based on the simulation view, both $\boldsymbol{S_0}$ and $\boldsymbol{S_1}$ learn nothing about the plaintexts data under these interactions. ■

4 IMPROVED SCHEME

4.1 Secure Transformation Technique

We resort to the emerging technique proposed by *Catalano et al.* [47] to reduce the communication complexity generated in $\mathtt{PPFDL_{bsc}}$. For simplicity, we call this emerging technique as Secure Transformation Technique (`STT`). `STT` can transform `SecMul` into an additively homomorphic encryption (such as Paillier cryptosystem) based protocol to perform multiplication over encrypted data.

`STT` is also derived from secure two-party technology, which helps clients use two untrusted servers to do some calculations on outsourced clouds. Specifically, assume that a client has plaintexts x_1 and x_2. It first encrypts x_1 as $(x_1 - \sigma_1, Enc_{pk_c}(\sigma_1))$ and x_2 as $(x_2 - \sigma_2, Enc_{pk_c}(\sigma_2))$, and then sends them to two servers $\boldsymbol{S_0}$ and $\boldsymbol{S_1}$. $\boldsymbol{S_0}$ receives $C_1^{(1)} = (x_1 - \sigma_1, Enc_{pk_c}(\sigma_1))$ and $C_2^{(1)} = (x_2 - \sigma_2, Enc_{pk_c}(\sigma_2))$ while $\boldsymbol{S_1}$ receives $C_1^{(2)} = (\sigma_1)$ and $C_2^{(2)} = (\sigma_2)$, where pk_c is the public key of an additively homomorphic encryption scheme, σ_1 and σ_2 are two random values. In this paper, we called $C^{(1)}$ as the first-level `STT` ciphertext and $C^{(2)}$ as the second-level `STT` ciphertext. We can see that both servers learn nothing about x_1 and x_2 from ciphertexts as long as the two servers do not collude with each other. To enable the client to get the multiplication result $x_1 \cdot x_2$, $\boldsymbol{S_0}$ executes computation as follows:

$$Enc_{pk_c}((x_1 - \sigma_1)(x_2 - \sigma_2))Enc_{pk_c}(\sigma_1)^{(x_2-\sigma_2)}Enc_{pk_c}(\sigma_2)^{(x_1-\sigma_1)} \tag{6}$$

which is equal to

$$Enck_{pk_c}(x_1x_2 - \sigma_1\sigma_2) \tag{7}$$

Then, server $\boldsymbol{S_1}$ holding $C_1^{(2)} = (\sigma_1)$ and $C_2^{(2)} = (\sigma_2)$ can compute $\sigma_1 \cdot \sigma_2$. After receiving $Enck_{pk_c}(x_1x_2 - \sigma_1\sigma_2)$ and $\sigma_1 \cdot \sigma_2$ from the two servers, respectively, the client can decrypt $Enck_{pk_c}(x_1x_2 - \sigma_1\sigma_2)$ as $x_1x_2 - \sigma_1\sigma_2$, and then get x_1x_2 via $x_1x_2 - \sigma_1\sigma_2 + \sigma_1\sigma_2$. Note that `STT` can also support for multiplication of a constant and a ciphertext [47]. Given a constant t, $\boldsymbol{S_0}$ can compute $(t(x_1 - \sigma_1), Enc_{pk_c}(\sigma_1)^t) = (tx_1 - t\sigma_1, Enc_{pk_c}(t\sigma_1))$ while $\boldsymbol{S_1}$ computes $t\sigma_1$. Similarly, the client can obtain tx_1 by Eqn.(6) and Eqn.(7).

4.2 Our Idea

We integrate and adapt the `STT` in our $\mathtt{PPFDL_{imd}}$ to improve the communication complexity between the two servers. Our main idea is to let $\boldsymbol{S_0}$ hold the first-level `STT` ciphertext while $\boldsymbol{S_1}$ hold the second-level `STT` ciphertext. Then, servers $\boldsymbol{S_0}$ and $\boldsymbol{S_1}$ perform the multiplication operations locally in batches. When the operation is completed, each server locally aggregates the calculation results so that only the aggregated results between the two servers are transmitted. For example, assume that $\boldsymbol{S_0}$ has K multiplication results, i.e., $\{Enck_{pk_1}(x_{1k}x_{2k} - \sigma_{1k}\sigma_{2k})\}_{k=1}^{k=K}$, and $\boldsymbol{S_1}$ has $\{\sigma_{1k}\sigma_{2k}\}_{k=1}^{k=K}$. Then, $\boldsymbol{S_0}$ and $\boldsymbol{S_1}$ can compute $Enck_{pk_1}(\sum_{k=1}^{k=K}(x_{1k}x_{2k} - \sigma_{1k}\sigma_{2k}))$ and $\sum_{k=1}^{k=K}\sigma_{1k}\sigma_{2k}$, respectively. After that, $\boldsymbol{S_1}$ sends $Enck_{pk_1}(\sum_{k=1}^{k=K}\sigma_{1k}\sigma_{2k})$ to $\boldsymbol{S_0}$. As a result, $\boldsymbol{S_0}$ can obtain $Enck_{pk_1}(\sum_{k=1}^{k=K}(x_{1k}x_{2k}))$ via $Enck_{pk_1}(\sum_{k=1}^{k=K}(x_{1k}x_{2k} - \sigma_{1k}\sigma_{2k})) \cdot Enck_{pk_1}(\sum_{k=1}^{k=K}\sigma_{1k}\sigma_{2k})$. In this way, $\boldsymbol{S_0}$ and $\boldsymbol{S_1}$ only need $O(1)$ communication to get the sum of K multiplication values.

4.3 Construction of $\mathtt{PPFDL_{imd}}$

We now describe the workflow of our improved protocol. The improved protocol also generally includes two phases: system setup and encrypted $\mathtt{Meth_{IU}}$.

4.3.1 System Setup

We require a trusted Third Party (TA) to generate a pair of asymmetric key (pk_1, sk_1) of Paillier cryptosystem for the server $\boldsymbol{S_1}$, where pk_1 is the public key, and secret key sk_1 is kept by the server $\boldsymbol{S_1}$. Similarly, TA also generates a pair of shared key (pk_u, sk_u) of Paillier cryptosystem for each user.

4.3.2 Encrypted $\mathtt{Meth_{IU}}$

Each user i encrypts its gradients $x_i^m, m \in [1, M]$ as $(x_i^m - \sigma_i^m, Enc_{pk_1}(\sigma_i^m))$. Then, each user i sends $(x_i^m - \sigma_i^m, Enc_{pk_1}(\sigma_i^m))$ to the server $\boldsymbol{S_0}$ and σ_i^m to $\boldsymbol{S_1}$. After receiving the encrypted local gradients from users, server $\boldsymbol{S_0}$ and $\boldsymbol{S_1}$ work together to execute the encrypted $\mathtt{Meth_{IU}}$, so as to obtain the encrypted aggregated value for each type of gradients. As described in Section 3.3.2, $\mathtt{Meth_{IU}}$ also consists of four parts: (i) encrypted aggregated value initialization, (ii) encrypted user's reliability update and (iii) encrypted aggregated value update, (iv) weighted federated parameter update. The technical details of each component is described as below.

(a) Encrypted aggregated value initialization: Similarly, the value of x_*^m used in the first iteration needs to be initialized. This can be computed as follows: Firstly, the server $\boldsymbol{S_0}$ calculates $\sum_{i=1}^{i=N}(x_i^m - \sigma_i^m)$ and $Enc_{pk_1}(\sum_{i=1}^{i=N}\sigma_i^m)$ based on the homomorphic addition property. After that, $\boldsymbol{S_0}$ calculates $x_{*(0)}^m - \sigma_{*(0)}^m = \sum_{i=1}^{i=N}(x_i^m - \sigma_i^m)/N$ and $Enc_{pk_1}(\sigma_{*(0)}^m) = Enc_{pk_1}(\sum_{i=1}^{i=N}\sigma_i^m)^{-N}$. Meanwhile, $\boldsymbol{S_1}$ calculates $\sigma_{*(0)}^m = \sum_{i=1}^{i=N}\sigma_i^m/N$.

(b) Encrypted user's reliability update: Given the ciphertexts $Enc_{pk_1}(\sigma_i^m)$ and $(x_*^m - \sigma_*^m, Enc_{pk_1}(\sigma_*^m))$ at $\boldsymbol{S_0}$ side, and σ_*^m at $\boldsymbol{S_1}$ side, the secure reliability update for each user i is described steps 3-9 in Fig. 6. In the final step, $\boldsymbol{S_0}$ executes the `SecDiv` protocol with $\boldsymbol{S_1}$ to get the new user i's reliability, i.e., $\boldsymbol{S_0}$ obtains $(\mathcal{T}_i - \sigma_i^{\mathcal{T}}, Enc_{pk_1}(\sigma_i^{\mathcal{T}}))$ and $\boldsymbol{S_1}$ obtains $\sigma_i^{\mathcal{T}}$.

