A Practical Privacy-preserving Method in Federated Deep Learning

Yan Feng[*¶§], Xue Yang[*¶§✉], Weijun Fang[*¶§], Shu-Tao Xia[*], Xiaohu Tang[†], Jun Shao[‡], and Tao Xiong[‖]
[*]Tsinghua Univerisity
y-feng18@mails.tsinghua.edu.cn, nankaifwj@163.com, {yang.xue, xiast}@sz.tsinghua.edu.cn
[¶]Peng Cheng Laboratory
[†]Southwest Jiaotong University, xhutang@swjtu.edu.cn
[‡]Zhejiang Gongshang University, chn.junshao@gmail.com
[‖]Ant Financial Services Group, weilue.xt@alibaba-inc.com

Abstract—**Although federated learning improves privacy of training data by exchanging model updates rather than raw data, many research results show that sharing the model updates may still involve risks. To alleviate this problem, many privacy-preserving techniques have been introduced to federated learning. However, considering deep learning models in federated learning, the resulting schemes either cannot implement non-linear activation functions well, or cannot remain the same model accuracy as the original training, or suffer from unaffordable costs. In this paper, we customize a *practical privacy-preserving method for federated deep learning*, which is versatile and applicable to most state-of-the-art models, such as ResNet and DenseNet. In particular, this method can support non-linear activation functions well on the encrypted domain, hence supporting semi-trusted clients to efficiently train deep neural network locally over encrypted model iterates (i.e., protecting the privacy of the model for server-side). Meanwhile, it can be combined with the secret sharing technique to further ensure the semi-trusted server cannot obtain local gradients of each client (i.e., protect the privacy of training data for client-side). Detailed security analysis and extensive experiments demonstrate that the proposed method can achieve privacy-preservation without sacrificing model accuracy and introducing too much extra costs.**

I. Introduction

With the continued emergence of privacy breaches and data abuse [34], data privacy and security issues gradually impede the flourishing development of deep learning [38]. In order to mitigate the privacy concerns of users, *federated learning* (FL) [22] has recently been presented as a promising solution, where many clients collaboratively train a shared global model under the orchestration of a central server, while ensuring that each client's raw data is stored locally and not exchanged or transferred. Based on the type of clients, FL is divided into two settings [15]: the cross-device FL, where clients are mobile or edge devices, and the cross-silo FL, where clients are relatively reliable organizations (e.g., medical or financial institutions). In this paper, we focus on solving the challenges faced in the cross-silo FL that has received greatly interests recently.

Although FL improves the privacy of local training data by exchanging model updates (e.g., local gradients or updated parameters) rather than raw data, from a privacy angle, sharing even the model updates is still a well-known privacy risk [41], [25]. Thus, many existing researches consider the semi-trusted security model (i.e., semi-trusted server and/or semi-trusted clients) and focus on preventing the semi-trusted server and semi-trusted clients from obtaining local training data and real model, respectively. Specifically, they mainly adopt existing privacy-preserving techniques, like differential privacy (DP) [9], multi-party computation (MPC) [2] or homomorphic encryption (HE) [28] to support training or predicting on the encrypted or perturbed model. To the best of our knowledge, these techniques can be well performed in traditional machine learning, such as linear regression [38], [26] and k-means [33], to protect privacy, but cannot elegantly applied in deep learning. More specifically, for the privacy-preserving deep learning, they face the following issues [15], [38]:

- The HE-based schemes cannot address non-linear activation functions well on the encrypted domain;
- The computational and communication costs of the HE and MPC-based schemes are too high to be applied in practice;
- The DP-based schemes have to sacrifice the model accuracy to achieve privacy-preservation.

Therefore, the survey of FL [15] emphasizes that how to efficiently allow clients to train deep neural networks locally on encrypted models is still a challenge. Consequently, in this paper, we customize *a practical privacy-preserving method for federated deep learning*, where the main contributions are two-fold:

1) It addresses the operations of the non-linear activation functions (like *ReLU*) and the widely used loss functions (such as *mean squared error and cross-entropy loss functions*) on the encrypted domain, thereby supporting clients to train deep model locally without knowing true model updates or accessing the true model predictions (i.e., protect the privacy of the server-side). Besides, it can be further combined with the secret sharing technique to ensure the semi-trusted server cannot obtain local gradients of each client, thereby protecting the privacy of both server and client sides.
2) It theoretically ensures that the server can accurately

[§] Equal contribution
[✉]Corresponding author

recover model updates, meanwhile, after pretraining clients can use encrypted models to get true predictions. Besides, it provides a trade-off between model privacy and efficiency. For the most efficient case, the upper limit for extra computational and communication costs are constant times the original ones, which ensures the efficiency and usability of our method in practice. Furthermore, extensive experiments conducted on real-world data also demonstrate the accuracy and the efficiency of our method.

II. Preliminaries and Problem Statement

In this section, we first outline the concept of cross-silo FL and the Hadamard product. After that, we state the system model, threat model and design goals.

A. The Cross-Silo FL

In the cross-silo FL [15], clients are different organizations (e.g. medical or financial), the network connection is relatively stable and the network bandwidth is relatively large. That is, all clients are always available and can afford relatively large communication cost. Thus, the cross-silo FL allows all clients to join each iteration.

Formally, consider the FL with K clients, where the k-th client has the local training dataset $\mathcal{D}_k = \{(\boldsymbol{x}_i, \bar{\boldsymbol{y}}_i)\}_{i=1}^{N_k}$, $\boldsymbol{x}_i = (x_{i1}, \ldots, x_{id})^T$ and $\bar{\boldsymbol{y}}_i = (\bar{y}_{i1}, \ldots, \bar{y}_{ic})^T$ are the feature vector and the ground-truth label vector, respectively. Thus, the cross-silo FL aims to solving an optimization problem [22], [20]:

$$\min_{W} F(W) \triangleq \sum_{k=1}^{K} \frac{N_k}{N} F_k(W), \tag{1}$$

where W is the model parameter, N is the total sample size such that $N = \sum_{k=1}^{K} N_k$, and $F_k(W)$ is the local object of the k-th client such that

$$F_k(W) \triangleq \frac{1}{N_k} \sum_{i=1}^{N_k} \mathcal{L}\left(W; (\boldsymbol{x}_i, \bar{\boldsymbol{y}}_i)\right), \tag{2}$$

where $\mathcal{L}(\cdot;\cdot)$ is the specific loss function, like the mean square error loss function or cross entropy loss function.

In general, this optimization problem can be handled with stochastic gradient descent (SGD). Thus, each client first computes local gradients by adopting the SGD technique and returns them to the server for aggregation and updating

$$W_{t+1} \leftarrow W_t - \eta \sum_{k=1}^{K} \frac{N_k}{N} \nabla F_k(W_t), \tag{3}$$

where $\nabla F_k(W_t)$ is the local gradient on local data of k-th client at the current model W_t, and η is the learning rate.

B. Hadamard Product

The Hadamard product [13] takes two matrices of the same dimensions and produces another matrix of the same dimension as the operands.

Definition 1. *For two matrices A and B of the same dimension $m \times n$, the Hadamard product $A \circ B$ (or $A \odot B$) is a matrix of the same dimension as the operands, with elements given by*

$$(A \circ B)_{ij} = (A \odot B)_{ij} = (A)_{ij}(B)_{ij}$$

Besides, two properties of Hadamard product used in our scheme are given as follows:

- For any two matrices A and B, and a diagonal matrix D, we have
$$\begin{cases} D(A \circ B) = (DA) \circ B. \\ (A \circ B)D = (AD) \circ B. \end{cases} \tag{4}$$
- For any two column vectors $\boldsymbol{a}$ and $\boldsymbol{b}$, the corresponding Hadamard product is: $\boldsymbol{a} \circ \boldsymbol{b} = D_{\boldsymbol{a}}\boldsymbol{b}$,, where $D_{\boldsymbol{a}}$ is the corresponding diagonal matrix with the vector $\boldsymbol{a}$ as its main diagonal.

C. Problem Statement

In this part, we introduce the system and threat models considered in this paper, and identify our design goals.

1) System Model: As shown in Fig. 1, our system model mainly includes two components: a server and a number of clients, where the corresponding roles are described as follows:

- **The Server** is responsible for initializing the model and assisting clients in training the global model. Particularly, in order to protect the privacy, the server sends the noisy models to clients for training.
- **Clients** have their own local training data and want to collaboratively train a global model. Specifically, each client computes local gradients with their own data and the noisy model received from the server, and then returns noisy local gradients to the server for aggregating and updating.

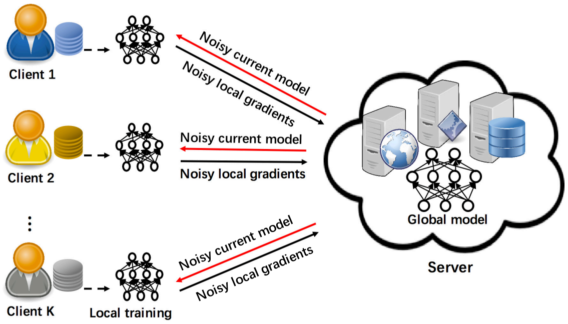

Fig. 1: System architecture of the proposed scheme.

2) Threat Model and Design Goals: Similar to [15], we assume the server and clients are honest-but-curious [12], [28], which means that they honestly follow the underlying scheme, but attempt to infer other entities' data privacy independently. Consequently, the design goals of our method mainly include the following three aspects:

- **Functionality**: Each client can train models locally on the noisy model, meanwhile, the server can recover the exact model updates.

- **Confidentiality**: The proposed method should ensure the confidentiality of both the server and every client. On the one hand, clients cannot know true model updates and prediction, as well as the well-trained model parameters. On the other hand, the server cannot obtain local gradients of individual client from the received information.
- **Efficiency**: The proposed method should minimize extra computation and communication overhead without reducing model accuracy.

III. Proposed Scheme

In this section, we introduce our method, which mainly includes four parts: parameter perturbation, noisy gradient computation, model update and data inference. The overall framework is described as follows:

- *Model training*: In our privacy-preserving federated deep learning, the model training mainly includes the following three phases:
  1) The server encrypts the global model iterates with one-time random numbers and then sends it to each clients for local training, which is described in Section III-A.
  2) Each client first trains the local model with local training data and the received encrypted model iterates to get local gradients. Then, the client perturbs local gradients with one-time random numbers that satisfy the condition that the sum of all clients' random numbers is zero, and returns the perturbed local gradients to the server. The details are shown in Section III-B.
  3) The server updates the current global model based on the received local gradients from all clients, which is introduced in Section III-C.

  The server and all clients interactively iterate the processes in Sections III-A-III-C until the convergence. Consequently, the server obtains well-trained model.
- *Data inference*: The server encrypts the well-trained model and sends it to all clients for data inference, which is given in Section III-D.

A. Parameter Perturbation

The parameter perturbation is performed by the server, which takes model parameters W and random noises R as inputs, and outputs noisy model parameters $\widehat{W}$[1]. In order to facilitate the understanding of our encryption method, we give a simple case of multiple layer perceptron (MLP) with ReLU non-linear activation. Then, we show this method can be easily applied on the state-of-the-art models such as ResNet[11] and DenseNet[14].

1) Multiple Layer Perceptron: Consider an MLP with L layers, where the parameter and the output of the l-th layer are denoted as $W^{(l)} \in \mathbb{R}^{n_l \times n_{l-1}}$ and $\boldsymbol{y}^{(l)} \in \mathbb{R}^{n_l}$, respectively, and n_l is the number of neurons in the l-th layer. Specifically, $\boldsymbol{y}^{(l)}$ is computed as

$$\boldsymbol{y}^{(l)} = \begin{cases} ReLU\left(W^{(l)}\boldsymbol{y}^{(l-1)}\right), & \text{for } 1 \leq l \leq L-1. \\ W^{(l)}\boldsymbol{y}^{(l-1)}, & \text{for } l = L. \end{cases} \tag{5}$$

where $ReLU(\cdot)$ is the operation of ReLU non-linear activation. Note that when $l = 1$, then $\boldsymbol{y}^{(0)}$ is the input of neural network $\boldsymbol{x} = (x_1, x_2, \ldots, x_d)^T$.

1. Parameter encryption. In order to protect the privacy of model parameters $W = \{W^{(l)}\}_{l=1}^{L}$, the server needs to encrypt or perturb them before distributing. Specifically, the parameter encryption consists of the following two steps:

- Key selection: The server randomly selects the multiplicative noisy vector $\boldsymbol{r}^{(l)} \in \mathbb{R}^{n_l}_{>0}$ for $1 \leq l \leq L-1$, the additive noisy vectors $\boldsymbol{\gamma}, \boldsymbol{r}_a \in \mathbb{R}^{n_L}$ with $\boldsymbol{r}_a$ has pairwise different components and there is a partition $\sqcup_{s=1}^{m}\{I_s\}$ of $\{1, 2, \ldots, n_L\}$ (that is $\cup_{s=1}^{m} I_s = \{1, 2, \ldots, n_L\}$ and for any $i \neq j$, $I_i \cap I_j = \emptyset$), such that for any i, j in the same I_s satisfy that $\boldsymbol{\gamma}_i = \boldsymbol{\gamma}_j$. Denote $\boldsymbol{\gamma}_{I_s} := \boldsymbol{\gamma}_i$ $(i \in I_s)$. The private key is $(\{\boldsymbol{r}^{(l)}\}_{l=1}^{L}, \{\boldsymbol{\gamma}_{I_s}\}_{s=1}^{m})$.
- Parameter encryption: For model parameters $W^{(l)}$, the server computes

$$\widehat{W}^{(l)} = \begin{cases} R^{(l)} \circ W^{(l)}, & \text{for } 1 \leq l \leq L-1; \\ R^{(l)} \circ W^{(l)} + R^{a}, & \text{for } l = L, \end{cases} \tag{6}$$

where $R^{(l)} \in \mathbb{R}^{n_l \times n_{l-1}}$ and $R^{a} \in \mathbb{R}^{n_L \times n_{L-1}}$ satisfy

$$R_{ij}^{(l)} = \begin{cases} \boldsymbol{r}_i^{(1)}, & \text{when } l = 1 \\ \boldsymbol{r}_i^{(l)}/\boldsymbol{r}_j^{(l-1)}, & \text{when } 2 \leq l \leq L-1 \\ 1/\boldsymbol{r}_j^{(L-1)}, & \text{when } l = L \end{cases} \tag{7}$$

$$R_{ij}^{a} = \boldsymbol{\gamma}_i \cdot \boldsymbol{r}_{a,i}, \tag{8}$$

where $i \in [1, n_l]$ and $j \in [1, n_{l-1}]$ in Eq. (7), and $i \in [1, n_L]$ and $j \in [1, n_{L-1}]$ in Eq. (8).