(c) Encrypted aggregated value update: After $\boldsymbol{S_0}$ obtaining $(\mathcal{T}_i - \sigma_i^{\mathcal{T}}, Enc_{pk_1}(\sigma_i^{\mathcal{T}}))$; $\boldsymbol{S_1}$ obtaining σ_i^m and $\sigma_i^{\mathcal{T}}$ for each user i, the encrypted aggregated value for each gradient m can be updated in the encrypted domain accordingly. The detail description is presented steps 10-16 in Fig. 6. In the final step, $\boldsymbol{S_0}$ executes the `SecDiv` protocol with $\boldsymbol{S_1}$ to get

the new aggregated value for gradient m, i.e., $\boldsymbol{S_0}$ obtains $(x_*^m - \sigma_*^m, Enc_{pk_1}(\sigma_*^m))$ and $\boldsymbol{S_1}$ obtains σ_*^m.

(d) Weighted federated parameter update: As shown in Fig.6, $\boldsymbol{S_0}$ and $\boldsymbol{S_1}$ perform steps 3 to 16 iteratively until the preset convergence conditions are met. Then, $\boldsymbol{S_0}$ and $\boldsymbol{S_1}$ return $Enc_{pk_u}(x_*^m - \sigma_*^m)$ and $Enc_{pk_u}(\sigma_*^m)$ to all users, respectively. Then, each user decrypts $Enc_{pk_u}(x_*^m - \sigma_*^m)$ and $Enc_{pk_u}(\sigma_*^m)$ to get the aggregated result x_*^m for each type of gradient. Next, based on the Eqn.(3), where x_*^m is the weighted value of $\frac{\sum_{n\in N}\boldsymbol{x}_n^j}{\sum_{n\in N}|\mathcal{D}_n^j|}$, each user can update parameters of the local network. Finally, each user returns to step 1 for next iteration.

4.4 Complexity Analysis

We now discuss the communication and computation complexity of $\texttt{PPFDL}_{\texttt{bsc}}$ and $\texttt{PPFDL}_{\texttt{imd}}$. For the sake of simplicity, here we do not consider the existence of users dropping out in the execution.

In $\texttt{PPFDL}_{\texttt{bsc}}$, $\boldsymbol{S_0}$ needs to interact with $\boldsymbol{S_1}$ to achieve encrypted multiplication and division via `SecDiv` or `SecMul`. Firstly, we note that `SecDiv` is exploited to execute N times in the phase of the secure user's reliability update, and M times in the phase of the secure aggregated value update. So the communication complexity incurred in `SecDiv` is $O(\mathcal{K}(N+M))$, where $\mathcal{K}$ denotes the number of iterations during training. Secondly, we note that `SecMul` is exploited to compute $Enc_{pk_1}(\mu_i^m)^2$ for each $i \in [1, N]$ in the phase of the secure user's reliability update (step 2 in Fig. 5), and $Enc_{pk_1}(\mathcal{T}_i \cdot x_i^m)$ for each $m \in [1, M]$ (step 7 in Fig. 5). For the first term, `SecMul` needs to be called M times for each user since the number of gradients held by each user i is M. Hence, given N users, `SecMul` will be called $M \cdot N$ times to compute the first term for all users. For the second term, `SecMul` needs to be called N times for every gradient since the number of users providing data for gradient m is N. Similarly, given M types of gradients, `SecMul` will be called $N \cdot M$ times to compute the second term for all types of gradients. Therefore, the communication complexity between the two servers incurred in `SecDiv` is $O(\mathcal{K}(N \cdot M))$, which means that the communication overhead of $\texttt{PPFDL}_{\texttt{bsc}}$ depends on the number of users and the number of gradients submitted by each user. This may be poor scalability when the number of users in the system or the number of gradients held by each user is large.

In $\texttt{PPFDL}_{\texttt{imd}}$, for each user i', $\boldsymbol{S_1}$ computes $\sum_{m=1}^{m=M}(\sigma_i^m - \sigma_*^m)^2$ and sends $Enc_{pk_1}(\sum_{m=1}^{m=M}(\sigma_i^m - \sigma_*^m)^2)$ to $\boldsymbol{S_0}$ (steps 6-7 in Fig. 6). Therefore, $\boldsymbol{S_1}$ only needs $O(1)$ communication to get the sum of M multiplication values, thereby requiring $O(N)$ communication complexity for all N users. In secure aggregated value update, $\boldsymbol{S_1}$ computes $\sum_{i=1}^{i=N}\sigma_i^m \cdot \sigma_i^{\mathcal{T}}$ for each gradient, and sends $Enc_{pk_1}(\sum_{i=1}^{i=N}\sigma_i^m \cdot \sigma_i^{\mathcal{T}})$ to $\boldsymbol{S_0}$ (steps 12-13 in Fig. 6). Therefore, for each gradient m, $\boldsymbol{S_1}$ only needs $O(1)$ communication to get the sum of N multiplication values, thereby requiring $O(M)$ communication complexity for all M types of gradients. As a result, given N users and M gradients for each user, the communication complexity of secure multiplication is now $O(\mathcal{K}(N+M))$. We know that the communication complexity of other interactions (e.g., secure division) are same as the basic model $\texttt{PPFDL}_{\texttt{bsc}}$, i.e., $O(\mathcal{K}(N+M))$. Therefore, $\texttt{PPFDL}_{\texttt{imd}}$ improves the communication complexity to $O(\mathcal{K}(N+M))$.

We emphasize that $\texttt{PPFDL}_{\texttt{imd}}$ does not compromise computation overhead while reducing communication overhead. In fact, changing the way the servers computing multiplications under ciphertext also increases the computation performance of $\texttt{PPFDL}_{\texttt{imd}}$. As described in TABLE I, we can observe that $\texttt{PPFDL}_{\texttt{imd}}$ is superior to $\texttt{PPFDL}_{\texttt{bsc}}$ in terms of computation overhead. Specifically, in $\texttt{PPFDL}_{\texttt{imd}}$, we utilize the technique of Secure Transformation Technique (`STT`) to perform multiplication over encrypted data, instead of exploiting `SecMul`. This allows servers to compute the product of multiple plaintexts locally. As a result, each user only needs $O(\mathcal{K}(M+N))$ computation complexity compared to $\texttt{PPFDL}_{\texttt{bsc}}$.

4.5 Security Guarantees

Proposition 4: The $\texttt{PPFDL}_{\texttt{imd}}$ *holds the security property that* $\boldsymbol{S_0}$ *and* $\boldsymbol{S_1}$ *learn nothing about each user-related private data, i.e., the gradient and reliability of each user, as well as the aggregation results. Besides, each user learns nothing about users' reliability.*

Proof: Similar to the proof of $\texttt{PPFDL}_{\texttt{bsc}}$, we also prove that there exists a *PPT* simulator $\mathcal{SIM}$ such that for all k and $\mathcal{C}$, the output of $\mathcal{SIM}$ (i.e., $\boldsymbol{SIM}_{\boldsymbol{\mathcal{C}}}^{\mathcal{N},k}$) is computationally indistinguishable from the output of $\boldsymbol{Real}_{\boldsymbol{\mathcal{C}}}^{\mathcal{N},k}$. The detailed proof please refer to **APPENDIX**.

5 PERFORMANCE EVALUATION

In this section, we conduct experiments to evaluate the performance of our PPFDL. The configuration is as follows: All the experiments are compiled in the JAVA language, where each user is replaced by a Huawei nova3 android phone equipped with 6GB RAM, four-core 2.36GHz Cortex A73 processor and four-core Cortex A53 1.8GHz processor. The "Cloud" is simulated with two Lenovo servers which have 2 Intel(R) Xeon(R) E5-2620 2.10GHZ CPU, 32GB RAM, 512SSD, 2TB mechanical hard disk and runs on the Ubuntu 18.04 operating system. For cryptographic primitives, we use the Paillier library [1] to implement homomorphic encryption and the ObliVM-lang-programming framework [2] for garbled circuits. In addition, all the raw data are selected from MNIST database [3] which has a training set of 60,000 examples, and a test set of 10,000 examples. Besides, we let all users run a unified convolutional neural network (CNN) offline to obtain the gradients corresponding to the local data, where the CNN adopted in our experiments consists of 2 convolutional layers (containing 20 feature maps and 50 feature maps, respectively), one average pooling layer and two fully connected layers (256 and 10 neurons, respectively). We set the learning rate to 0.01, and the mini-batch size selected each time to 128.

5.1 Functionality

We first analyze the functional advantages of PPFDL compared to state-of-the-art privacy-preserving federated deep

1. https://mshcruz.wordpress.com/2017/01/26/using-paillier-library/
2. https://github.com/oblivm/ObliVMLang
3. http://yann.lecun.com/exdb/mnist/

TABLE I: Comparison of computation overhead between $\texttt{PPFDL}_{\texttt{bsc}}$ and $\texttt{PPFDL}_{\texttt{imd}}$

Types	S_0		S_1	
	$\texttt{PPFDL}_{\texttt{bsc}}$	$\texttt{PPFDL}_{\texttt{imd}}$	$\texttt{PPFDL}_{\texttt{bsc}}$	$\texttt{PPFDL}_{\texttt{imd}}$
`SecDiv`	$O(\mathcal{K}(M+N))$	$O(\mathcal{K}(M+N))$	$O(\mathcal{K}(M+N))$	$O(\mathcal{K}(M+N))$
`SecMul`	$O(2\mathcal{K}MN)$	0	$O(2\mathcal{K}MN)$	0
Homomorphic	$O(2\mathcal{K}MN)$	$O(\mathcal{K}(M+N))$	0	$O(\mathcal{K}(M+N))$

Implementation of $\texttt{PPFDL}_{\texttt{imd}}$

Setup: Given the security parameter k, the trusted Third Party (TA) generates a pair of asymmetric key (pk_1, sk_1) of Paillier cryptosystem for server S_1, where pk_1 is the public key, and secret key sk_1 is kept by server S_1. Similarly, TA also generates a pair of shared key (pk_u, sk_u) of Paillier cryptosystem for each user.

Encrypted $\texttt{Meth}_{\texttt{IU}}$:

<u>$user_i$</u>:

1. Each user i runs the unified neural network with a minibatch stochastic gradient descent rule to obtain the gradients in the current data set.
2. Each user i encrypts its gradients $x_i^m, m \in [1, M]$ as $(x_i^m - \sigma_i^m, Enc_{pk_1}(\sigma_i^m))$. Then, each user i sends $(x_i^m - \sigma_i^m, Enc_{pk_1}(\sigma_i^m))$ to the server S_0, and σ_i^m to S_1.

<u>S_0</u>:

3. For each $m \in [1, M]$, S_0 computes $(x_i^m - x_*^m) - (\sigma_i^m - \sigma_*^m) = \mu_i^m - (\sigma_i^m - \sigma_*^m)$.
4. S_0 computes $Enc_{pk_1}((\mu_i^m)^2 - (\sigma_i^m - \sigma_*^m)^2)$ through the property of `STT` technique (Eqn.(6)).
5. S_0 computes $Enc_{pk_1}(\mu_i - \sum_{m=1}^{m=M}(\sigma_i^m - \sigma_*^m)^2)$ via the property of homomorphic addition, where $\mu_i = \sum_{m=1}^{m=M}(\mu_i^m)^2 = \sum_{m=1}^{m=M}(x_i^m - x_*^m)^2$.

<u>S_1</u>:

6. S_1 computes $\sum_{m=1}^{m=M}(\sigma_i^m - \sigma_*^m)^2$.
7. S_1 sends $Enc_{pk_1}(\sum_{m=1}^{m=M}(\sigma_i^m - \sigma_*^m)^2)$ to S_0.

<u>S_0</u>:

8. S_0 computes $Enc_{pk_1}(\mu_i - \sum_{m=1}^{m=M}(\sigma_i^m - \sigma_*^m)^2) \cdot Enc_{pk_1}(\sum_{m=1}^{m=M}(\sigma_i^m - \sigma_*^m)^2)$, which is equal to $Enc_{pk_1}(\mu_i)$.

<u>S_0, S_1</u>:

9. S_0 executes the `SecDiv` protocol with S_1 to get the new user i's reliability, i.e., S_0 obtains $(\mathcal{T}_i - \sigma_i^{\mathcal{T}}, Enc_{pk_1}(\sigma_i^{\mathcal{T}}))$ and S_1 obtains $\sigma_i^{\mathcal{T}}$.

<u>S_0</u>:

10. For each user $i \in [1, N]$, S_0 computes $Enc_{pk_1}(\mathcal{T}_i \cdot x_i^m - \sigma_i^m \cdot \sigma_i^{\mathcal{T}})$ based on the property of `STT` technique (Eqn.(6)).
11. S_0 computes $Enc_{pk_1}(\sum_{i=1}^{i=N}\mathcal{T}_i \cdot x_i^m - \sigma_i^m \cdot \sigma_i^{\mathcal{T}}) = \prod_{i=1}^{i=N} Enc_{pk_1}(\mathcal{T}_i \cdot x_i^m - \sigma_i^m \cdot \sigma_i^{\mathcal{T}})$ via the property of homomorphic addition.

<u>S_1</u>:

12. S_1 computes $\sum_{i=1}^{i=N}\sigma_i^m \cdot \sigma_i^{\mathcal{T}}$.
13. S_1 sends $Enc_{pk_1}(\sum_{i=1}^{i=N}\sigma_i^m \cdot \sigma_i^{\mathcal{T}})$ to S_0.

<u>S_0</u>:

14. S_0 computes $Enc_{pk_1}(\sum_{i=1}^{i=N}\mathcal{T}_i \cdot x_i^m) = Enc_{pk_1}(\sum_{i=1}^{i=N}(\mathcal{T}_i \cdot x_i^m - \sigma_i^m \cdot \sigma_i^{\mathcal{T}}) \cdot Enc_{pk_1}(\sum_{i=1}^{i=N}\sigma_i^m \cdot \sigma_i^{\mathcal{T}})$.
15. S_0 computes $Enc_{pk_1}(\sum_{i=1}^{i=N}\mathcal{T}_i) = Enc_{pk_1}(\sum_{i=1}^{i=N}(\mathcal{T}_i - \sigma_i^{\mathcal{T}})) \cdot Enc_{pk_1}(\sum_{i=1}^{i=N}\sigma_i^{\mathcal{T}})$.

<u>S_0, S_1</u>:

16. S_0 executes the `SecDiv` protocol with S_1 to get the new aggregated value for gradient m, i.e., S_0 obtains $(x_*^m - \sigma_*^m, Enc_{pk_1}(\sigma_*^m))$ and S_1 obtains σ_*^m.

<u>S_0, S_1</u>:

17. S_0 and S_1 perform steps 3 to 16 iteratively until the preset convergence conditions are met. Then, S_0 and S_1 return $Enc_{pk_u}(x_*^m - \sigma_*^m)$ and $Enc_{pk_u}(\sigma_*^m)$ to all users, respectively.

<u>$user_i$</u>:

18. Each user i decrypts ciphertexts and updates parameters of the local network, return to step 1.

Fig. 6: Detailed description of $\texttt{PPFDL}_{\texttt{imd}}$

TABLE II: Comparison of functionality with existing models

Function / Model	Protection of user's gradients	Protection of aggregated result	Robust to users dropping out	Support for irregular users	Server setting
SecProbe [14]	✔	✗	✗	✔	**Honest-but-Curious**
PPFL [12]	✔	✗	✔	✗	**Honest-but-Curious**
PPDL [10]	✔	✔	✗	✗	**Honest-but-Curious**
SDLW [48]	✔	✗	✗	✗	**Honest-but-Curious**
PPFDL	✔	✔	✔	✔	**Honest-but-Curious**

learning models, i.e., SecProbe [14], PPFL [12], PPDL [10], and SDLW [48]. As shown in TABLE II, SecProbe [14] is the first solution to handle irregular users during federated training. However, SecProbe does not guarantee the confidentiality of the aggregation results, nor can it support users to go offline during the training process. PPDL [10]

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2020.3005909, IEEE Transactions on Dependable and Secure Computing

and SDLW [48] mainly use homomorphic encryption to implement training tasks, whose goal focuses on how to achieve efficient parameter sharing under a small-scale user set. Therefore, the case of dealing with users dropping out and irregular users during federated training are beyond their scope of works. PPFL [12] is the first work to support users offline during the training process. The authors exploit secret sharing and the key exchange protocol to ensure the confidentiality of each user' s local gradients. However, PPFL default aggregation results are publicly available parameters and assumes that all users hold undifferentiated data. Compared with these approaches, our PPFDL integrates the technologies of additive homomorphism and Yao's garbled circuits, which can ensure the high security of all user-related information, including the gradient and reliability of each user, as well as the aggregation results. Then, PPFDL designs $\texttt{Meth}_{\texttt{IU}}$ to reduce the impact of users with data of low quality in the training process. Moreover, since PPFDL only requires each user i to upload its encrypted local gradients to the server $\boldsymbol{S_0}$ with the same public key pk_1, it will not affect the subsequent ciphertext operation even if some users fail to upload their data. Hence, PPFDL is also robustness to users dropping out during the whole implementation.