Finally, the server sends $\widehat{W} = \{\widehat{W}^{(l)}\}_{l=1}^{L}$ together with $\boldsymbol{r}_a$ to each client for local training.

2. Forward propagation. In order to facilitate the understanding of our parameter perturbation method, we introduce the forward propagation in advance. According to Eq. (5), each client computes the noisy output $\hat{\boldsymbol{y}} = \{\hat{\boldsymbol{y}}^{(l)}\}_{l=1}^{L}$ with the received $\widehat{W}$ and the sample $(\boldsymbol{x}, \bar{\boldsymbol{y}})$, i.e.,

$$\hat{\boldsymbol{y}}^{(l)} = \begin{cases} ReLU\left(\widehat{W}^{(l)}\hat{\boldsymbol{y}}^{(l-1)}\right), & \text{for } 1 \leq l \leq L-1. \\ \widehat{W}^{(l)}\hat{\boldsymbol{y}}^{(l-1)}, & \text{for } l = L. \end{cases} \tag{9}$$

Lemma 1 shows the important relations between the noisy outputs and the true outputs.

Lemma 1. *For any $1 \leq l \leq L$, the noisy output vector $\hat{\boldsymbol{y}}^{(l)}$ and the true output vector $\boldsymbol{y}^{(l)}$ have the following relationships*

$$\hat{\boldsymbol{y}}^{(l)} = \boldsymbol{r}^{(l)} \circ \boldsymbol{y}^{(l)}, \text{ when } 1 \leq l \leq L-1. \tag{10}$$

$$\hat{\boldsymbol{y}}^{(L)} = \boldsymbol{y}^{(L)} + \alpha\boldsymbol{\gamma} \circ \boldsymbol{r}_a = \boldsymbol{y}^{(L)} + \alpha\boldsymbol{r}. \tag{11}$$

where $\alpha = \sum_{j=1}^{n_{L-1}} \hat{\boldsymbol{y}}_j^{(L-1)}$ and $\boldsymbol{r} = \boldsymbol{\gamma} \circ \boldsymbol{r}_a$.

[1] In the rest of the paper, the notations with the symbol “ ˆ ” indicate the parameters that are affected by the noise, e.g., $\widehat{W}, \hat{\boldsymbol{y}}^{(l)}$.

Proof. Based on Eq. (7), we can deduce that

$$\begin{cases} R^{(1)} = D_{\boldsymbol{r}^{(1)}} E^{(1)}, \\ R^{(l)} D_{\boldsymbol{r}^{(l-1)}} = \boldsymbol{r}^{(l)} E^{(l)}, \text{ for } 2 \leq l \leq L-1, \\ R^{(L)} D_{\boldsymbol{r}^{(L-1)}} = E^{(L)}. \end{cases}$$

where $E^{(l)}$ is the $n_l \times n_{l-1}$ matrix whose entries are all 1s.

First, we prove Eq. (10) by induction. When $l = 1$, we can obtain that

$$\begin{aligned} \hat{\boldsymbol{y}}^{(1)} &= ReLU\left(\widehat{W}^{(1)}\boldsymbol{x}\right) = ReLU\left(\left(R^{(1)} \circ W^{(1)}\right)\boldsymbol{x}\right) \\ &= ReLU\left(\left(D_{\boldsymbol{r}^{(1)}}E^{(1)} \circ W^{(1)}\right)\boldsymbol{x}\right) \\ &= ReLU\left(D_{\boldsymbol{r}^{(1)}}W^{(1)}\boldsymbol{x}\right) = ReLU\left(\boldsymbol{r}^{(1)} \circ (W^{(1)}\boldsymbol{x})\right) \\ &\overset{(a)}{=} \boldsymbol{r}^{(1)} \circ ReLU(W^{(1)}\boldsymbol{x}) = \boldsymbol{r}^{(1)} \circ \boldsymbol{y}^{(1)}, \end{aligned}$$

where (a) follows from the condition $\boldsymbol{r}^{(1)} \in R^{n_1}_{>0}$.

Then, for $2 \leq l \leq L-1$, assuming $\hat{\boldsymbol{y}}^{(l-1)} = \boldsymbol{r}^{(l-1)} \circ \boldsymbol{y}^{(l-1)}$ by induction, we have

$$\begin{aligned} \hat{\boldsymbol{y}}^{(l)} &= ReLU\left(\widehat{W}^{(l)}\hat{\boldsymbol{y}}^{(l-1)}\right) \\ &= ReLU\left(\left(R^{(l)} \circ W^{(l)}\right)\left(\boldsymbol{r}^{(l-1)} \circ \boldsymbol{y}^{(l-1)}\right)\right) \\ &= ReLU\left(\left(R^{(l)} \circ W^{(l)}\right)D_{\boldsymbol{r}^{(l-1)}}\boldsymbol{y}^{(l-1)}\right) \\ &= ReLU\left(\left((R^{(l)}D_{\boldsymbol{r}^{(l-1)}}) \circ W^{(l)}\right)\boldsymbol{y}^{(l-1)}\right) \\ &= ReLU\left(\left(D_{\boldsymbol{r}^{(l)}}E^{(l)} \circ W^{(l)}\right)\boldsymbol{y}^{(l-1)}\right) \\ &= ReLU\left(D_{\boldsymbol{r}^{(l)}}W^{(l)}\boldsymbol{y}^{(l-1)}\right) \\ &= ReLU\left(\boldsymbol{r}^{(l)} \circ (W^{(l)}\boldsymbol{y}^{(l-1)})\right) = \boldsymbol{r}^{(l)} \circ \boldsymbol{y}^{(l)}. \end{aligned}$$

Thus the Eq. (10) is proved. Furthermore, we have

$$\begin{aligned} \hat{\boldsymbol{y}}^{(L)} &= \widehat{W}^{(L)}\hat{\boldsymbol{y}}^{(L-1)} = (R^{(L)} \circ W^{(L-1)} + R^a)\hat{\boldsymbol{y}}^{(L-1)} \\ &= (R^{(L)} \circ W^{(l)})(\boldsymbol{r}^{(L-1)} \circ \boldsymbol{y}^{(L-1)}) + R^a\hat{\boldsymbol{y}}^{(L-1)} \\ &= (R^{(L)}D_{\boldsymbol{r}^{(L-1)}}) \circ W^{(L)}\boldsymbol{y}^{(L-1)}) + R^a\hat{\boldsymbol{y}}^{(L-1)} \\ &= (E^{(L)} \circ W^{(L)})\boldsymbol{y}^{(L-1)} + R^a\hat{\boldsymbol{y}}^{(L-1)} \\ &= W^{(L)}\boldsymbol{y}^{(L-1)} + R^a\hat{\boldsymbol{y}}^{(L-1)} \\ &= \boldsymbol{y}^{(L)} + \alpha\boldsymbol{\gamma} \circ \boldsymbol{r}_a = \boldsymbol{y}^{(L)} + \alpha\boldsymbol{r}. \end{aligned}$$

□

2) Convolutional Neural Networks: Convolutional Neural Networks (CNN) have proven to be effective in many computer vision and natural language processing tasks. In this section, we continue to demonstrate how to apply our encryption method to CNNs. Although convolution operation can be applied to any dimensional input, we focus on 3-dimensional inputs since we are most concerned about image data. For any input $I^{(l)} \in \mathbb{R}^{c_l \times h_l \times w_l}$, where c_l, w_l and h_l are the number of channels, weight and height, respectively, the convolution operation, denoted by $I^{(l+1)} = I^{(l)} * W^{(l)}$, is defined as

$$I^{(l+1)}_{k,i,j} = \sum_{c'=1}^{c_l}\sum_{i'=1}^{f}\sum_{j'=1}^{f} W^{(l)}_{k,c',i',j'} I^{(l)}_{c',i+i',j+j'} \tag{12}$$

where $W^{(l+1)} \in \mathbb{R}^{c_{l+1} \times c_l \times f \times f}$ is a set of c_{l+1} filters with each filter of shape $c_l \times f \times f$, and $I^{(l+1)} \in \mathbb{R}^{c_{l+1} \times h_{l+1} \times w_{l+1}}$ is the output image.

From the above, convolutional layer can be implemented with a convolution operation followed by a non-linear function, and a CNN can be constructed by interweaving several convolutoinal and spatial pooling layers. At last, the CNN ends with a fully-connected layer for regression or classification tasks. In order to apply our encryption method to CNNs, we choose ReLU as the non-linear function and MaxPooling as the spatial pooling layer. In fact, [21] proved the equivalence of convolutional and fully connected operations. Therefore, the multiplicative noise introduced for MLP will not affect the operation of convolutional layers, which can be adopted with small alteration.

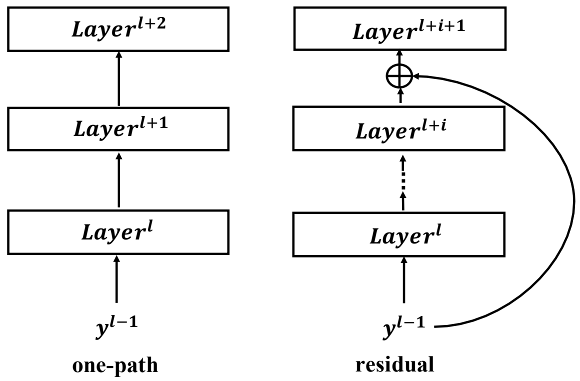

Fig. 2: Common connections in neural networks. ⊕ can be element-wise summation, element-wise multiplication or concatenation. In this paper we choose concatenation.

Section III-A1 mainly discusses about the "one-path" connection, i.e., the input of l-th layer is exactly the output of $(l-1)$-th layer. Most state-of-the-art CNN models, e.g., ResNet [11] and DenseNet [14], however, utilize residual connections in their structures, i.e., the l-th layer takes the output of the multiple precedent layers as input, as illustrated in Fig. 2. There are various kind of residual connections, for simplicity, we choose to connect the precedent outputs over the channel dimension.

1. Parameter encryption. Similar to Section III-A1, the parameter encryption consists of the following two steps:

- Key selection: The server randomly selects the multiplicative noisy vector $\boldsymbol{r}^{(l)} \in \mathbb{R}^{c_l}_{>0}$ for $1 \leq l \leq L-1$, the additive noisy vectors $\boldsymbol{\gamma}, \boldsymbol{r}_a \in \mathbb{R}^{n_L}$ with $\boldsymbol{r}_a$ has pairwise different components and the components of $\boldsymbol{\gamma}$ may not be independent. Similar to the MLP, the parameter $\boldsymbol{\gamma}$ satisfies that for any i, j in the same I_s, $\gamma_i = \gamma_j$. The private key is $(\{\boldsymbol{r}^{(l)}\}_{l=1}^{L}, \{\boldsymbol{\gamma}_{I_s}\}_{s=1}^{m})$.
- Parameter encryption: For the current model parameter $W^{(l)}$, the server computes

$$\widehat{W}^{(l)} = \begin{cases} R^{(l)} \circ W^{(l)}, \text{ for } 1 \leq l \leq L-1, \\ R^{(l)} \circ W^{(l)} + R^a, \text{ for } l = L, \end{cases} \tag{13}$$

where $R^{(l)} \in \mathbb{R}^{c_l \times c_{l-1} \times f \times f}$ satisfies

$$R^{(l)}_{k,c',i',j'} = \begin{cases} \boldsymbol{r}^{(l)}_k, \text{ when } l = 1, \\ \boldsymbol{r}^{(l)}_k / \boldsymbol{r}^{(l,in)}_{c'} \text{ when } 2 \le l \le L-1. \end{cases} \tag{14}$$

Note that $k \in [1, c_l]$, $c' \in [1, c_{l-1}]$ $i' \in [1, f]$ and $j' \in [1, f]$. $\boldsymbol{r}^{(l,in)} = \left(\boldsymbol{r}^{(m)}\right)_{m \in P(l)}$ is the concatenate vector of all vectors $\boldsymbol{r}^{(m)}$ for $m \in P(l)$, and $P(l)$ denotes the set of layers connected with the l-th layer.

Besides, $R^L, R^a \in \mathbb{R}^{n_L \times (c_{L-1} h_{L-1} w_{L-1})}$ satisfy

$$R^{(L)}_{ij} = 1/\boldsymbol{r}^{(L-1)}_{\lfloor j/(h_{L-1} w_{L-1}) \rfloor} \tag{15}$$

$$R^{a}_{ij} = \boldsymbol{\gamma}_i \cdot \boldsymbol{r}_{a,i}, \tag{16}$$

where $i \in [1, n_L]$ and $j \in [1, n_{L-1}]$.

It is worth noting that compared with the noisy vector of MLP, we can ignore the spatial dimensions (i.e., height and width) and just impose the same noisy value on them.

2. Forward propagation. Similar to the MLP, each client computes the noisy output of each layer based on Eq.(12), and the relation between the noisy outputs and true outputs is given in Lemma 2.