5.2 Accuracy

In this section, we discuss the training accuracy of PPFDL. As described before, many factors affect the accuracy of the model, including the iteration times in the encrypted $\texttt{Meth}_{\texttt{IU}}$, the proportion of irregular users, the number of users N participating in training, and the number of gradients M held by each user. In our experiments, we assume that each user i holds a dataset $\mathcal{D}_i$ of the same size, i.e., $|\mathcal{D}_1| = |\mathcal{D}_2| = \cdots |\mathcal{D}_N|$, where $\sum_{i=1}^{i=N} \mathcal{D}_i = 60000$. To simulate irregular users, we randomly select half of the users and replace the P part of their data with random noise in the range $[0, 1]$. As a result, these noise-added data are considered as irregular data. We change P to evaluate the robustness of PPFDL to irregular users. This setting is also used in SecProbe [14], the first privacy-preserving collaborative deep learning with irregular users. In the following subsections, we analyze the impact of these factors in detail.

5.2.1 Effect of iteration times with different proportions of irregular data

We first discuss the effect of iteration times in encrypted $\texttt{Meth}_{\texttt{IU}}$. As described in Section 3.3 and Section 4.3, we know that in the initialization process of $\texttt{PPFDL}_{\texttt{bsc}}$ and $\texttt{PPFDL}_{\texttt{imd}}$, the value of x_*^m used in the first iteration needs to be initialized for each gradient m. Works [25] give a guideline that each gradient x_*^m can be initialized as the mean of the sum of all users on m-th gradient. However, there are errors in this approach due to the variety of data quality held by each user. Fortunately, work [25] shows that the error rate can be reduced by multiple iterations between the two servers. On the other hand, the proportion of irregular data in the system also affects the accuracy of the model. Intuitively, the more users with high-quality data in the system bring about the better model accuracy. To observe the accuracy of PPFDL under these variables, we conduct experiments to record the accuracy with a different number of iterations, where we also consider the factor of different proportions of irregular data in the training process. Fig. 7 shows the comparison of accuracy with a different number of iterations, where OFL [28] is the original model performing learning in the plaintext environment. We can observe that the accuracy of PPFDL is significantly higher than OFL as long as the number of iterations exceeds a certain threshold (such as 3). Moreover, the accuracy of OFL drops very quickly when the proportion of P reaches 25%, while the accuracy of PPFDL has only a small change. This is mainly due to the superiority of our $\texttt{Meth}_{\texttt{IU}}$, which guarantees the result of aggregation from the contribution of high-quality users by assigning high reliability to these users holding high-quality data.

5.2.2 Effect of N with different proportions of irregular data

We next analyze the impact of the number of users N participating in the training. As introduced before, *Zhao et al.* [14] proposed SecProbe, the first privacy-preserving collaborative deep learning with irregular participants, which exploited differential privacy-based technologies to perturb the objective function of targeted DNNs, and claimed that SecProbe can ensure a good balance between accuracy and security. Therefore, in order to make the experimental results more convincing, we also conduct experiments to compare the accuracy with SecProbe. The experimental configuration for implementing SecProbe is as follows: For fairness, the experimental hardware configuration is consistent with our PPFDL, i.e., the experiments are compiled in the JAVA language, where each user is replaced by a Huawei nova3 android phone equipped with 6GB RAM, four-core 2.36GHz Cortex A73 processor, and four-core Cortex A53 1.8GHz processor. The " Cloud " is simulated with two Lenovo servers which have 2 Intel(R) Xeon(R) E5-2620 2.10GHZ CPU, 32GB RAM, 512SSD, 2TB mechanical hard disk and runs on the Ubuntu 18.04 operating system. We also train a CNN (with 2 convolutional layers, one average pooling layer, and two fully connected layers) utilizing the framework designed in [14]. Other hyperparameters are the same as those mentioned in SecProbe, the learning rate is set to 0.01, and the mini-batch size selected each time to 128.

Fig. 8 shows the comparison of accuracy with a different number of users. where we set the gradients held by each user to 2500, and privacy budget ϵ in SecProbe to 10 (same as the experimental setup in SecProbe [14]). It can be observed that the increase in the number of users in the system helps to improve model accuracy. This is well understood. As the number of users increases, the data set shared between users becomes larger, so that the results predicted under a large data set tend to be in a good direction. Also, the accuracy of our PPFDL is much higher than OFL and SecProbe. In addition, it is worth noting that SecProbe does not show a significant advantage in accuracy to OFL. This stems from the use of differential privacy in SecProbe. Indeed, SecProbe has made great efforts to reduce the impact of added noise on the final result, by using functional encryption to perturb the objective function of the neural network rather than adding noise directly to the gradients. However, the use of differential privacy techniques inevitably incurs noise that is difficult to offset. As shown in the state-of-the-art

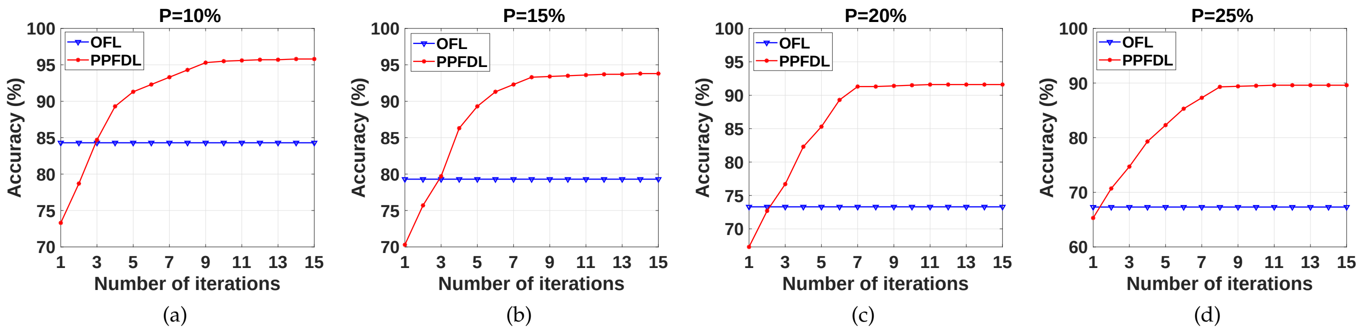


Fig. 7: Comparison of accuracy with different number of iterations.

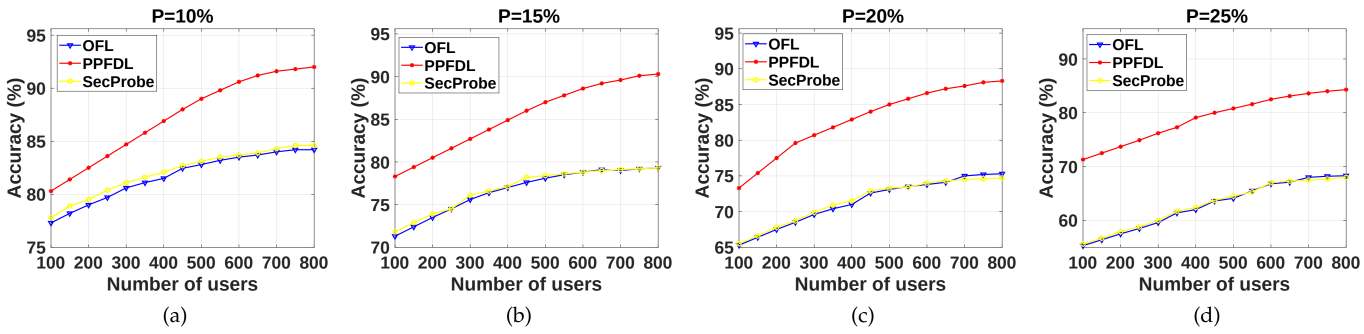


Fig. 8: Comparison of accuracy with different number of users.

works [6], [16], current mechanisms for differentially private deep learning rarely offer acceptable utility-privacy trade-offs for complex learning tasks: the adversary can still easily recover the user's sensitive data if the model is acceptable for accuracy. Conversely, our $\mathtt{Meth}_{\mathtt{IU}}$ is derived from the traditional truth discovery algorithms [49], [50], which have shown excellent performance in estimating real data from numerous heterogeneous data, and can guarantee the result of aggregation from the contribution of high-quality users.