Lemma 2. *For CNN, the noisy outputs and true outputs satisfy the following relation:*

- *For the l-th layer, the corresponding noisy output $\hat{I}^{(l)}_k$ is a matrix for the k-th channel, where each element in $\hat{I}^{(l)}_k$ is represented as*

$$\hat{I}^{(l)}_{k,i',j'} = \boldsymbol{r}^{(l)}_k I^{(l)}_{k,i',j'}, \quad \text{for} \quad l = 1, 2, \ldots, L-1. \tag{17}$$

- *For the last fully connected layer, the final noisy prediction vector $\hat{\boldsymbol{y}}^{(L)}$ is computed as*

$$\hat{\boldsymbol{y}}^{(L)} = \boldsymbol{y}^{(L)} + \alpha \boldsymbol{\gamma} \circ \boldsymbol{r}_a = \boldsymbol{y}^{(L)} + \alpha \boldsymbol{r}, \tag{18}$$

where $\alpha = \sum_{j=1}^{n_{L-1}} \text{Flatten}(\hat{I}^{(L-1)})_j$, and the $\text{Flatten}(\hat{I}^{(L-1)})$ means expanding $\hat{I}^{(L-1)}$ into a vector along the channel, height and width dimensions, which implies $n_{L-1} = c_{L-1} h_{L-1} w_{L-1}$.

Proof. As defined in Section III-A2, the convolutional layer is implemented as a convolution operation followed by a ReLU. Due to the existence of residual connection, the l-th layer may be connected to a set of preceding layers, denoted as $P(l)$. Obviously, the input of the l-th convolutional layer, denoted as $I^{(l,in)} \in \mathbb{R}^{c_{l,in} \times h_{l,in} \times w_{l,in}}$, is the concatenation of the output of layers in set $P(l)$ along the channel dimension, which can be represented as

$$I^{(l,in)} = \underbrace{\left[I^{(m)}_{1,i,j}; I^{(m)}_{2,i,j}; \ldots; I^{(m)}_{c_m,i,j} \right]_{m \in P(l)}}_{\text{Concatenation along the channel dimension}}.$$

Therefore, the output of the l-th convolutional layer $I^{(l)} \in \mathbb{R}^{c_l \times h_l \times w_l}$ is defined as:

$$\begin{aligned} I^{(l)}_{k,i,j} &= ReLU\left(\sum_{c'=1}^{c_{l,in}} \sum_{i'=1}^{f} \sum_{j'=1}^{f} W^{(l)}_{k,c',i',j'} I^{(l,in)}_{c',i+i',j+j'} \right) \\ &= ReLU\left(\sum_{c',i',j'} W^{(l)}_{k,c',i',j'} I^{(l,in)}_{c',i+i',j+j'} \right), \end{aligned}$$

where $W^{(l)} \in \mathbb{R}^{c_l \times c_{l,in} \times f \times f}$ is the parameter of the l-th layer.

In what follows, we prove Eq. (17) by induction:

- When l = 1, we have

$$\begin{aligned} \hat{I}^{(1)}_{k,i,j} &= ReLU\left(\sum_{c',i',j'} \widehat{W}^{(1)}_{k,c',i',j'} I^{(1,in)}_{c',i+i',j+j'} \right) \\ &= ReLU\left(\sum_{c',i',j'} R^{(1)}_{k,c',i',j'} W^{(1)}_{k,c',i',j'} I^{(1,in)}_{c',i+i',j+j'} \right) \\ &= ReLU\left(\sum_{c',i',j'} \boldsymbol{r}^{(1)}_k W^{(1)}_{k,c',i',j'} I^{(1,in)}_{c',i+i',j+j'} \right) \\ &= \boldsymbol{r}^{(1)}_k ReLU\left(\sum_{c',i',j'} W^{(1)}_{k,c',i',j'} I^{(1,in)}_{c',i+i',j+j'} \right) \\ &= \boldsymbol{r}^{(1)}_k I^{(1)}_{k,i,j}. \end{aligned}$$

- Then, for $2 \le l \le L-1$, assuming we have $\hat{I}^{(h)}_{k,i,j} = \boldsymbol{r}^{(h)}_k I^{(h)}_{k,i,j}$, h = 1, 2, ..., $l-1$ by induction. Obviously, by the definition of $I^{(l,in)}$ ($\hat{I}^{(l,in)}$), we can deduce that

$$\hat{I}^{(l,in)}_{k,i,j} = \boldsymbol{r}^{(l,in)}_k I^{(l,in)}_{k,i,j}.$$

Then the noisy output of the l-th layer is given as

$$\begin{aligned} \hat{I}^{(l)}_{k,i,j} &= ReLU\left(\sum_{c',i',j'} \widehat{W}^{(l)}_{k,c',i',j'} \hat{I}^{(l,in)}_{c',i+i',j+j'} \right) \\ &= ReLU\left(\sum_{c',i',j'} R^{(l)}_{k,c',i',j'} W^{(l)}_{k,c',i',j'} \boldsymbol{r}^{(l,in)}_{c'} I^{(l,in)}_{c',i+i',j+j'} \right) \\ &= ReLU\left(\sum_{c',i',j'} \frac{\boldsymbol{r}^{(l)}_k}{\boldsymbol{r}^{(l,in)}_{c'}} \boldsymbol{r}^{(l,in)}_{c'} W^{(l)}_{k,c',i',j'} I^{(l,in)}_{c',i+i',j+j'} \right) \\ &= \boldsymbol{r}^{(l)}_k ReLU\left(\sum_{c'=1}^{c_{l,in}} \sum_{i'=1}^{f} \sum_{j'=1}^{f} W^{(l)}_{k,c',i',j'} I^{(l,in)}_{c',i+i',j+j'} \right) \\ &= \boldsymbol{r}^{(l)}_k I^{(l)}_{k,i,j}. \end{aligned}$$

Next, we prove the correctness of Eq. (18). More specifically, let a vector $\boldsymbol{y}^{(l)} \in \mathbb{R}^{c_l h_l w_l}$ be the flatten result of $I^{(l)}$, i.e., $\boldsymbol{y}^{(l)}_{kij} = I^{(l)}_{k,i,j}$. Based on Eq. (17), we have

$$\hat{\boldsymbol{y}}^{(L-1)}_{kij} = \hat{I}^{(L-1)}_{k,i,j} = \boldsymbol{r}^{(L-1)}_k I^{(L-1)}_{k,i,j} = \boldsymbol{r}^{(L-1)}_k \boldsymbol{y}^{(L-1)}_{kij}.$$

Denote $n_l = c_l h_l w_l$, then the output of last fully connected layer $\hat{\boldsymbol{y}}^{(L)}_i$, where $i = 1, 2, \ldots, n_L$, can be computed as

$$\begin{aligned} \hat{\boldsymbol{y}}^{(L)}_i &= \sum_{j=1}^{n_{L-1}} \widehat{W}^{(L)}_{ij} \hat{\boldsymbol{y}}^{(L-1)}_j \\ &= \sum_{j=1}^{n_{L-1}} \left(R^{(L)}_{ij} W^{(L)}_{ij} + R^a_{ij} \right) \boldsymbol{r}^{(L-1)}_{\lfloor j/(h_{L-1} w_{L-1}) \rfloor} \boldsymbol{y}^{(L-1)}_j \end{aligned}$$

$$
\begin{aligned}
&= \sum_{j=1}^{n_{L-1}} \left(R_{ij}^{(L)} W_{ij}^{(L)} \boldsymbol{r}_{\lfloor j/(h_{L-1}w_{L-1}) \rfloor}^{(L-1)} \boldsymbol{y}_j^{(L-1)} + R_{ij}^{a} \hat{\boldsymbol{y}}_j^{(L-1)} \right) \\
&= \sum_{j=1}^{n_{L-1}} \left(1/\boldsymbol{r}_{\lfloor j/(h_{L-1}w_{L-1}) \rfloor}^{(L-1)} W_{ij}^{(L)} \boldsymbol{r}_{\lfloor j/(h_{L-1}w_{L-1}) \rfloor}^{(L-1)} \boldsymbol{y}_j^{(L-1)} \right) \\
&\quad + \sum_{j=1}^{n_{L-1}} \left(\boldsymbol{\gamma}_i \boldsymbol{r}_{a,i} \hat{\boldsymbol{y}}_j^{(L-1)} \right) \\
&= \sum_{j=1}^{n_{L-1}} W_{ij}^{(L)} \boldsymbol{y}_j^{(L-1)} + \boldsymbol{\gamma}_i \boldsymbol{r}_{a,i} \sum_{j=1}^{n_{L-1}} \hat{\boldsymbol{y}}_j^{(L-1)} \\
&= \boldsymbol{y}_i^{(L)} + \alpha \boldsymbol{r}_i,
\end{aligned}
$$

which implies that $\hat{\boldsymbol{y}}^{(L)} = \boldsymbol{y}^{(L)} + \alpha\boldsymbol{\gamma} \circ \boldsymbol{r}_a = \boldsymbol{y}^{(L)} + \alpha\boldsymbol{r}$. □

B. Noisy Gradient Computation

Generally, the local training performed by each client includes the forward and backward propagation. Since the forward propagation is introduced in Section III-A, we just show the backward propagation here. Note that the derivations given in the following contents can be applied in both MLP and CNN. After computing local gradients, clients perturb them with the secret sharing technique to guarantee that the semi-trusted server can only recover the aggregated gradients rather than individual local gradients.

1) Backward propagation: After obtain the noisy outputs of each layer in forward propagation process, i.e., $\hat{\boldsymbol{y}}^{(l)}$ for $l = 1, 2, \ldots, L$, the client calculates the corresponding gradients based on the specific loss function. Specifically, we first use the *Mean Squared Error (MSE)* loss function as the basis and then show the case of the *Cross-Entropy (CE)* loss function.

(a) The case of MSE: Considering our encrypted model, for each sample $(\boldsymbol{x}, \bar{\boldsymbol{y}})$, from Lemma 1, the noisy MSE is

$$
\widehat{\mathcal{L}} = \frac{1}{2} \sum_{i=1}^{n_L} \left(\hat{\boldsymbol{y}}_i^{(L)} - \bar{\boldsymbol{y}}_i \right)^2 = \frac{1}{2} \sum_{i=1}^{n_L} \left(\boldsymbol{y}_i^{(L)} - \bar{\boldsymbol{y}}_i + \alpha \boldsymbol{r}_i \right)^2 .
$$

In what follows, Lemma 3 shows the relation between the noisy gradients and the true gradients.

Lemma 3. *For any $1 \leq l \leq L$, the noisy gradient matrix $\frac{\partial \widehat{\mathcal{L}}}{\partial \widehat{W}^{(l)}}$ and the true gradient matrix $\frac{\partial \mathcal{L}}{\partial W^{(l)}}$ satisfy*

$$
\frac{\partial \widehat{\mathcal{L}}}{\partial \widehat{W}^{(l)}} = \frac{1}{R^{(l)}} \circ \frac{\partial \mathcal{L}}{\partial W^{(l)}} + \boldsymbol{r}^T \boldsymbol{\sigma}^{(l)} - \upsilon \boldsymbol{\beta}^{(l)}, \tag{19}
$$

where $\boldsymbol{\sigma}^{(l)} = \left(\alpha \frac{\partial \hat{\boldsymbol{y}}^{(L)}}{\partial \widehat{W}^{(l)}} + (\frac{\partial \widehat{\mathcal{L}}}{\partial \hat{\boldsymbol{y}}^{(L)}})^T \frac{\partial \alpha}{\partial \widehat{W}^{(l)}} \right)$, $\upsilon = \boldsymbol{r}^T \boldsymbol{r}$ and $\boldsymbol{\beta}^{(l)} = \alpha \frac{\partial \alpha}{\partial \widehat{W}^{(l)}}$.

Proof. Based on the noisy MSE, we have

$$
\frac{\partial \widehat{\mathcal{L}}}{\partial \hat{\boldsymbol{y}}^{(L)}} = (\hat{\boldsymbol{y}}^{(L)} - \bar{\boldsymbol{y}})^T = \frac{\partial \mathcal{L}}{\partial \boldsymbol{y}^{(L)}} + \alpha \boldsymbol{r}^T,
$$

Thus, we can derive that

$$
\begin{aligned}
\frac{\partial \widehat{\mathcal{L}}}{\partial \widehat{W}^{(l)}} &= \frac{\partial \widehat{\mathcal{L}}}{\partial \hat{\boldsymbol{y}}^{(L)}} \frac{\partial \hat{\boldsymbol{y}}^{(L)}}{\partial \widehat{W}^{(l)}} = \left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{y}^{(L)}} + \alpha \boldsymbol{r}^T \right) \frac{\partial \hat{\boldsymbol{y}}^{(L)}}{\partial \widehat{W}^{(l)}} \\
&= \frac{\partial \mathcal{L}}{\partial \boldsymbol{y}^{(L)}} \left(\frac{\partial \boldsymbol{y}^{(L)}}{\partial \widehat{W}^{(l)}} + \frac{\partial (\alpha \boldsymbol{r})}{\partial \widehat{W}^{(l)}} \right) + \alpha \boldsymbol{r}^T \frac{\partial \hat{\boldsymbol{y}}^{(L)}}{\partial \widehat{W}^{(l)}} \\
&= \frac{\partial \mathcal{L}}{\partial \widehat{W}^{(l)}} + \left(\frac{\partial \widehat{\mathcal{L}}}{\partial \hat{\boldsymbol{y}}^{(L)}} - \alpha \boldsymbol{r}^T \right) \frac{\partial (\alpha \boldsymbol{r})}{\partial \widehat{W}^{(l)}} + \alpha \boldsymbol{r}^T \frac{\partial \hat{\boldsymbol{y}}^{(L)}}{\partial \widehat{W}^{(l)}} \\
&= \frac{\partial \mathcal{L}}{\partial \widehat{W}^{(l)}} + \boldsymbol{r}^T \left(\alpha \frac{\partial \hat{\boldsymbol{y}}^{(L)}}{\partial \widehat{W}^{(l)}} + \left(\frac{\partial \widehat{\mathcal{L}}}{\partial \hat{\boldsymbol{y}}^{(L)}} \right)^T \frac{\partial \alpha}{\partial \widehat{W}^{(l)}} \right) \\
&\quad - \upsilon \alpha \frac{\partial \alpha}{\partial \widehat{W}^{(l)}} \\
&= \frac{1}{R^{(l)}} \circ \frac{\partial \mathcal{L}}{\partial W^{(l)}} + \boldsymbol{r}^T \boldsymbol{\sigma}^{(l)} - \upsilon \boldsymbol{\beta}^{(l)}.
\end{aligned}
$$

□

Note that $\boldsymbol{\sigma}^{(l)}$ and $\boldsymbol{\beta}^{(l)}$ can be computed directly by the clients. For all samples in a mini-batch dataset $\mathcal{D}_k^t \in \mathcal{D}_k$, the k-th client computes the average gradients and the noisy items:

$$
\begin{cases}
\nabla \widehat{W}^{(l)} = \dfrac{1}{|\mathcal{D}_k^t|} \displaystyle\sum_{n \in \mathcal{D}_k^t} \dfrac{\partial \widehat{\mathcal{L}}_n}{\partial \widehat{W}^{(l)}}; \\
\boldsymbol{\sigma}^{(l)} = \dfrac{1}{|\mathcal{D}_k^t|} \displaystyle\sum_{n \in \mathcal{D}_k^t} \boldsymbol{\sigma}_n^{(l)}; \\
\boldsymbol{\beta}^{(l)} = \dfrac{1}{|\mathcal{D}_k^t|} \displaystyle\sum_{n \in \mathcal{D}_k^t} \boldsymbol{\beta}_n^{(l)}.
\end{cases}
$$

(b) The case of CE: Since the CE loss is widely adopted for classification tasks, we discuss how to incorporate CE loss into our method with one additional round of interaction between the server and clients. Similarly, for each sample $(\boldsymbol{x}, \bar{\boldsymbol{y}})$, from Lemma 1, the form of noisy CE is

$$
\widehat{\mathcal{L}} = -\sum_{i=1}^{n_L} \bar{\boldsymbol{y}}_i \log \frac{\exp(\hat{\boldsymbol{y}}_i^{(L)})}{\sum_{j=1}^{n_L} \exp(\hat{\boldsymbol{y}}_j^{(L)})}. \tag{20}
$$

Lemma 4 shows the gradient of CE loss w.r.t the final output of models.