5.2.3 *Effect of M with different proportions of irregular data*

We now describe the impact of the number of gradients M held by each user on the accuracy of PPFDL. As shown in Fig. 9, the accuracy will be improved as more gradients are involved in the training process. It is obvious that sharing rich gradients between users will speed up the convergence of training and also make the results more realistic. It is not difficult to find that our solution still shows better accuracy than solutions OFL and SecProbe. The reason is described above, i.e., our $\mathtt{Meth}_{\mathtt{IU}}$ guarantees high reliability to users with high-quality data, and ensures that the results of the aggregation are mainly from the contributions of these users.

5.3 Computation Overhead

In this section, we evaluate the computation overhead of our PPFDL. As described before, PPFDL consists of two schemes, i.e., $\mathtt{PPFDL}_{\mathtt{bsc}}$ and $\mathtt{PPFDL}_{\mathtt{imd}}$, where $\mathtt{PPFDL}_{\mathtt{imd}}$ is the improved version of $\mathtt{PPFDL}_{\mathtt{bsc}}$ in terms of computation and

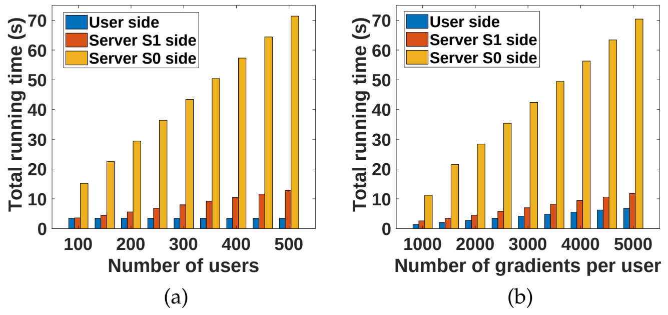


Fig. 10: Computation cost of $\mathtt{PPFDL}_{\mathtt{imd}}$.

communication overhead. For the sake of convenience, here we will only discuss the performance of $\mathtt{PPFDL}_{\mathtt{imd}}$. In addition, to visualize the experimental results, the latest work PPFL [12] is also included and compared with PPFDL in our experiments. PPFL [12] is the first work to achieve privacy-preserving federated training. Besides, similar to ours, it is also robustness to users dropping out during the training process. The experimental configuration for implementing PPFL is as follows: For fairness, the experimental hardware configuration is consistent with our PPFDL, where we also train a CNN (with 2 convolutional layers, one average pooling layer, and two fully connected layers) utilizing the framework designed in [12]. Other hyperparameters are the same as those mentioned in PPFDL. i.e., we adopt the stan-

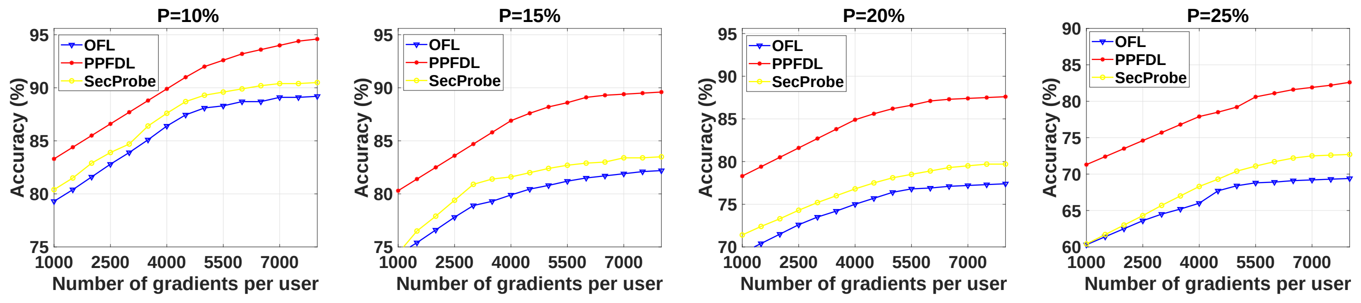

(a) (b) (c) (d)

Fig. 9: Comparison of accuracy with different number of gradients per user.

dard Shamir's secret sharing protocol [51] to generate the shares of secrets in PPFDL. In addition, the key agreement protocol based on Elliptic-Curve is exploited to achieve the key distribution between two users, the AES in counter mode and AES-GCM with 128-bit keys is used to achieve the authenticated encryption and pseudorandom generator, respectively.

Fig. 10 shows the computation cost of $\mathtt{PPFDL_{imd}}$ with a different number of users/gradients per user. We can see that the overhead of the user side is much smaller than the cost of the two servers. Besides, the cost of the S_0 side is higher than the server S_0 side. In $\mathtt{PPFDL_{imd}}$, each user is only required to send its own encrypted gradient set to the server, and all ciphertext processing is done by two servers interacting with each other. Therefore, $\mathtt{PPFDL_{imd}}$ is very friendly for end-users with limited computing power. On the other hand, to update each user's reliability or each gradient's aggregated value, server S_1 only needs to deal with fast arithmetic operations over random values and a fixed number of homomorphic encryptions. Conversely, most of the ciphertext operations are carried out by the server S_0. This inevitably makes the S_0 need more computing resources to complete the corresponding tasks of ciphertext computing.

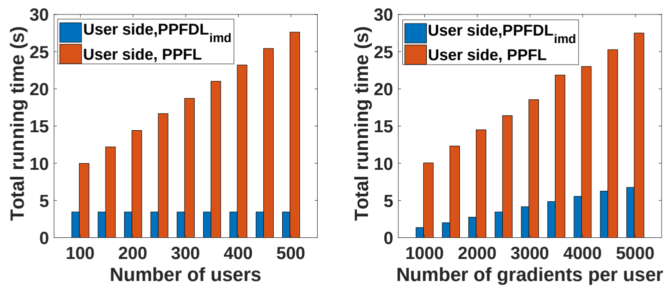

(a) (b)

Fig. 11: Comparison of each user's computation cost between $\mathtt{PPFDL_{imd}}$ and PPFL [12].

Fig. 11 shows the comparison of each user's computation cost between $\mathtt{PPFDL_{imd}}$ and PPFL [12]. We can observe that the computation overhead of $\mathtt{PPFDL_{imd}}$ is much lower than PPFL as the number of users/gradients per user increases. This is mainly due to the extremely frequent interaction between users and the server in PPFL. Specifically, PPFL exploits secret sharing and the key exchange protocol to protect the privacy of local gradients, and achieve robustness to users dropping out during the training process. This means that in addition to encrypting its own gradient data set, each user is also required to encrypt the shares of its secrets and send them to all other users. Moreover, when some users are offline during the training process, in order to ensure successful decryption, all online users need to jointly perform a secret sharing protocol to recover the offline users' secrets and send them to the server. This makes each user's computation complexity as high as $O(\mathcal{K}(N+M\times N))$ in the worst case, where N is the number of users in the system, and M denotes the gradients held by each user.

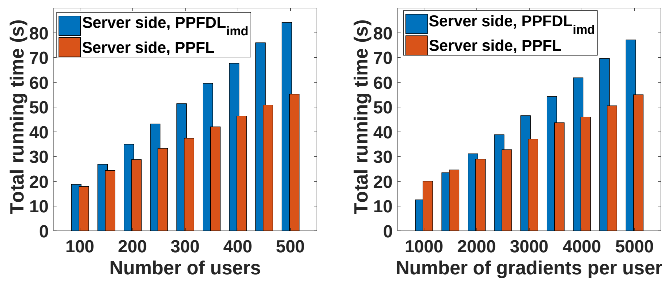

(a) (b)

Fig. 12: Comparison of server's computation cost between $\mathtt{PPFDL_{imd}}$ and PPFL [12].

Fig. 12 shows the comparison of the server's computation cost between $\mathtt{PPFDL_{imd}}$ and PPFL [12], where the server's computation cost of $\mathtt{PPFDL_{imd}}$ represents the sum of the cost of S_0 and server S_1. We can see that the cost of our $\mathtt{PPFDL_{imd}}$ is slightly higher than PPFL. The main reason is that in $\mathtt{PPFDL_{imd}}$, two servers need to call garbled circuits to implement the ciphertext division. This inevitably leads to computation overhead that cannot be ignored. However, the cost of our solution is still reasonable and can be used in practical applications. For example, when the system user is 500 and the amount of data held by each user is 5000, the PPFL needs 51 seconds to complete a training update, and $\mathtt{PPFDL_{imd}}$ only needs 78 seconds. In addition, compared to

PPFL, $\texttt{PPFDL}_{\texttt{imd}}$ guarantees the confidentiality of aggregated results as well as each user's reliability. Moreover, it also designs a novel solution to reduce the impact of users with data of low quality in the training process.