Lemma 4. *The gradient of cross entropy loss $\mathcal{L} = -\sum_{i=1}^{n_L} \bar{\boldsymbol{y}}_i \log \frac{\exp(\boldsymbol{y}_i^{(L)})}{\sum_{j=1}^{n_L} \exp(\boldsymbol{y}_j^{(L)})}$ w.r.t the final output of the model $\boldsymbol{y}^{(L)}$ can be derived as*

$$
\frac{\partial \mathcal{L}}{\partial \boldsymbol{y}_i^{(L)}} = \frac{\exp(\boldsymbol{y}_i^{(L)})}{\sum_{j=1}^{n_L} \exp(\boldsymbol{y}_j^{(L)})} - \bar{\boldsymbol{y}}_i. \tag{21}
$$

Proof. The cross-entropy loss $\mathcal{L}$ can be deduced as

$$
\begin{aligned}
\mathcal{L} &= -\sum_{i=1}^{n_L} \bar{\boldsymbol{y}}_i \log \frac{\exp(\boldsymbol{y}_i^{(L)})}{\sum_{j=1}^{n_L} \exp(\boldsymbol{y}_j^{(L)})} \\
&= (\sum_{i=1}^{n_L} \bar{\boldsymbol{y}}_i) \log(\sum_{j=1}^{n_L} \exp(\boldsymbol{y}_j^{(L)})) - \sum_{i=1}^{n_L} \bar{\boldsymbol{y}}_i \boldsymbol{y}_i^{(L)} \\
&= \log(\sum_{j=1}^{n_L} \exp(\boldsymbol{y}_j^{(L)})) - \sum_{i=1}^{n_L} \bar{\boldsymbol{y}}_i \boldsymbol{y}_i^{(L)}.
\end{aligned}
$$

Thus, we have

$$
\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \boldsymbol{y}_i^{(L)}} &= \frac{\partial \log(\sum_{j=1}^{n_L} \exp(\boldsymbol{y}_j^{(L)}))}{\partial \boldsymbol{y}_i^{(L)}} - \bar{\boldsymbol{y}}_i \\
&= \frac{\exp(\boldsymbol{y}_i^{(L)})}{\sum_{j=1}^{n_L} \exp(\boldsymbol{y}_j^{(L)})} - \bar{\boldsymbol{y}}_i.
\end{aligned}
$$

□

From Lemma 4, the noisy gradient of the noisy loss function Eq. (20) is computed as:

$$\begin{aligned}\frac{\partial\widehat{\mathcal{L}}}{\partial\hat{\boldsymbol{y}}_i^{(L)}} &= \frac{\exp(\hat{\boldsymbol{y}}_i^{(L)})}{\sum_{j=1}^{n_L}\exp(\hat{\boldsymbol{y}}_j^{(L)})}-\bar{\boldsymbol{y}}_i \\ &= \frac{\exp(\boldsymbol{y}_i^{(L)}+\alpha\boldsymbol{r}_i)}{\sum_{j=1}^{n_L}\exp(\boldsymbol{y}_j^{(L)}+\alpha\boldsymbol{r}_j)}-\bar{\boldsymbol{y}}_i. \qquad (22)\end{aligned}$$

Due to the complexity of Eq. (22), if the client directly returns the noisy gradients, the server cannot extract the true gradients even though it knows the private key. Thus, in order to help the server to obtain the true gradients, an interactive protocol between the client and the server needs to be conducted before performing the process of the backward propagation. More specifically, the corresponding interactive protocol is introduced as follows:

(1) **Client → Server:** For each class i, the client selects a random number $\boldsymbol{\lambda}_i$ and computes $\boldsymbol{\mu}_{ij}=\frac{\exp(\boldsymbol{y}_j^{(L)}+\boldsymbol{r}_j\alpha)}{\exp(\boldsymbol{y}_i^{(L)}+\boldsymbol{r}_i\alpha)}+\boldsymbol{\lambda}_i$ for $j=1,2,\ldots,n_L$ and $j\neq i$. Then, the client sends $\{\boldsymbol{\mu}_{ij}\}_{j=1,j\neq i}^{n_L}$ and α to the server.

(2) **Server → Client:** After receiving $\{\boldsymbol{\mu}_{ij}\}_{j=1,j\neq i}^{n_L}$, the server first computes:

$$\begin{aligned}&\boldsymbol{\mu}_{ij}\cdot\exp(\boldsymbol{r}_{ij}^*) \\ =& \left(\frac{\exp\left(\boldsymbol{y}_j^{(L)}+\boldsymbol{r}_j\alpha\right)}{\exp\left(\boldsymbol{y}_i^{(L)}+\boldsymbol{r}_i\alpha\right)}+\boldsymbol{\lambda}_i\right)\exp(\boldsymbol{r}_{ij}^*) \\ =& \frac{\exp\left(\boldsymbol{y}_j^{(L)}\right)}{\exp\left(\boldsymbol{y}_i^{(L)}\right)}\cdot\exp((\boldsymbol{r}_j-\boldsymbol{r}_i)\alpha+\boldsymbol{r}_{ij}^*)+\lambda\exp(\boldsymbol{r}_{ij}^*) \\ =& \frac{\exp(\boldsymbol{y}_j^{(L)})}{\exp(\boldsymbol{y}_i^{(L)})}\cdot\exp(\boldsymbol{\delta}_i)+\boldsymbol{\lambda}_i\exp(\boldsymbol{r}_{ij}^*), \qquad (23)\end{aligned}$$

where $j=1,2,\ldots,n_L$ and $j\neq i$. Given $\boldsymbol{\delta}_i$ randomly selected by the server, the random number $\boldsymbol{r}_{ij}^*$ should satisfy $(\boldsymbol{r}_j-\boldsymbol{r}_i)\alpha+\boldsymbol{r}_{ij}^*=\boldsymbol{\delta}_i$ for $j=1,2,\ldots,n_L$ and $j\neq i$. Then, the server computes:

$$\begin{aligned}\hat{\boldsymbol{y}}_i' &= \exp(\boldsymbol{\delta}_i)+\sum_{j=1,j\neq i}^{n_L}\boldsymbol{\mu}_{ij}\cdot\exp(\boldsymbol{r}_{ij}^*) \\ &= \exp(\boldsymbol{\delta}_i)+\sum_{j=1,j\neq i}^{n_L}\Big(\frac{\exp(\boldsymbol{y}_j^{(L)})}{\exp(\boldsymbol{y}_i^{(L)})}\cdot\exp(\boldsymbol{\delta}_i)+ \\ &\quad \boldsymbol{\lambda}_i\exp(\boldsymbol{r}_{ij}^*)\Big) \\ &= \exp(\boldsymbol{\delta}_i)\left(1+\sum_{j=1,j\neq i}^{n_L}\frac{\exp(\boldsymbol{y}_j^{(L)})}{\exp(\boldsymbol{y}_i^{(L)})}\right)+ \\ &\quad \boldsymbol{\lambda}_i\sum_{j=1,j\neq i}^{n_L}\exp(\boldsymbol{r}_{ij}^*) \\ &= \exp(\boldsymbol{\delta}_i)\frac{\sum_{j=1}^{n_L}\exp(\boldsymbol{y}_j^{(L)})}{\exp(\boldsymbol{y}_i^{(L)})}+\boldsymbol{\lambda}_i\sum_{j=1,j\neq i}^{n_L}\exp(\boldsymbol{r}_{ij}^*).\end{aligned}$$

Finally, the server generates a random vector $\boldsymbol{\xi}\in\mathbb{R}^{n_L}$, which satisfies that for any i,j in the same I_s, we have $\boldsymbol{\xi}_i=\boldsymbol{\xi}_j$ (Recall that $\sqcup_{s=1}^m\{I_s\}$ is a partition of $\{1,2,\ldots,n_L\}$, and we denote $\boldsymbol{\xi}_{I_s}=\xi_i$ for $i\in I_s$). Then the server returns $\hat{\boldsymbol{y}}_i'$, $\frac{\boldsymbol{\xi}_i}{1-\exp(\boldsymbol{\delta}_i)}$ together with $\sum_{j=1,j\neq i}^{n_L}\exp(\boldsymbol{r}_{ij}^*)$ to the client.

After receiving $\hat{\boldsymbol{y}}_i'$, $\frac{1-\exp(\boldsymbol{\delta}_i)}{\boldsymbol{\xi}_i}$ and $\sum_{j=1,j\neq i}^{n_L}\exp(\boldsymbol{r}_{ij}^*)$, the client uses the random number $\boldsymbol{\lambda}_i$ to compute

$$\begin{aligned}\hat{\boldsymbol{y}}_i^* &= \frac{1}{\hat{\boldsymbol{y}}_i'-\boldsymbol{\lambda}_i\sum_{j=1,j\neq i}^{n_L}\exp(\boldsymbol{r}_{ij}^*)}=\frac{1}{\exp(\boldsymbol{\delta}_i)\frac{\sum_{j=1}^{n_L}\exp(\boldsymbol{y}_j^{(L)})}{\exp(\boldsymbol{y}_i^{(L)})}} \\ &= \frac{1}{\exp(\boldsymbol{\delta}_i)}\cdot\frac{\exp(\boldsymbol{y}_i^{(L)})}{\sum_{j=1}^{n_L}\exp(y_j^{(L)})}. \qquad (24)\end{aligned}$$

Let $\exp(\boldsymbol{\delta})=(\exp(\boldsymbol{\delta}_1),\exp(\boldsymbol{\delta}_2),...,\exp(\boldsymbol{\delta}_{n_L}))$. Then the client reconstruct the noisy gradient as

$$\nabla\widehat{W}^{(l)}=(\hat{\boldsymbol{y}}^*-\bar{\boldsymbol{y}})^T\frac{\partial\hat{\boldsymbol{y}}^{(L)}}{\partial\widehat{W}^{(l)}}. \qquad (25)$$

Then, Lemma 5 shows the relationship between the reconstructed noisy gradients and true gradients.

Lemma 5. *For any $1\leq l\leq L$, the noisy gradient matrix $\nabla\widehat{W}^{(l)}$ and the true gradient matrix $\frac{\partial\mathcal{L}}{\partial W^{(l)}}$ satisfy*

$$\nabla\widehat{W}^{(l)}=\frac{1}{R^{(l)}}\circ\frac{\partial\mathcal{L}}{\partial W^{(l)}}+\boldsymbol{r}^T\boldsymbol{\sigma}^{(l)}-(\boldsymbol{r}^T\circ\boldsymbol{\xi}^T)\boldsymbol{\beta}^{(l)}+\boldsymbol{\xi}^T\psi^{(l)}, \qquad (26)$$

where $\boldsymbol{\sigma}^{(l)}=\left((\hat{\boldsymbol{y}}^*-\bar{\boldsymbol{y}})\frac{\partial\alpha}{\partial\widehat{W}^{(l)}}\right)$, $\boldsymbol{\beta}^{(l)}=\widetilde{\boldsymbol{y}}^*\frac{\partial\alpha}{\partial\widehat{W}^{(l)}}$,$\boldsymbol{\psi}^{(l)}=D_{\widetilde{\boldsymbol{y}}^{*T}}\frac{\partial\hat{\boldsymbol{y}}^{(L)}}{\partial\widehat{W}^{(l)}}$, and $\widetilde{\boldsymbol{y}}^*:=\hat{\boldsymbol{y}}^*\circ(\frac{1-\exp(\boldsymbol{\delta})}{\boldsymbol{\xi}})$.