5.4 Communication Overhead

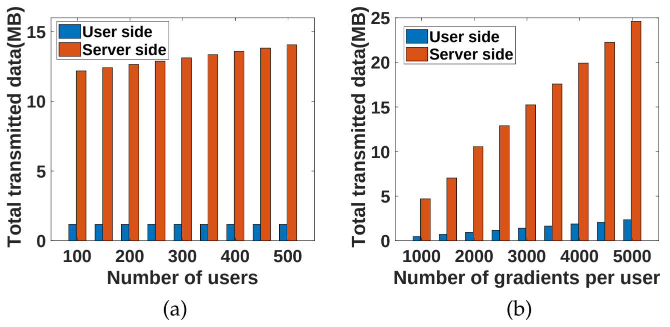


Fig. 13: Communication cost of $\texttt{PPFDL}_{\texttt{imd}}$.

In this section, we evaluate the communication overhead of $\texttt{PPFDL}_{\texttt{imd}}$. In $\texttt{PPFDL}_{\texttt{imd}}$, server S_0 primarily interacts with server S_1, in other words, the communication overhead of the two servers can be approximated to be equal. Therefore, in the experiments, we simply consider the communication overhead of the server side as the sum of the costs of two servers, and no longer analyze the cost of each server separately. Fig. 13 shows the communication cost of $\texttt{PPFDL}_{\texttt{imd}}$ with a different number of users/gradients per user. We can see that the overhead of the user side is much smaller than the cost of the server-side. This is because in $\texttt{PPFDL}_{\texttt{imd}}$, each user is only required to send their own encrypted gradient set to the server, and then it can go offline. However, multiple interactions between the two servers are required to complete the update of each user's reliability and the update of the aggregated results for each gradient.

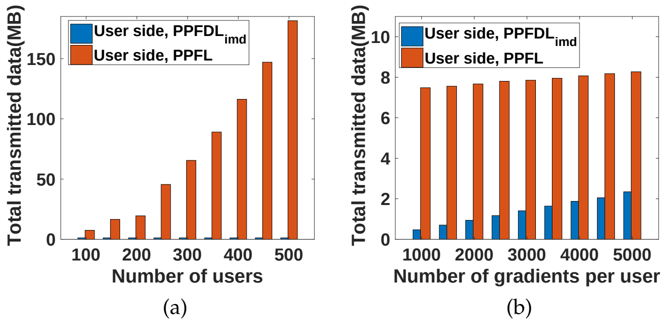


Fig. 14: Comparison of each user's communication cost between $\texttt{PPFDL}_{\texttt{imd}}$ and PPFL [12].

Fig. 14 shows the comparison of each user's communication cost between $\texttt{PPFDL}_{\texttt{imd}}$ and PPFL [12]. We can observe that the communication overhead of $\texttt{PPFDL}_{\texttt{imd}}$ is much lower than PPFL as the number of users/gradients per user increases. In PPFL [12], each user is required to send its encrypted gradient set to the cloud. Besides, each user still needs to send the key used to encrypt the plaintext to the server through the key sharing protocol. Moreover, when some users are offline during the training process, in order to ensure successful decryption, all online users need to jointly perform a secret sharing protocol to recover the offline users' secrets and send them to the server. This results in huge communication overhead for each user. However, in $\texttt{PPFDL}_{\texttt{imd}}$, each user is only required to send their own encrypted gradient set to the server, and all ciphertext processing is done by two servers interacting with each other. Therefore, $\texttt{PPFDL}_{\texttt{imd}}$ is very friendly for end users with limited communication power.

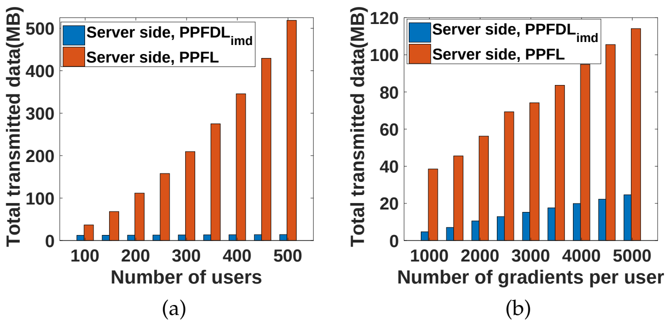


Fig. 15: Comparison of server's communication cost between $\texttt{PPFDL}_{\texttt{imd}}$ and PPFL [12].

Fig. 15 shows the comparison of the server's communication cost between $\texttt{PPFDL}_{\texttt{imd}}$ and PPFL [12], where the server's computation cost of $\texttt{PPFDL}_{\texttt{imd}}$ represents the sum of the cost of S_0 and server S_1. It can be observed that the communication overhead of $\texttt{PPFDL}_{\texttt{imd}}$ is much better than PPFL [12]. In PPFL, in addition to receiving encrypted gradients sent by all users, the server also needs to receive each user's secret shares to other users and broadcast these shares to all online users. Moreover, the server needs to additionally interact with each online user for successful decryption of the aggregated results, and ensuring the confidentially of all users' local gradients. This makes the server's communication complexity as high as $O(N^2+M\times N)$ in the worst case. However, as analysis before, the communication complexity of $\texttt{PPFDL}_{\texttt{imd}}$ is only $O(N+M)$. Hence, the communication overhead of $\texttt{PPFDL}_{\texttt{imd}}$ is much smaller than the cost of PPFL.

6 RELATED WORK

In this section, we introduce the existing works of privacy-preserving federated learning. Besides, since we apply truth discovery in a secure manner to federated learning, we also review the related works of privacy-preserving truth discovery.

6.1 Privacy-preserving Federated Learning

Existing privacy-preserving federated deep learning approaches are mostly evolved from three underlying techniques: *Differential Privacy* [3], [8], *Homomorphic Encryption*

[10], [11] and *Secure Multi-Party Computation (SMC)* [12], [13]. For example, *Phong et. al* [10] proposed a privacy-preserving deep learning system under collaborative deep training, where LWE-based homomorphic encryption is adopted to protect the privacy of shared local gradients between multiple participants. Besides, asynchronous stochastic gradient descent and secure channel such as TLS/SSL are used in the training process for efficiency and secure communication. *Keith Bonawitz et al.* [12] presented a practical and secure data aggregation protocol in the federated training process, where the authors utilized the SMC to guarantee the privacy of local gradients submitted to the server. Besides, considering that users have only sporadic access to power and network connectivity, they also exploited secret sharing and key exchange protocol to achieve robustness to users dropping out. More recently, *Yu et. al* [52] designed concentrated differential privacy(CDP), a universal differential privacy technology to optimize both privacy loss and model accuracy for training neural networks. Given a privacy budget, the authors demonstrated the outstanding performance of CDP in terms of privacy loss accounting, training efficiency and model quality.

In summary, Homomorphic Encryption, such as Fully Homomorphic Encryption (FHE), can be used for private deep learning based on its homomorphism of multiplication and addition in the ciphertext domain. However, FHE is not suitable for deep learning with large-scale users and high-dimensional training sets. To ensure the expected security level, it leads to huge computation overhead. SMC performs better in terms of computational overhead. However, it usually requires multiple interactions between participants to complete targeted calculations. This mode not only causes high communication overhead, but also requires all users to stay online during the whole workflow. Compared with homomorphic encryption and SMC, differential privacy is outstanding in cost. It only requires each user to add noise to the local dataset before uploading them. However, differential privacy always needs to make a trade-off between security and accuracy. A strong security requirement must sacrifice certain prediction accuracy. The result [16] has shown that current mechanisms for differentially private deep learning rarely offer acceptable utility-privacy trade-offs for complex learning tasks: the adversary can still easily recover the user's sensitive data if the model is acceptable for accuracy. Moreover, work [6] has shown that distributed, federated, or decentralized deep learning approach is fundamentally broken even all parameters shared between participants are obfuscated via differential privacy.

On the other hand, all of the above solutions do not consider a fundamental issue of *irregular users*. To address this problem, *Zhao et al.* [14] proposed SecProbe, the first privacy-preserving collaborative deep learning with irregular participants, which exploited differential privacy-based technologies to perturb the objective function of targeted DNNs, and claimed that SecProbe can ensure a good balance between accuracy and security. However, as described above, the state-of-the-art results [6], [16] have shown that current mechanisms for differentially private deep learning rarely offer acceptable utility-privacy trade-offs for complex learning tasks. Compared with these approaches, Our PPFDL integrates the technologies of additive homomorphism and Yao's garbled circuits, which can protect the privacy of users' local gradients, aggregated results as well as each user's reliability. PPFDL also designs a novel solution to reduce the impact of users with data of low quality in the training process. Moreover, since PPFDL only requires each user to upload its encrypted local gradients to the server with the same public key, our model is very friendly for end-users with limited computing power. Besides, PPFDL is also robustness to users dropping out during the whole implementation.

6.2 Privacy-preserving Truth Discovery

Truth Discovery (TD) has shown excellent performance in estimating real data from numerous heterogeneous data. To protect data privacy, many Privacy-Preserving Truth Discovery (PPTD) approaches have been also proposed. In summary, existing works focused on finding the optimal trade-off between security and efficiency in the private TD process. *Miao et al.* [25] proposed the first PPTD framework for crowdsensing systems. Based on the threshold Paillier Cryptosystem [53], it requires the server to interact with multiple data providers to complete the TD, so as to ensure the confidentiality of providers' data in this process. This solution has two shortcomings: one is that multiple interactions inevitably consume substantial resources of each data provider; the other is that the privacy of the aggregated results is not considered. To combat this, many efficient PPTD [26], [40], [41], [42], [54], [55], [56] are designed and applied to different application scenarios, among which the typical schemes are [26], [42], [56], [57]. These works designed more efficient solutions than work [25], by incorporating various technologies, such as Data aggregation [58], Homomorphic encryption [53] and Secure multiparty computing [36]. However, most of them require all users to be online all the time to successfully decrypt the final aggregate value, which is vulnerable once communication between users is affected. Recently, works [41], [55] resorted to differential privacy technology to implement PPTD with excellent computing performance, which allows users to add random noise to data before submitting them. However, these works do not consider the privacy of user's reliability, and also suffer from an inherent trade-off between utility and privacy.

In this paper, we proposed a PPTD called $\texttt{Meth}_{\texttt{IU}}$. Compared with previous works, it can guarantee the privacy of all user-related information, including the gradient and reliability of each user, as well as the aggregation results. $\texttt{Meth}_{\texttt{IU}}$ is also very friendly for end-users with limited computing power since most of the computations in $\texttt{Meth}_{\texttt{IU}}$ are done by the servers. Moreover, for practicality, our $\texttt{Meth}_{\texttt{IU}}$ is also robust to users dropping out during the entire TD process with various unpredictable reasons.

7 Conclusion

In this paper, we have proposed PPFDL, which can provide a high level of privacy protection for each user's local gradients data, user's reliability, and aggregated results. Besides, PPFDL also designed a novel strategy to reduce the negative impact of irregular users on the accuracy of training results.

Experiment results also demonstrate the high performance of PPFDL in terms of accuracy, computation and communication overheads. In further works, we intend to further improve the training accuracy of PPFDL and find ways to reduce the computation and communication overhead of PPFDL.

ACKNOWLEDGMENT

This work is supported by the National Key R&D Program of China under Grants 2017YFB0802300 and 2017YFB0802000, the National Natural Science Foundation of China under Grants 61802051, 61772121, 61728102, 61972094 and 61472065, the Peng Cheng Laboratory Project of Guangdong Province PCL2018KP004, the Guangxi Key Laboratory of Cryptography and Information Security under Grant GCIS201804.

REFERENCES

[1] Y. Zhang, C. Xu, X. Liang, H. Li, Y. Mu, and X. Zhang, "Efficient public verification of data integrity for cloud storage systems from indistinguishability obfuscation," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 3, pp. 676–688, 2017.

[2] G. Xu, H. Li, Y. Dai, K. Yang, and X. Lin, "Enabling efficient and geometric range query with access control over encrypted spatial data," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 4, pp. 870–885, 2019.

[3] R. Shokri and V. Shmatikov, "Privacy-preserving deep learning," in *Proceedings of ACM CCS*, 2015, pp. 1310–1321.

[4] N. Papernot, M. Abadi, U. Erlingsson, I. Goodfellow, and K. Talwar, "Semi-supervised knowledge transfer for deep learning from private training data," *arXiv preprint arXiv:1610.05755*, 2016.

[5] N. Phan, Y. Wang, X. Wu, and D. Dou, "Differential privacy preservation for deep auto-encoders: an application of human behavior prediction," in *Proceedings of AAAI*, 2016.

[6] B. Hitaj, G. Ateniese, and F. Perez-Cruz, "Deep models under the gan: information leakage from collaborative deep learning," in *Proceedings of ACM CCS*, 2017, pp. 603–618.

[7] Y. Aono, T. Hayashi, L. Wang, S. Moriai *et al.*, "Privacy-preserving deep learning via additively homomorphic encryption," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 5, pp. 1333–1345, 2017.

[8] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep learning with differential privacy," in *Proceedings of ACM CCS*, 2016, pp. 308–318.

[9] G. Xu, H. Li, H. Ren, K. Yang, and R. H. Deng, "Data security issues in deep learning: Attacks, countermeasures, and opportunities," *IEEE Communications Magazine*, vol. 57, no. 11, pp. 116–122, 2019.

[10] L. T. Phong, Y. Aono, T. Hayashi, L. Wang, and S. Moriai, "Privacy-preserving deep learning via additively homomorphic encryption," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 5, pp. 1333–1345, 2018.

[11] P. Mohassel and Y. Zhang, "Secureml: A system for scalable privacy-preserving machine learning," in *Proceedings of IEEE Security and Privacy*, 2017, pp. 19–38.

[12] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, "Practical secure aggregation for privacy-preserving machine learning," in *Proceedings of ACM CCS*, 2017, pp. 1175–1191.

[13] B. Jayaraman, L. Wang, D. Evans, and Q. Gu, "Distributed learning without distress: Privacy-preserving empirical risk minimization," in *Proceedings of the NeurIPS*, 2018, pp. 6343–6354.

[14] L. Zhao, Y. Zhang, Q. Wang, Y. Chen, C. Wang, and Q. Zou, "Privacy-preserving collaborative deep learning with unreliable participants," *IEEE Transactions on Information Forensics and Security*, 2019, DOI:10.1109/TIFS.2019.2939713.

[15] J. Zhang, Z. Zhang, X. Xiao, Y. Yang, and M. Winslett, "Functional mechanism: regression analysis under differential privacy," *Proceedings of VLDB Endowment*, vol. 5, no. 11, pp. 1364–1375, 2012.

[16] B. Jayaraman and D. Evans, "Evaluating differentially private machine learning in practice," in *Proceedings of USENIX Security*, 2019, pp. 1–18.

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

[18] G. Xu, H. Li, and R. Lu, "Practical and privacy-aware truth discovery in mobile crowd sensing systems," in *Proceedings of ACM CCS*, 2018, pp. 2312–2314.

[19] V. Nikolaenko, S. Ioannidis, U. Weinsberg, M. Joye, N. Taft, and D. Boneh, "Privacy-preserving matrix factorization," in *Proceedings of ACM CCS*, 2013, pp. 801–812.

[20] V. Nikolaenko, U. Weinsberg, S. Ioannidis, M. Joye, D. Boneh, and N. Taft, "Privacy-preserving ridge regression on hundreds of millions of records," in *Proceedings of IEEE Security and Privacy*, 2013, pp. 334–348.

[21] G. Xu, H. Li, S. Liu, K. Yang, and X. Lin, "Verifynet: Secure and verifiable federated learning," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 911–926, 2020.

[22] J. Chen, X. Pan, R. Monga, S. Bengio, and R. Jozefowicz, "Revisiting distributed synchronous sgd," *arXiv preprint arXiv:1604.00981*, 2016.

[23] P. Mohassel, M. Rosulek, and Y. Zhang, "Fast and secure three-party computation: The garbled circuit approach," in *Proceedings of ACM CCS*, 2015, pp. 591–602.

[24] C. Peikert, V. Vaikuntanathan, and B. Waters, "A framework for efficient and composable oblivious transfer," in *Proceedings of CRYPTO*. Springer, 2008, pp. 554–571.

[25] C. Miao, W. Jiang, L. Su, Y. Li, S. Guo, Z. Qin, H. Xiao, J. Gao, and K. Ren, "Cloud-enabled privacy-preserving truth discovery in crowd sensing systems," in *Proceedings of ACM SenSys*, 2015, pp. 183–196.

[26] G. Xu, H. Li, C. Tan, D. Liu, Y. Dai, and K. Yang, "Achieving efficient and privacy-preserving truth discovery in crowd sensing systems," *Computers and Security*, vol. 69, pp. 114–126, 2017.