Proof. From Eq.(24) and Lemma 4, we have

$$\begin{aligned}\hat{\boldsymbol{y}}^*-\bar{\boldsymbol{y}} &= \frac{1}{\exp(\boldsymbol{\delta})}\circ\frac{\exp(\boldsymbol{y}^{(L)})}{\sum_{j=1}^{n_L}\exp(y_j^{(L)})}-\bar{\boldsymbol{y}} \\ &= \left(\frac{\exp(\boldsymbol{y}^{(L)})}{\sum_{j=1}^{n_L}\exp(y_j^{(L)})}-\bar{\boldsymbol{y}}\right)+ \\ &\quad \left(\frac{1}{\exp(\boldsymbol{\delta})}-1\right)\circ\frac{\exp\left(\boldsymbol{y}^{(L)}\right)}{\sum_{j=1}^{n_L}\exp\left(y_j^{(L)}\right)} \\ &= \left(\frac{\partial\mathcal{L}}{\partial\boldsymbol{y}^{(L)}}\right)^T+\boldsymbol{\xi}\circ\widetilde{\boldsymbol{y}}^*.\end{aligned}$$

Then, we can obtain

$$\begin{aligned}\nabla\widehat{W}^{(l)} &= (\hat{\boldsymbol{y}}^*-\bar{\boldsymbol{y}})^T\frac{\partial\hat{\boldsymbol{y}}^{(L)}}{\partial\widehat{W}^{(l)}} \\ &= \left((\frac{\partial\mathcal{L}}{\partial\boldsymbol{y}^{(L)}})^T+\boldsymbol{\xi}\circ\widetilde{\boldsymbol{y}}^*\right)^T\left(\frac{\partial\boldsymbol{y}^{(L)}}{\partial\widehat{W}^{(l)}}+\frac{\partial(\alpha\boldsymbol{r})}{\partial\widehat{W}^{(l)}}\right) \\ &= \frac{1}{R^{(l)}}\circ\frac{\partial\mathcal{L}}{\partial W^{(l)}}+\boldsymbol{r}^T\left((\hat{\boldsymbol{y}}^*-\bar{\boldsymbol{y}})\frac{\partial\alpha}{\partial\widehat{W}^{(l)}}\right) \\ &\quad -(\boldsymbol{r}^T\circ\boldsymbol{\xi}^T)\left(\widetilde{\boldsymbol{y}}^*\frac{\partial\alpha}{\partial\widehat{W}^{(l)}}\right)+\boldsymbol{\xi}^TD_{\widetilde{\boldsymbol{y}}^*}\frac{\partial\hat{\boldsymbol{y}}^{(L)}}{\partial\widehat{W}^{(l)}}\end{aligned}$$

$$= \quad \frac{1}{R^{(l)}} \circ \frac{\partial \mathcal{L}}{\partial W^{(l)}} + \boldsymbol{r}^T \boldsymbol{\sigma}^{(l)} - (\boldsymbol{r}^T \circ \boldsymbol{\xi}^T)\boldsymbol{\beta}^{(l)} + \boldsymbol{\xi}^T \boldsymbol{\psi}^{(l)},$$

where the third equality from the fact that $(\boldsymbol{\xi}^T \circ \widetilde{\boldsymbol{y}}^{*T})\boldsymbol{r} = \boldsymbol{r}^T(\boldsymbol{\xi} \circ \widetilde{\boldsymbol{y}}^*) = \boldsymbol{r}^T D_{\boldsymbol{\xi}} \widetilde{\boldsymbol{y}}^* = (\boldsymbol{r}^T \circ \boldsymbol{\xi}^T)\widetilde{\boldsymbol{y}}^*$ and $\boldsymbol{\xi}^T \circ \widetilde{\boldsymbol{y}}^{*T} = \boldsymbol{\xi}^T D_{\widetilde{\boldsymbol{y}}^{*T}}$. □

Similarly, $\boldsymbol{\sigma}^{(l)}$, $\boldsymbol{\beta}^{(l)}$ and $\boldsymbol{\psi}^{(l)}$ can be computed directly by the clients. For all samples in a mini-batch dataset $\mathcal{D}_k^t \in \mathcal{D}_k$, the k-th client computes the average gradients and the noisy items:

$$\begin{cases} \nabla\widehat{W}^{(l)} = \dfrac{1}{|\mathcal{D}_k^t|} \displaystyle\sum_{n \in \mathcal{D}_k^t} \nabla\widehat{W}_n^{(l)}; \\ \boldsymbol{\sigma}^{(l)} = \dfrac{1}{|\mathcal{D}_k^t|} \displaystyle\sum_{n \in \mathcal{D}_k^t} \boldsymbol{\sigma}_n^{(l)}; \\ \boldsymbol{\beta}^{(l)} = \dfrac{1}{|\mathcal{D}_k^t|} \displaystyle\sum_{n \in \mathcal{D}_k^t} \boldsymbol{\beta}_n^{(l)}; \\ \boldsymbol{\psi}^{(l)} = \dfrac{1}{|\mathcal{D}_k^t|} \displaystyle\sum_{n \in \mathcal{D}_k^t} \boldsymbol{\psi}_n^{(l)}. \end{cases}$$

2) Local gradients perturbation: First, all K clients consult with a set of random number matrices $\left\{\boldsymbol{\phi}_1^{(l)}, \boldsymbol{\phi}_2^{(l)}, \ldots, \boldsymbol{\phi}_K^{(l)}\right\}$ such that $\sum_{k=1}^K N_k \boldsymbol{\phi}_k^{(l)} = \mathbf{0}_{n_l \times n_{l-1}}$ for $l = 1, 2, \ldots, L$, which are unknown to the server. Without loss of generality, we assume the k-th client holds $\{\boldsymbol{\phi}_k^{(l)}\}_{l=1}^L$. Note that for different iteration, clients can consult with a different set of random matrices. After each client k computes the noisy local gradients $\left\{\left(\nabla\widehat{W}^{(l)}\right)\right\}_{l=1}^L$ as well as noisy terms ($\{\boldsymbol{\sigma}_k^{(l)}, \boldsymbol{\beta}_k^{(l)}\}_{l=1}^L$ for MSE and $\{\boldsymbol{\sigma}_k^{(l)}, \boldsymbol{\beta}_k^{(l)}, \boldsymbol{\psi}_k^{(l)}\}_{l=1}^L$ for CE), the k-th client masks them with the secret parameter $\boldsymbol{\phi}_k^{(l)}$ as:

$$\begin{cases} \nabla\widehat{F}_k(\widehat{W}^{(l)}) = \left(\nabla\widehat{W}^{(l)}\right)_k + \boldsymbol{\phi}_k^{(l)}, \\ \widehat{\boldsymbol{\sigma}}_k^{(l)} = \boldsymbol{\sigma}_k^{(l)} + \boldsymbol{\phi}_k^{(l)}, \\ \widehat{\boldsymbol{\beta}}_k^{(l)} = \boldsymbol{\beta}_k^{(l)} + \boldsymbol{\phi}_k^{(l)}, \\ \widehat{\boldsymbol{\psi}}_k^{(l)} = \boldsymbol{\psi}_k^{(l)} + \boldsymbol{\phi}_k^{(l)} \ (for\ the\ case\ of\ CE). \end{cases} \tag{27}$$

where $\boldsymbol{\phi}_k^{(l)}$ used for computing $\nabla\widehat{F}_k(\widehat{W}^{(l)})$, $\widehat{\boldsymbol{\sigma}}_k^{(l)}$, $\widehat{\boldsymbol{\beta}}_k^{(l)}$ and $\widehat{\boldsymbol{\psi}}_k^{(l)}$ can be different, for simplicity, we do not use other symbols to distinguish.

Finally, the k-th client returns the perturbed gradients together with the additional noisy terms to the server:

- For MSE, the k-th client returns $\{\nabla\widehat{F}_k(\widehat{W}^{(l)})\}_{l=1}^L$ and $(m+1)$ noisy terms $\{\widetilde{\boldsymbol{\sigma}}_{k,1}^{(l)}, \widetilde{\boldsymbol{\sigma}}_{k,2}^{(l)}, \cdots, \widetilde{\boldsymbol{\sigma}}_{k,m}^{(l)}, \widehat{\boldsymbol{\beta}}_k^{(l)}\}_{l=1}^L$ to the server, where $\widetilde{\boldsymbol{\sigma}}_{k,s}^{(l)} = \boldsymbol{r}_{a,I_s}^T \widehat{\boldsymbol{\sigma}}_{k_{|I_s}}^{(l)}$ for $s = 1, 2, \ldots, m$.
- For CE, the k-th client returns $\{\nabla\widehat{F}_k(\widehat{W}^{(l)})\}_{l=1}^L$ and $3m$ noisy terms $\{\widetilde{\boldsymbol{\sigma}}_{k,1}^{(l)}, \widetilde{\boldsymbol{\sigma}}_{k,2}^{(l)}, \cdots, \widetilde{\boldsymbol{\sigma}}_{k,m}^{(l)}, \widetilde{\boldsymbol{\beta}}_{k,1}^{(l)}, \widetilde{\boldsymbol{\beta}}_{k,2}^{(l)}, \cdots, \widetilde{\boldsymbol{\beta}}_{k,m}^{(l)}, \widetilde{\boldsymbol{\psi}}_{k,1}^{(l)}, \widetilde{\boldsymbol{\psi}}_{k,2}^{(l)}, \cdots, \widetilde{\boldsymbol{\psi}}_{k,m}^{(l)}\}_{l=1}^L$ to the server, where $\widetilde{\boldsymbol{\beta}}_{k,s}^{(l)} = \boldsymbol{r}_{a,I_s}^T \widehat{\boldsymbol{\beta}}_{k_{|I_s}}^{(l)}$ and $\widetilde{\boldsymbol{\psi}}_{k,s}^{(l)} = \mathbf{1}_{I_s}^T \widehat{\boldsymbol{\psi}}_{k_{|I_s}}^{(l)}$ for $s = 1, 2, \ldots, m$.

Note that $\mathbf{1}_{I_s}$ is denoted as the all 1's vector of length $|I_s|$. For the additive noisy vector $\boldsymbol{r}_a$, we denote $\boldsymbol{r}_{a,I_s}$ the restriction of $\boldsymbol{r}_a$ on I_s. For any $c \times n$ matrix M, we denote $M_{|I_s}$ the sub-matrix of M consists of the rows indexed by I_s.

C. Model Update

After receiving the noisy local gradients from all clients, the server first needs to aggregate received data from all clients, and then recover the exact model updates for the next iteration (i.e., $(t+1)$-th iteration). Specifically, the server performs the following operations.

1) The case of MSE: The server aggregates the received messages as follows:

$$\begin{cases} \nabla\widehat{F}(\widehat{W}^{(l)}) = \displaystyle\sum_{k=1}^K \frac{N_k}{N} \nabla\widehat{F}_k(\widehat{W}^{(l)}), \\ \widetilde{\boldsymbol{\sigma}}_s^{(l)} = \displaystyle\sum_{k=1}^K \frac{N_k}{N} \widetilde{\boldsymbol{\sigma}}_{k,s}^{(l)}, s = 1, 2, \ldots, m, \\ \widehat{\boldsymbol{\beta}}^{(l)} = \displaystyle\sum_{k=1}^K \frac{N_k}{N} \widehat{\boldsymbol{\beta}}_k^{(l)}. \end{cases} \tag{28}$$

where $l = 1, 2, \ldots, L$. Then, according to Lemma 3, the server recovers the true aggregated gradients $\nabla F(W^{(l)})$ as follows: for $l = 1, 2, \ldots, L$,

$$\begin{aligned} \nabla F(W^{(l)}) &= R^{(l)} \circ \left(\nabla\widehat{F}(\widehat{W}^{(l)}) - (\sum_{s=1}^m \boldsymbol{\gamma}_{I_s} \widetilde{\boldsymbol{\sigma}}_s^{(l)}) + \upsilon\widehat{\boldsymbol{\beta}}^{(l)}\right) \\ &= R^{(l)} \circ \left(\nabla\widehat{F}(\widehat{W}^{(l)}) - \boldsymbol{r}^T \widehat{\boldsymbol{\sigma}}^{(l)} + \upsilon\widehat{\boldsymbol{\beta}}^{(l)}\right) \\ &= \sum_{k=1}^K \frac{N_k}{N} \frac{\partial \widehat{\mathcal{L}}}{\partial \widehat{W}^{(l)}} \end{aligned} \tag{29}$$

where $\widehat{\boldsymbol{\sigma}}^{(l)} = \sum_{k=1}^K \frac{N_k}{N} \widehat{\boldsymbol{\sigma}}_k^{(l)}$.

In what follows, we prove the correctness of Eq. (29).