[27] W. Luping, W. Wei, and L. Bo, "Cmfl: Mitigating communication overhead for federated learning," in *Proceedings of IEEE ICDCS*, 2019, pp. 954–964.

[28] V. Smith, C.-K. Chiang, M. Sanjabi, and A. S. Talwalkar, "Federated multi-task learning," in *Proceedings of NeurIPS*, 2017, pp. 4424–4434.

[29] K. Hsieh, A. Harlap, N. Vijaykumar, D. Konomis, G. R. Ganger, P. B. Gibbons, and O. Mutlu, "Gaia: Geo-distributed machine learning approaching {LAN} speeds," in *Proceedings of USENIX NSDI*, 2017, pp. 629–647.

[30] F. Sattler, S. Wiedemann, K.-R. Müller, and W. Samek, "Robust and communication-efficient federated learning from non-iid data," *IEEE transactions on neural networks and learning systems*, 2019.

[31] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra, "Federated learning with non-iid data," *arXiv preprint arXiv:1806.00582*, 2018.

[32] K. Hsieh, A. Phanishayee, O. Mutlu, and P. B. Gibbons, "The non-iid data quagmire of decentralized machine learning," *arXiv preprint arXiv:1910.00189*, 2019.

[33] J. Liu, M. Juuti, Y. Lu, and N. Asokan, "Oblivious neural network predictions via minionn transformations," in *Proceedings of ACM CCS*, 2017, pp. 619–631.

[34] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the NeurIPS*, 2012, pp. 1097–1105.

[35] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Proceedings of EUROCRYPTO*. Springer, 1999, pp. 223–238.

[36] Y. Lindell and B. Pinkas, "A proof of security of yaos protocol for two-party computation," *Journal of cryptology*, vol. 22, no. 2, pp. 161–188, 2009.

[37] Y. Huang, D. Evans, J. Katz, and L. Malka, "Faster secure two-party computation using garbled circuits," in *Proceedings of USENIX Security*, vol. 201, no. 1, 2011, pp. 331–335.

[38] C. Liu, X. S. Wang, K. Nayak, Y. Huang, and E. Shi, "Oblivm: A programming framework for secure computation," in *Proceedings of IEEE Security and Privacy*, 2015, pp. 359–376.

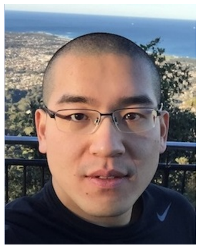
[39] E. M. Songhori, S. U. Hussain, A.-R. Sadeghi, T. Schneider, and F. Koushanfar, "Tinygarble: Highly compressed and scalable sequential garbled circuits," in *Proceedings of IEEE Security and Privacy*, 2015, pp. 411–428.

[40] C. Cai, Y. Zheng, and C. Wang, "Leveraging crowdsensed data streams to discover and sell knowledge: A secure and efficient realization," in *Proceedings of IEEE ICDCS*, 2018, pp. 589–599.
[41] Y. Li, H. Xiao, Z. Qin, C. Miao, L. Su, J. Gao, K. Ren, and B. Ding, "Towards differentially private truth discovery for crowd sensing systems," *arXiv preprint arXiv:1810.04760*, 2018.
[42] C. Miao, L. Su, W. Jiang, Y. Li, and M. Tian, "A lightweight privacy-preserving truth discovery framework for mobile crowd sensing systems," in *Proceedings of IEEE INFOCOM*, 2017, pp. 1–9.
[43] Z. Brakerski, "Fully homomorphic encryption without modulus switching from classical gapsvp," in *Proceedings of CRYPTO*. Springer, 2012, pp. 868–886.
[44] J.-S. Coron, D. Naccache, and M. Tibouchi, "Public key compression and modulus switching for fully homomorphic encryption over the integers," in *Proceedings of EUROCRYPT*. Springer, 2012, pp. 446–464.
[45] J. W. Bos, K. Lauter, J. Loftus, and M. Naehrig, "Improved security for a ring-based fully homomorphic encryption scheme," in *IMA International Conference on Cryptography and Coding*. Springer, 2013, pp. 45–64.
[46] C. Gentry, J. Groth, Y. Ishai, C. Peikert, A. Sahai, and A. Smith, "Using fully homomorphic hybrid encryption to minimize non-interative zero-knowledge proofs," *Journal of Cryptology*, vol. 28, no. 4, pp. 820–843, 2015.
[47] D. Catalano and D. Fiore, "Using linearly-homomorphic encryption to evaluate degree-2 functions on encrypted data," in *Proceedings of ACM CCS*, 2015, pp. 1518–1529.
[48] T. T. Phuong *et al.*, "Privacy-preserving deep learning via weight transmission," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 11, pp. 3003–3015, 2019.
[49] X. L. Dong, L. Berti-Equille, and D. Srivastava, "Truth discovery and copying detection in a dynamic world," *Proceedings of VLDB Endowment*, vol. 2, no. 1, pp. 562–573, 2009.
[50] X. Yin and W. Tan, "Semi-supervised truth discovery," in *Proceedings of ACM WWW*, 2011, pp. 217–226.
[51] A. Shamir, *How to share a secret*. Communications of the ACM, 1979.
[52] L. Yu, L. Liu, C. Pu, M. E. Gursoy, and S. Truex, "Differentially private model publishing for deep learning," in *proceedings of IEEE Security & Privacy*, 2019, pp. 309–326.
[53] I. Damgård and M. Jurik, "A generalisation, a simpli. cation and some applications of paillier′s probabilistic public-key system," in *Proceedings of PKC*. Springer, 2001, pp. 119–136.
[54] X. Tang, C. Wang, X. Yuan, and Q. Wang, "Non-interactive privacy-preserving truth discovery in crowd sensing applications," in *Proceedings of IEEE INFOCOM*, 2018, pp. 1988–1996.
[55] Y. Li, C. Miao, L. Su, J. Gao, Q. Li, B. Ding, Z. Qin, and K. Ren, "An efficient two-layer mechanism for privacy-preserving truth discovery," in *Proceedings of ACM SIGKDD*, 2018, pp. 1705–1714.
[56] Y. Zheng, H. Duan, X. Yuan, and C. Wang, "Privacy-aware and efficient mobile crowdsensing with truth discovery," *IEEE Transactions on Dependable and Secure Computing*, 2017.
[57] G. Xu, H. Li, S. Liu, M. Wen, and R. Lu, "Efficient and privacy-preserving truth discovery in mobile crowd sensing systems," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 4, pp. 3854–3865, 2019.
[58] J. Zhou, Z. Cao, X. Dong, and X. Lin, "Security and privacy in cloud-assisted wireless wearable communications: Challenges, solutions, and future directions," *IEEE wireless Communications*, vol. 22, no. 2, pp. 136–144, 2015.

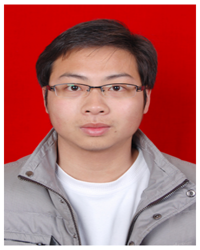
Guowen Xu (S'15) is currently a Ph.D. student at the School of Computer Science and Engineering, University of Electronic Science and Technology of China , China. His research interests include searchable encryption and privacy-preserving issues in Deep Learning.

Hongwei Li (M'12-SM'18) is currently the Head and a Professor at Department of Information Security, School of Computer Science and Engineering, University of Electronic Science and Technology of China. He received the Ph.D. degree from University of Electronic Science and Technology of China in June 2008. He worked as a Postdoctoral Fellow at the University of Waterloo from October 2011 to October 2012. His research interests include network security and applied cryptography. He is the Senior Member of IEEE, the Distinguished Lecturer of IEEE Vehicular Technology Society.

Yun Zhang (S'19) is working toward the Masters degree at the School of Computer Science and Engineering, University of Electronic Science and Technology of China. His research interests include Trust Excuation Environment, Machine Learning.



Shengmin Xu (M'18) is currently a research fellow at School of Information System, Singapore Management University, Singapore. His research interests include cryptography and information security.



Jianting Ning (M'17) is currently a research fellow at School of Information Systems, Singapore Management University. His research interests include applied cryptography and information security, in particular, public key encryption, secure and privacy-preserving computation.

Robert H. Deng (F'17) is AXA Chair Professor of Cybersecurity, Singapore Management University. His research interests are in the areas of data security and privacy, cloud security and IoT security. He served on many editorial boards and conference committees, including the editorial boards of IEEE Security & Privacy Magazine, IEEE Transactions on Dependable and Secure Computing, IEEE Transactions on Information Forensics and Security, and Steering Committee Chair of the ACM Asia Conference on Computer and Communications Security. He is an IEEE Fellow.