Proof. Before proving the correctness of Eq. (29), we first give results of Eq. (28) in details. Specifically, for $l = 1, 2, \ldots, L$,

$$\begin{aligned} \nabla\widehat{F}(\widehat{W}^{(l)}) &= \sum_{k=1}^K \frac{N_k}{N} \nabla\widehat{F}_k(\widehat{W}^{(l)}) \\ &= \sum_{k=1}^K \frac{N_k}{N} \frac{\partial \widehat{\mathcal{L}}}{\partial \widehat{W}^{(l)}} + \mathbf{0}_{n_l \times n_{l-1}} \\ &= \sum_{k=1}^K \frac{N_k}{N} \frac{\partial \widehat{\mathcal{L}}}{\partial \widehat{W}^{(l)}}. \end{aligned}$$

$$\begin{aligned} \widehat{\boldsymbol{\sigma}}^{(l)} &= \sum_{k=1}^K \frac{N_k}{N} \widehat{\boldsymbol{\sigma}}_k^{(l)} = \sum_{k=1}^K \frac{N_k}{N} \left(\boldsymbol{\sigma}_k^{(l)} + \boldsymbol{\phi}_k^{(l)}\right) \\ &= \sum_{k=1}^K \frac{N_k}{N} \boldsymbol{\sigma}_k^{(l)} + \mathbf{0}_{n_l \times n_{l-1}} = \sum_{k=1}^K \frac{N_k}{N} \boldsymbol{\sigma}_k^{(l)}. \end{aligned}$$

$$\begin{aligned} \widehat{\boldsymbol{\beta}}^{(l)} &= \sum_{k=1}^K \frac{N_k}{N} \widehat{\boldsymbol{\beta}}_k^{(l)} = \sum_{k=1}^K \frac{N_k}{N} \left(\boldsymbol{\beta}_k^{(l)} + \boldsymbol{\phi}_k^{(l)}\right) \\ &= \sum_{k=1}^K \frac{N_k}{N} \boldsymbol{\beta}_k^{(l)} + \mathbf{0}_{n_l \times n_{l-1}} = \sum_{k=1}^K \frac{N_k}{N} \boldsymbol{\beta}_k^{(l)}. \end{aligned}$$

Consequently, we can derive that

$$\begin{aligned}\nabla F(W^{(l)}) &= R^{(l)} \circ \Big(\nabla \widehat{F}(\widehat{W}^{(l)}) - \boldsymbol{r}^T \widehat{\boldsymbol{\sigma}}^{(l)} + \upsilon \widehat{\boldsymbol{\beta}}^{(l)}\Big) \\ &= R^{(l)} \circ \sum_{k=1}^{K} \frac{N_k}{N} \left(\frac{\partial \widehat{\mathcal{L}}}{\partial \widehat{W}^{(l)}} - \boldsymbol{r}^T \boldsymbol{\sigma}_k^{(l)} + \upsilon \boldsymbol{\beta}_k^{(l)} \right) \\ &= R^{(l)} \circ \sum_{k=1}^{K} \frac{N_k}{N} \left(\frac{1}{R^{(l)}} \circ \frac{\partial \mathcal{L}}{\partial W^{(l)}} + \boldsymbol{r}^T \boldsymbol{\sigma}_k^{(l)} - \right) \\ &\quad \upsilon \boldsymbol{\beta}_k^{(l)} - \boldsymbol{r}^T \boldsymbol{\sigma}_k^{(l)} + \upsilon \boldsymbol{\beta}_k^{(l)} \Big) \\ &= \sum_{k=1}^{K} \frac{N_k}{N} \frac{\partial \mathcal{L}}{\partial W^{(l)}}.\end{aligned}$$

Note that the above equation holds when the multiplicative noisy vectors $\{\boldsymbol{r}^{(l)}\}_{l=1}^{L}$, the additive noisy vector $\boldsymbol{r}_a$ and a random coefficient $\boldsymbol{\gamma}$ for all clients are the same. In other words, the noisy parameters $\{\widehat{W}^{(l)}\}_{l=1}^{L}$ sent to all clients are the same. □

2) The case of CE: Similarly, the server computes

$$\begin{cases} \nabla \widehat{F}(\widehat{W}^{(l)}) = \sum_{k=1}^{K} \frac{N_k}{N} \nabla \widehat{F}_k(\widehat{W}^{(l)}), \\ \widetilde{\boldsymbol{\sigma}}_s^{(l)} = \sum_{k=1}^{K} \frac{N_k}{N} \widetilde{\boldsymbol{\sigma}}_{k,s}^{(l)}, s = 1, 2, \ldots, m, \\ \widetilde{\boldsymbol{\beta}}_s^{(l)} = \sum_{k=1}^{K} \frac{N_k}{N} \widetilde{\boldsymbol{\beta}}_{k,s}^{(l)}, \\ \widetilde{\boldsymbol{\psi}}_s^{(l)} = \sum_{k=1}^{K} \frac{N_k}{N} \widetilde{\boldsymbol{\psi}}_{k,s}^{(l)}, \end{cases}$$

where $l = 1, 2, \ldots, L$. Then, according to Lemma 5, the server recovers the true aggregated gradients $\nabla F(W^{(l)})$ as follows: for $l = 1, 2, \ldots, L$,

$$\begin{aligned}\nabla F(W^{(l)}) &= R^{(l)} \circ \Big(\nabla \widehat{F}(\widehat{W}^{(l)}) - (\sum_{s=1}^{m} \boldsymbol{\gamma}_{I_s} \widetilde{\boldsymbol{\sigma}}_s^{(l)}) + \\ &\quad (\sum_{s=1}^{m} \boldsymbol{\gamma}_{I_s} \boldsymbol{\xi}_{I_s} \widetilde{\boldsymbol{\beta}}_s^{(l)}) - (\sum_{s=1}^{m} \boldsymbol{\xi}_{I_s} \widetilde{\boldsymbol{\psi}}_s^{(l)})\Big) \\ &= R^{(l)} \circ \Big(\nabla \widehat{F}(\widehat{W}^{(l)}) - \boldsymbol{r}^T \widehat{\boldsymbol{\sigma}}^{(l)} + \\ &\quad (\boldsymbol{r}^T \circ \boldsymbol{\xi}^T) \widehat{\boldsymbol{\beta}}^{(l)} - \boldsymbol{\xi}^T \widehat{\boldsymbol{\psi}}^{(l)}\Big) \\ &= \sum_{k=1}^{K} \frac{N_k}{N} \frac{\partial \mathcal{L}}{\partial W^{(l)}},\end{aligned}$$

where $\widehat{\boldsymbol{\sigma}}^{(l)} = \sum_{k=1}^{K} \frac{N_k}{N} \widehat{\boldsymbol{\sigma}}_k^{(l)}, \widehat{\boldsymbol{\beta}}^{(l)} = \sum_{k=1}^{K} \frac{N_k}{N} \widehat{\boldsymbol{\beta}}_k^{(l)}$, and $\widehat{\boldsymbol{\psi}}^{(l)} = \sum_{k=1}^{K} \frac{N_k}{N} \widehat{\boldsymbol{\psi}}_k^{(l)}$.

Finally, based on Eq. (3), the server updates the global model for the $(t+1)$-th iteration as: for $l = 1, 2, \ldots, L$,

$$W_{t+1}^{(l)} \leftarrow W^{(l)} - \eta \nabla F(W^{(l)}).$$

D. Data Inference

After finishing the model training, the server needs to send the well-trained model (i.e., model parameters $W = \{W^{(l)}\}_{l=1}^{L}$) to each client. In order to prevent clients from knowing the real model W and allow each client to predict locally, the server still needs to encrypt W. Specifically, the operations are similar to that in Section III-A, the only difference is that in the last layer, the server does not adopt the additive noises $\boldsymbol{r}_a$ and $\boldsymbol{\gamma}$. Thus, the noisy model parameters are computed as

$$\widehat{W}^{(l)} = R^{(l)} \circ W^{(l)}, \text{ for } 1 \le l \le L,$$

where $R^{(l)}$ satisfies that

- For the MLP:

  $$R_{ij}^{(l)} = \begin{cases} \boldsymbol{r}_i^{(1)}, & \text{for } l = 1; \\ \boldsymbol{r}_i^{(l)} / \boldsymbol{r}_j^{(l-1)}, & \text{for } 2 \le l \le L-1; \\ 1/\boldsymbol{r}_j^{(L-1)}, & \text{for } l = L; \end{cases}$$

  where $i \in [1, n_l]$ and $j \in [1, n_{l-1}]$.
- For the CNN:

  $$R_{k,c',i',j'}^{(l)} = \begin{cases} \boldsymbol{r}_k^{(l)}, \text{ for } l = 1, \\ \frac{\boldsymbol{r}_k^{(l)}}{\boldsymbol{r}_{c'}^{(l,in)}}, \text{ for } 2 \le l \le L-1. \end{cases}$$

  where $k \in [1, c_l]$, $c' \in [1, c_{l-1}]$, $i' \in [1, f]$ and $j' \in [1, f]$.

  $$R_{ij}^{(L)} = 1/\boldsymbol{r}_{\lfloor j/(h_{L-1} w_{L-1}) \rfloor}^{(L-1)},$$

  where $i \in [1, n_L]$ and $j \in [1, n_{L-1}]$.

Obviously, without the influence of additive noises $\boldsymbol{r}_a$ and γ, based on Lemmas 1 and 2, it can be verified that the final prediction is

$$\hat{\boldsymbol{y}}^{(L)} = \boldsymbol{y}^{(L)},$$

which is the true prediction.

IV. Security Analysis

Based on design goals, we analyze the security properties of our scheme in this section. Particularly, our analysis includes two aspects: the privacy of server-side and the privacy of client-side.

A. The Privacy of Server-side

As stated in [41], [25], the adversary can infer some information from the true gradient, and thus the intuitive idea is to prevent the attacker from obtaining the true gradient. Hence, we first prove that our method is semantic security against semi-trusted clients. Then, we show that the prediction can be protected from leaking to clients during model training.

1) Confidentiality of model parameters: As introduced in Section III-A, in order to prevent the semi-trusted clients from obtaining the true gradient, the server encrypts the global model before distributing. After getting the encrypted global model $\widehat{W} = \{\widehat{W^{(l)}}\}_{l=1}^{L}$, clients perform forward propagation and backward propagation to compute local noisy gradients. We can observe that clients perform linear and derivative operations based on the encrypted global model, which would not affect the security of the gradient, and thus the crux is the security of $\widehat{W} = \{\widehat{W^{(l)}}\}_{l=1}^{L}$. Next, we show that our method is semantically secure. Particularly, the server randomly selects different private keys for different iterations, i.e., the private key is one-time used, and thus we first would like to review the definition of semantic security for a one-time key [35], [5].

Definition 2. *For a cipher $\mathbb{E} = (E, D)$, where E and D are encryption and decryption operations, respectively. Consider an adversary $\mathcal{A}$ that selects two messages m_0 and m_1 with the same length from the message space. The challenger then flips a fair binary coin b, encrypts one of the messages $E(k, m_b)$ using a random k selected from the key space and sends it back to $\mathcal{A}$. $\mathcal{A}$ now guesses $b^* \in \{0,1\}$ that yielded the particular encryption. Let Z_0 be the event where $b = 0$ and $\mathcal{A}$ guesses $b^* = 1$ and let Z_1 be the event where $b = 1$ and $\mathcal{A}$ guesses $b^* = 1$. Then, a cipher $\mathbb{E}$ is* ***semantically secure*** *if the advantage*

$$Add_{SS}(\mathcal{A}, \mathbb{E}) = |\Pr(Z_0) - \Pr(Z_1)|$$

is negligible for all efficient adversaries.

Lemma 6. *Our one-time-used-key encryption method is semantically secure.*

Proof. In this part, we take the case of MLP as an example, and the case of CNN is similar to MLP.

- The polynomial-time adversary $\mathcal{A}$ chooses two messages $W_0 = \{W_0^{(l)}\}_{l=1}^{L}$ and $W_1 = \{W_1^{(l)}\}_{l=1}^{L}$, of equal length, and gives these to an encryption oracle.
- The encryption oracle generates a random key $sk = (\{\boldsymbol{r}^{(l)}\}_{l=1}^{L}, \boldsymbol{\gamma})$ such that $\boldsymbol{r}^{(l)} \in \mathbb{R}_{>0}^{n_l}$ and $\boldsymbol{\gamma} \in \mathbb{R}^{n_L}$, along with random $b = \{0, 1\}$, and encrypts the message $W_b = \{W_b^{(l)}\}_{l=1}^{L}$ using the key sk:

$$\widehat{W_b}^{(l)} = \begin{cases} R^{(l)} \circ W_b^{(l)}, \text{ for } 1 \leq l \leq L-1; \\ R^{(L)} \circ W_b^{(L)} + R^a, \text{ for } l = L, \end{cases}$$

  where $R^{(l)} \in \mathbb{R}^{n_l \times n_{l-1}}$ and $R^a \in \mathbb{R}^{n_L \times n_{L-1}}$ satisfy

$$\begin{aligned} R_{ij}^{(l)} &= \begin{cases} \boldsymbol{r}_i^{(1)}, & \text{when } l = 1 \\ \boldsymbol{r}_i^{(l)} / \boldsymbol{r}_j^{(l-1)}, & \text{when } 2 \leq l \leq L-1 \\ 1/\boldsymbol{r}_j^{(L-1)}, & \text{when } l = L \end{cases} \\ R_{ij}^{a} &= \boldsymbol{\gamma}_i \cdot \boldsymbol{r}_{a,i}. \end{aligned}$$

- The adversary $\mathcal{A}$ is then given the resulting ciphertext $\widehat{W_b} = \{\widehat{W_b}^{(l)}\}_{l=1}^{L}$. Finally, $\mathcal{A}$ outputs a guess $b^* \in \{0,1\}$.

Since the private key sk is randomly selected from positive real space, $\widehat{W_0}$ and $\widehat{W_1}$ are also uniformly random, which means that the distributions $\widehat{W_0}$ and $\widehat{W_1}$ are identically distributed (no algorithm can distinguish them). Then Z_0 and Z_1 are identical events and so

$$Add_{SS}(\mathcal{A}, our\ method) = |\Pr(Z_0) - \Pr(Z_1)| = 0,$$

which is negligible for all adversaries. □

2) Confidentiality of the true prediction for classification tasks: As shown in Section III-A, the client computes the noisy prediction vector $\hat{\boldsymbol{y}}^{(L)}$ as $\boldsymbol{y}^{(L)} + \alpha\boldsymbol{\gamma} \circ \boldsymbol{r}_a$, where $\boldsymbol{y}^{(L)}$ is the true prediction vector. The parameter α and noisy vector $\boldsymbol{r}_a$ are known to the client, while $\boldsymbol{\gamma}$ is chosen by the server randomly which is unknown to the client. Recall that there exists a partition $\sqcup_{s=1}^{m}\{I_s\}$ of $\{1, 2, \ldots, n_L\}$, such that for any i, j in the same I_s satisfy that $\boldsymbol{\gamma}_i = \boldsymbol{\gamma}_j$. The trade-off between the privacy-preservation of the true prediction and the extra computation and communication cost (which is related to m) is given in the following lemma.

Lemma 7. *For classification tasks, if $m = 1$, the probability that the clients obtain the true prediction is less than 1;*

If $2 \leq m \leq n_L$, the probability that the clients obtain the true prediction is less than or equal to $1/m$.

Proof. If $m = 1$, then $\boldsymbol{\gamma}_1 = \boldsymbol{\gamma}_2 = \cdots = \boldsymbol{\gamma}_{n_L}$ which is chosen randomly by the server, and denoted by $\boldsymbol{\gamma}_0$. Note that $\hat{\boldsymbol{y}}_i^{(L)} = \boldsymbol{y}_i^{(L)} + \alpha\boldsymbol{\gamma}_0\boldsymbol{r}_{a,i}$, $i = 1, 2, \ldots, n_L$. Since $\boldsymbol{r}_{a,1}, \boldsymbol{r}_{a,2}, \ldots, \boldsymbol{r}_{a,n_L}$ are pairwise distinct, the probability that the clients obtain the true prediction is obviously less than 1.

If $m = 2$, for $1 \leq s \leq m$, recall that $\boldsymbol{\gamma}_{I_s} := \boldsymbol{\gamma}_i$ (for $i \in I_s$), and let $\boldsymbol{y}_{s'}^{(L)} = \max_{i \in I_s}\{\boldsymbol{y}_i^{(L)}\}$, for some $s' \in I_s$. Then $\max_{1 \leq i \leq c}\{\boldsymbol{y}_i^{(L)}\} = \max_{1 \leq s \leq m}\{y_{s'}^{(L)}\}$. Then the probability that the clients obtain the true prediction is less than or equal to the probability that the clients obtain the maximal one among $\boldsymbol{y}_{1'}^{(L)}, \boldsymbol{y}_{2'}^{(L)}, \ldots, \boldsymbol{y}_{m'}^{(L)}$.

- For the case of MSE: The parameters $\hat{\boldsymbol{y}}_{1'}^{(L)}, \hat{\boldsymbol{y}}_{2'}^{(L)}, \ldots, \hat{\boldsymbol{y}}_{m'}^{(L)}$ and α are known to the clients, which satisfy that

$$\begin{cases} \hat{\boldsymbol{y}}_{1'}^{(L)} = \boldsymbol{y}_{1'}^{(L)} + \alpha\boldsymbol{\gamma}_{I_1}\boldsymbol{r}_{a,1'}, \\ \hat{\boldsymbol{y}}_{2'}^{(L)} = \boldsymbol{y}_{2'}^{(L)} + \alpha\boldsymbol{\gamma}_{I_2}\boldsymbol{r}_{a,2'}, \\ \quad\vdots \\ \hat{\boldsymbol{y}}_{m'}^{(L)} = \boldsymbol{y}_{m'}^{(L)} + \alpha\boldsymbol{\gamma}_{I_m}\boldsymbol{r}_{a,m'}. \end{cases} \tag{30}$$

  Since $\boldsymbol{\gamma}_{I_1}, \boldsymbol{\gamma}_{I_2}, \ldots, \boldsymbol{\gamma}_{I_m}$ are independent randomly chosen by the server, for any m-tuple $(\boldsymbol{y}_{1'}^{(L)}, \boldsymbol{y}_{2'}^{(L)}, \ldots, \boldsymbol{y}_{m'}^{(L)}) \in \mathbb{R}^m$, there always exists an m-tuple $(\boldsymbol{\gamma}_{I_1}, \boldsymbol{\gamma}_{I_2}, \ldots, \boldsymbol{\gamma}_{I_m})$ satisfying the Eq. (30). Thus the probability that the clients obtain the maximal one among $\boldsymbol{y}_{1'}^{(L)}, \boldsymbol{y}_{2'}^{(L)}, \ldots, \boldsymbol{y}_{m'}^{(L)}$ is less than or equal to $1/m$.
- For the case of CE: The parameters $\hat{\boldsymbol{y}}_{1'}^{*}, \hat{\boldsymbol{y}}_{2'}^{*}, \ldots, \hat{\boldsymbol{y}}_{m'}^{*}, \boldsymbol{\xi}_{1'}', \boldsymbol{\xi}_{2'}', \ldots, \boldsymbol{\xi}_{m'}'$ are also known to the clients, which satisfy that

$$\begin{cases} \hat{\boldsymbol{y}}_{s'}^{*} = \dfrac{1}{\exp(\boldsymbol{\delta}_{s'})} \cdot \dfrac{\exp(\boldsymbol{y}_{s'}^{(L)})}{\sum_{j=1}^{n_L}\exp(y_j^{(L)})}, \\ \boldsymbol{\xi}_{s'}' = \dfrac{\boldsymbol{\xi}_{s'}}{1 - \exp(\boldsymbol{\delta}_{s'})}, \end{cases} \tag{31}$$

for all $s = 1, 2, \ldots, m$. Since $\gamma_{I_1}, \gamma_{I_2}, \ldots, \gamma_{I_m}$, $\boldsymbol{\delta}_{1'}, \boldsymbol{\delta}_{2'}, \ldots, \boldsymbol{\delta}_{m'}, \boldsymbol{\xi}_{1'}, \boldsymbol{\xi}_{2'}, \ldots, \boldsymbol{\xi}_{m'}$ are independent randomly chosen by the server, for any m-tuple $(\boldsymbol{y}_{1'}^{(L)}, \boldsymbol{y}_{2'}^{(L)}, \ldots, \boldsymbol{y}_{m'}^{(L)}) \in \mathbb{R}^m$, there always exists an $3m$-tuple $(\gamma_{I_1}, \gamma_{I_2}, \ldots, \gamma_{I_m}, \boldsymbol{\delta}_{1'}, \boldsymbol{\delta}_{2'}, \ldots, \boldsymbol{\delta}_{m'}, \boldsymbol{\xi}_{1'}, \boldsymbol{\xi}_{2'}, \ldots, \boldsymbol{\xi}_{m'})$ simultaneously satisfying the Eq. (30) and Eq. (31). Thus the probability that the clients obtain the maximal one among $\boldsymbol{y}_{1'}^{(L)}, \boldsymbol{y}_{2'}^{(L)}, \ldots, \boldsymbol{y}_{m'}^{(L)}$ is also less than or equal to $1/m$.

□

According to Lemma 7, the clients cannot obtain the true prediction.

B. The Privacy of Client-side

In federated learning, the basic requirement is to protect the training data of each client (i.e., the privacy of client-side), and thus our method also needs to satisfy this requirement. As stated in Section II-C2, the server needs to recover the accurate model to ensure model accuracy. Hence, we ensure that the server can only get the aggregated gradients rather than the gradients of each client.

Lemma 8. *The proposed local gradients perturbation can ensure that the probability of getting the true gradients of each client is negligible for the semi-trusted server.*

Proof. Similar to [3], because the random number matrices $\{\boldsymbol{\phi}_1^{(l)}, \boldsymbol{\phi}_2^{(l)}, \ldots, \boldsymbol{\phi}_K^{(l)}\}_{l=1}^{L}$ that clients add are uniformly sampled from the real space, the value $\nabla \widehat{F}_k(\widehat{W}^{(l)})$ (see Eq. (27)) appears uniformly random to the server. Hence, there exist infinite pairs $((\frac{\partial \widehat{\mathcal{L}}}{\partial \widehat{W}^{(l)}})_k, \boldsymbol{\phi}_k^{(l)})$ satisfying the given $\nabla \widehat{F}_k(\widehat{W}^{(l)})$. That is, the probability of identifying the correct solution from an infinite number of solutions satisfying $\nabla \widehat{F}_k(\widehat{W}^{(l)})$ is almost zero. Therefore, the server cannot identify the noisy gradients $(\frac{\partial \widehat{\mathcal{L}}}{\partial \widehat{W}^{(l)}})_k$ of the k-th client. Similarly, the server also cannot identify individual additional noisy terms $\{\boldsymbol{\sigma}_k^{(l)}, \boldsymbol{\beta}_k^{(l)}\}$. □

Consequently, it is almost impossible for the server to obtain the true gradients of each client, let alone the local training data. Note that in the cross-silo FL [15], clients are different organizations (e.g. medical or financial), the network connection is relatively stable and the network bandwidth is relatively large. Thus, we can neglect the stragglers that cannot return the model updates to the server.

V. Performance Evaluation

We empirically evaluate our method on real-world datasets, from two different perspectives: **effectiveness** (i.e., how well our algorithm perform on these datasets) and **efficiency** (i.e., how much extra computation and communication cost our method spends).

A. Experimental Setup

We implement our methods based on the native network layer in Pytorch [27] running on single Tesla M40 GPU. We adopt FedAvg [22] as the baseline algorithm for comparison. In all experiments, the training epochs and the batch-size of each client are set to be 200 and 32, respectively. Unless specified otherwise, the number of private key partition m is set to 1 for the experiments on **effectiveness**, since different choice of m has little impact on the decryption for server. For the **efficiency** experiments, we will have a detailed discussion on the impact of m.

Datasets and Metrics. We evaluate our method on three privacy-sensitive datasets covering both the bank and medical scenarios and one image dataset.

- **UCI Bank Marketing Dataset (UBMD)** [24] is related to direct marketing campaigns of a Portuguese banking institution and aims to predict the possibility of clients for subscribing deposits. It contains 41188 instances of 17 dimensional bank data. Following conventional practise, we split the dataset into training/validation/test sets by 8:1:1. We adopt MSE as the evaluation metric.
- **APTOS Blindness Detection (ABD)** [2] consists of 3.6k training and 1.9k test datasets of retina images for predicting the severity of diabetic retinopathy. For preprocessing, we center-crop images and resize them into size of 64 × 64. We also use MSE as the evaluation metric.
- **Lesion Disease Classification (LDC)** [32] [7] provides 8k training and 2k test skin images for the classification of lesion disease. We downsample the images into 64 × 64 and adopt classification accuracy as the evaluation metric.
- **CIFAR-100** [17] contains 60000 color images of size 32 × 32 divided into 100 classes, each of which contains 600 images. Following conventional practise, the dataset is divided into 50000 training images and 10000 test images.

B. Experiments on regression and classification

We evaluate the training accuracy of our algorithm against native FedAvg on both regression and classification tasks. Besides, we present the computation and communication overhead of the basic building blocks in our method.

1) Regression: We evaluate the performance of our method on regression tasks with UBMD and LDC. For a more comprehensive comparison, we train ResNet20 and MLP with 3, 5 and 7 layers on ABD and UBMD, respectively. We also evaluate the performance for $k = 1, 5, 10$ clients on both datasets. Table I shows the MSE for the final converged model on testsets. From the table, the accuracy of our method elegantly aligns with that of FedAvg under various settings, which verifies our derivation in Section III. Note that some operations (e.g., dividing by random vectors) may cause precision errors, but the corresponding effects are negligible.

2) Classification: In this section, we evaluate our method for the classification task with ResNet20, ResNet32 and ResNet56 models on the LDC dataset. Both FedAvg and our method

[2] https://www.kaggle.com/c/aptos2019-blindness-detection

TABLE I: MSE Result for regression tasks. Lower MSE means better performance.

			$k=1$	$k=5$	$k=10$
UBMD	FedAvg	MLP-3	0.059	0.079	0.097
		MLP-5	0.059	0.079	0.100
		MLP-7	0.058	0.086	0.113
	Ours	MLP-3	0.060	0.078	0.097
		MLP-5	0.059	0.077	0.101
		MLP-7	0.059	0.082	0.114
ABD	FedAvg	ResNet20	0.048	0.085	0.117
	Ours	ResNet20	0.047	0.088	0.114

adopt cross entropy as the loss function. The accuracy of converged models on testsets is shown in Table II. Similar to regression tasks, the accuracy of our method elegantly aligns with FedAvg.

TABLE II: Accuracy result for classification task.

			$k=1$	$k=5$	$k=10$
LDC	FedAvg	ResNet20	69.42	69.27	69.14
		ResNet32	70.96	70.78	70.64
		ResNet56	72.69	71.61	71.10
	Ours	ResNet20	69.33	69.29	69.22
		ResNet32	71.05	70.84	70.62
		ResNet56	72.83	71.55	71.23

C. Experiments on local gradient perturbation mechanism

1) Effectiveness Verification: In this section, we evaluate the performance of our method with local gradient perturbation mechanism, i.e., both the training iterates from the server and local model updates from the clients will be protected. The experiments are conducted on CIFAR-100. We adopt ResNet20 as the base model and MSE as the loss function. We show the accuracy of converged models on testsets with 1, 5, 10 clients respectively in Table III. As can be seen from the table, our result still aligns well with that of FedAvg, even with the gradient perturbation mechanism. This verify the effectiveness our client-side privacy protection method.

TABLE III: Accuracy on CIFAR-100 for ResNet20.

	$k=1/5/10$
FedAvg	72.35 / 71.68 / 71.44
Ours	72.22 / 71.77 / 71.62

2) Security Verification: Recent research, dubbed DLG [41], found that training data may be recovered from the local gradient information returned from the clients in federated learning. Differential privacy (DP) methods, by adding noise on gradients before sharing, are shown to be able to prevent the privacy leakage at the cost of model accuracy. Therefore, we compare our method with DP from the following two aspects: (converged) model accuracy and dependability effect. More specifically, we train ResNet20 models in CIFAR-100 on 5 clients with our gradient perturbation mechanism and DP mechanism (of different levels of Guassian noises with variance ranging from 10^{-4} to 10^{-1}), respectively. [41] tries to reconstruct the input image by minimizing the L_2 difference between the gradient corresponding to the original input and the one corresponding to the reconstructed input. Therefore, lower L_2 difference means the reconstructed input is closer to the original one, i.e., the defendability is worse. Here, we adopt the same metric for the evaluation of defendability as in [41]. The results are shown in Table IV.

TABLE IV: Accuracy and defense results on CIFAR-100 of our method against DLG attack, compared with DP method of various level of Guassian noise.

	DP-10^{-4}	DP-10^{-3}	DP-10^{-2}	DP-10^{-1}	Our method
Accuracy	72.19	69.87	43.55	1.12	72.22
Defendability	0.017	0.092	0.217	0.238	0.235

From the table, we can see that our gradient perturbation mechanism achieves comparable accuracy with the DP of lowest level of noise (10^{-4}), while having the same level of defendability (10^{-1}) as DP of highest level of noise. This comes from the secret sharing nature of our method, and the added perturbation can be elegantly cancelled out with the server-side gradient aggregation. We also provide visualization of reconstructed result on a randomly sampled image from the testset of CIFAR-100 in Fig. 3.

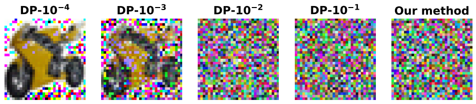

Fig. 3: Reconstructed results of DLG against different defenses.

D. Experiments of convergence

The main purpose of our method is to enable clients to train over encrypted models without accuracy loss. In this section, we further empirically prove this claim by comparing the convergence process of our method against the FedAvg [22]. Specifically, we train ResNet20 and ResNet32 [11] models on Lesion Disease Classification dataset [32] [7] for 200 epoches for our method and FedAvg, respectively. For the fairness of comparison, we use different random seeds for each run, and set the seeds for our method and FedAvg to be the same. Then we demonstrate the mean and standard deviation of test accuracy after each epoch in Fig. 4. As shown in the figure, overall, the convergence process of our method and FedAvg is almost exactly consistent. Although the curves of our method and the FedAvg may be noisy in the early stage, they tend to stabilize and converge to similar results as the training proceeds.

E. Communication and Computation

We dive into each of the components of our method and compare computational and communication overheads of our method with FedAvg. The experiments are conducted on ResNet56 with an input size of 224 × 224, a batch size of 32 and iteration number of 250. Note that the number of private key partition m only linearly affects the number of addition, which is negligible compared to the computational cost of other

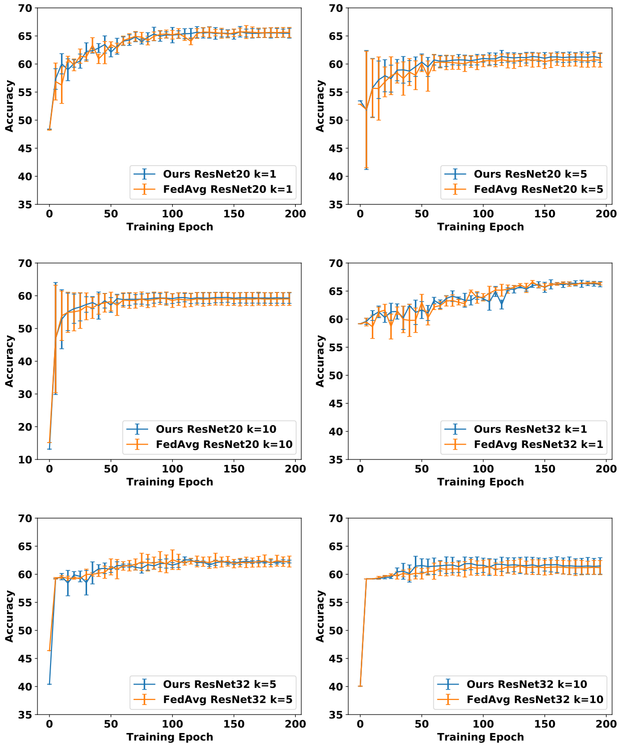

Fig. 4: Training curves for our method and FedAvg. The first and second rows demonstrate the results for ResNet20 and ResNet32 with 1, 5 and 10 clients respectively. The error bars stand for standard deviation and are shown every 5 epochs. Each experiment is repeated 5 times. Please note that it is best to view in color.

operations. Therefore, m is set as 1 for all the experiments and we will discuss the impact of m on communication cost in more details.

Table V shows the comparison of computational cost for our method and FedAvg, where **PP**, **LFP**, **LBP**, and **MU** stands for parameter perturbation, local forward propagation, local backward propagation and model update, respectively. From the table, we can see compared to FedAvg, the computational cost of our method has approximately doubled, which is mainly caused by the backward propagation on client side (i.e., 91.08 to 228.06 seconds). This is due to the computation of additional information, i.e. σ and β, for noisy gradient recovery. The increased cost of our method on the server side is basically negligible, indicating the server can afford the FL as usual.

TABLE V: Computational overhead of our method (Seconds).

		PP	LFP	LBP	MU	Total
FedAvg	Client	0	2.19	88.89	0	91.08
	Server	0	0	0	89.11	89.11
Ours	Client	0	2.20	225.86	0	228.06
	Server	3.66	0	0	92.39	96.05

The communication overhead is mainly affected by m. Specifically, the communication overhead mainly includes two interactions: the server sends noisy parameters $\widehat{W} = \{\widehat{W}^{(l)}\}_{l=1}^{L}$ together with the c-dimensional noisy vector $\boldsymbol{r}_a$ to clients and each client returns local noisy gradients $\{\nabla \widehat{W}\}_{l=1}^{L}$ together with extra noisy items $\{\boldsymbol{\sigma}^{(l)}, \boldsymbol{\beta}^{(l)}\}_{l=1}^{L}$ for MSE and $\{\boldsymbol{\sigma}^{(l)}, \boldsymbol{\beta}^{(l)}, \boldsymbol{\psi}^{(l)}\}_{l=1}^{L}$ for CE. Obviously, compared to FedAvg, the added communication costs are $\boldsymbol{r}_a$ and the noisy terms, where the cost of $\boldsymbol{r}_a$ can be negligible. Therefore, theoretical analysis shows that the additional communication is $\mathcal{O}(2m|W|)$ for MSE and $\mathcal{O}(3m|W|)$ for CE, where $|W|$ is the size of model parameters. The experiments also confirm our theoretical results. For example, when $m = 1$, both the server-to-client and client-to-server communication overheads in FedAvg are 0.85 MB, while the server-to-client and client-to-server communication overheads in our method are 0.85 MB and 2.55 MB, respectively.

In summary, in order to achieve the privacy preservation, we bring about certain amount of extra computational and communication costs. Nonetheless, we try the best to decrease the additional cost and keep it in constant level without decreasing the model accuracy compared to the original FL.

VI. Related Work

Federated learning was formally introduced by Google in 2016 [16] to address the data privacy in machine learning. Then, FedAvg [22] and its theoretical research [20] were introduced to implement and flourish FL. After that, many improvements and variants of FedAvg were deployed to deal with statistical challenges [31], [10], [23], communication challenges [1], [40], [6] and privacy issues[4], [36], [19], [3]. Considering the potential value of federal learning, many promising applications, such as healthcare [19], [37], virtual keyboard prediction [29], [39] and vehicle-to-vehicle communication [30], have tried to adopt FL as an innovative mechanism to train global model from multiple parties with privacy-preserving property.

Recently, some summary works on FL have been presented [8], [38], [18], [15]. Specifically, Dai *et al.* [8] provided an overview of the architecture and optimization approach for federated data analysis. Yang *et al.* [38] identified architectures for the FL framework and summarized general privacy-preserving techniques that can be applied to FL. Li *et al.* [18] provided a broad overview of current approaches and outlined several directions of future work of FL. Peter *et al.* [15] outlined the classification of FL and discussed recent advances and presented an extensive collection of open problems and challenges.

From the above, it is still a big challenge to effectively protect the intermediate iterates during the training phase and the final model parameters in FL [15].

VII. Conclusion

In this paper, we present a practical and bilateral privacy-preserving federated learning scheme, which aims to protect model iterates and final model parameters from disclosing. We introduce an efficient privacy-preserving technique to encrypt model iterates and final model parameters. This technique allows clients to train the updated model under noisy current model, and more importantly, ensures only the server can eliminate the noise to get accurate results. Security analysis shows the high security of our method under the honest but curious security setting. Besides, experiments conducted on real data also demonstrate the practical performance of our method.

References

[1] Naman Agarwal, Ananda Theertha Suresh, Felix X. Yu, Sanjiv Kumar, and Brendan McMahan. cpsgd: Communication-efficient and differentially-private distributed SGD. In *NeurIPS*, 2018.

[2] Nitin Agrawal, Ali Shahin Shamsabadi, Matt J. Kusner, and Adrià Gascón. QUOTIENT: two-party secure neural network training and prediction. In *CCS*, 2019.

[3] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for federated learning on user-held data. In *Advances in Neural Information Processing Systems - 30th NeurIPS Workshop on Private Multi-Party Machine Learning, (NeurIPS 2016)*, 2016.

[4] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In *CCS*, 2017.

[5] Dan Boneh. Prps and prfs. https://crypto.stanford.edu/~dabo/cs255/lectures/PRP-PRF.pdf, 2020.

[6] Yang Chen, Xiaoyan Sun, and Yaochu Jin. Communication-efficient federated deep learning with asynchronous model update and temporally weighted aggregation. *CoRR*, 2019.

[7] Noel C. F. Codella, Veronica Rotemberg, Philipp Tschandl, M. Emre Celebi, Stephen W. Dusza, David Gutman, Brian Helba, Aadi Kalloo, Konstantinos Liopyris, Michael A. Marchetti, Harald Kittler, and Allan Halpern. Skin lesion analysis toward melanoma detection 2018: A challenge hosted by the international skin imaging collaboration (ISIC). *CoRR*, 2019.

[8] Wenrui Dai, Shuang Wang, Hongkai Xiong, and Xiaoqian Jiang. Privacy preserving federated big data analysis. In *Guide to Big Data Applications*. 2018.

[9] Cynthia Dwork, Krishnaram Kenthapadi, Frank McSherry, Ilya Mironov, and Moni Naor. Our data, ourselves: Privacy via distributed noise generation. In *EUROCRYPT*, 2006.

[10] Hubert Eichner, Tomer Koren, Brendan McMahan, Nathan Srebro, and Kunal Talwar. Semi-cyclic stochastic gradient descent. In *ICML*, 2019.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[12] Briland Hitaj, Giuseppe Ateniese, and Fernando Pérez-Cruz. Deep models under the GAN: information leakage from collaborative deep learning. In *CCS*, 2017.

[13] Roger A. Horn and Charles R. Johnson. *Matrix Analysis, 2nd Ed.* Cambridge University Press, 2012.

[14] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. In *CVPR*, 2017.

[15] Peter Kairouz, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, and et al. Advances and open problems in federated learning. *hal-02406503*, 2019.

[16] Jakub Konecný, H. Brendan McMahan, Felix X. Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. *CoRR*, 2016.

[17] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images, 2009.

[18] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. Federated learning: Challenges, methods, and future directions. *CoRR*, 2019.

[19] Wenqi Li, Fausto Milletarì, Daguang Xu, Nicola Rieke, Jonny Hancox, Wentao Zhu, Maximilian Baust, Yan Cheng, Sébastien Ourselin, M. Jorge Cardoso, and Andrew Feng. Privacy-preserving federated brain tumour segmentation. In *Machine Learning in Medical Imaging - 10th International Workshop, (MLMI 2019)*, 2019.

[20] Xiang Li, Kaixuan Huang, Wenhao Yang, Shusen Wang, and Zhihua Zhang. On the convergence of fedavg on non-iid data. In *Proceedings of International Conference on Learning Representations (ICLR 2020)*, 2019.

[21] Wei Ma and Jun Lu. An equivalence of fully connected layer and convolutional layer. *CoRR*, 2017.

[22] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, (AISTATS 2017)*, 2017.

[23] Mehryar Mohri, Gary Sivek, and Ananda Theertha Suresh. Agnostic federated learning. In *ICML*, 2019.

[24] Sérgio Moro, Paulo Cortez, and Paulo Rita. A data-driven approach to predict the success of bank telemarketing. *Decision Support Systems*, 2014.

[25] Milad Nasr, Reza Shokri, and Amir Houmansadr. Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning. In *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*, pages 739–753. IEEE, 2019.

[26] Valeria Nikolaenko, Udi Weinsberg, Stratis Ioannidis, Marc Joye, Dan Boneh, and Nina Taft. Privacy-preserving ridge regression on hundreds of millions of records. In *IEEE S&P*, 2013.

[27] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *NeurIPS*, 2019.

[28] Le Trieu Phong, Yoshinori Aono, Takuya Hayashi, Lihua Wang, and Shiho Moriai. Privacy-preserving deep learning via additively homomorphic encryption. *IEEE Trans. Information Forensics and Security*, 2018.

[29] Swaroop Ramaswamy, Rajiv Mathews, Kanishka Rao, and Françoise Beaufays. Federated learning for emoji prediction in a mobile keyboard. *CoRR*, 2019.

[30] Sumudu Samarakoon, Mehdi Bennis, Walid Saad, and Mérouane Debbah. Federated learning for ultra-reliable low-latency V2V communications. In *GLOBECOM*, 2018.

[31] Virginia Smith, Chao-Kai Chiang, Maziar Sanjabi, and Ameet S. Talwalkar. Federated multi-task learning. In *NeurIPS*, 2017.

[32] Philipp Tschandl, Cliff Rosendahl, and Harald Kittler. The HAM10000 dataset: A large collection of multi-source dermatoscopic images of common pigmented skin lesions. *CoRR*, 2018.

[33] Jaideep Vaidya and Chris Clifton. Privacy-preserving *k*-means clustering over vertically partitioned data. In *ACM SIGKDD*, 2003.

[34] Wikipedia. Facebook-cambridge analytica data scandal. https://en.wikipedia.org/wiki/Facebook%E2%80%93Cambridge_Analytica_data_scandal, 2018.

[35] David Wu. Introduction to cryptography. https://www.cs.virginia.edu/dwu4/notes/CS255LectureNotes.pdf, 2012.

[36] Guowen Xu, Hongwei Li, Sen Liu, Kan Yang, and Xiaodong Lin. Verifynet: Secure and verifiable federated learning. *IEEE Trans. Information Forensics and Security*, 2020.

[37] Jie Xu and Fei Wang. Federated learning for healthcare informatics. *CoRR*, 2019.

[38] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. Federated machine learning: Concept and applications. *ACM TIST*, 2019.

[39] Timothy Yang, Galen Andrew, Hubert Eichner, Haicheng Sun, Wei Li, Nicholas Kong, Daniel Ramage, and Françoise Beaufays. Applied federated learning: Improving google keyboard query suggestions. *CoRR*.

[40] Hangyu Zhu and Yaochu Jin. Multi-objective evolutionary federated learning. *IEEE transactions on neural networks and learning systems*, 2019.

[41] Ligeng Zhu, Zhijian Liu, and Song Han. Deep leakage from gradients. In *NeurIPS*, 2019.