Efficient Privacy-Preserving Federated Learning for Resource-Constrained Edge Devices

Jindi Wu
Department of Computer Science
William & Mary

Qi Xia
Department of Computer Science
William & Mary

Qun Li
Department of Computer Science
William & Mary

***Abstract*—A large volume of data is generated by ubiquitous Internet-of-Things (IoT) devices and utilized to train machine learning models by IoT manufacturers to provide users with better services. Many deep learning systems for IoT data are required to perform all computation locally on small devices, which is not suitable for these resource-constrained devices. The devices can also send all the collected data to a server for costly model training by ignoring privacy concerns. To design an efficient and secure deep learning model training system, in this paper, we propose a federated learning system on the edge using the differential privacy mechanism to protect sensitive information and offload computation work from edge devices to edge servers, with consideration of communication reduction. In our system, a large-scale deep learning model is partitioned onto edge devices and edge servers, and trained in a distributed manner, in which all untrusted components are prevented from retrieving protected information from the training and inference process. We evaluate the proposed approach with respect to computation, communication, and privacy protection. The experiment results show that the proposed approach can preserve users' privacy while significantly reducing computation and communication costs.**

***Index Terms*—Federated Learning, Privacy-Preserving Algorithm, Communication Efficiency**

1. Introduction

Federated Learning (FL) enables distributed devices to collaboratively learn a shared machine learning model while keeping all the training data locally on devices. It is proven to be fitting for Internet-of-Things (IoT) devices or mobile devices. End devices, however, have limited computation and communication resources. Many times, it is not affordable for these devices to train a model locally. One solution is to offload the computation task to an edge server [1]. There will be two problems with respect to this approach. First, the large amount of training data needs to be transmitted to an edge server [2]. Second, the training data transmitted to the edge server will inevitably disclose much private information [3, 4, 5].

In this paper, we aim to design an efficient privacy-preserving federated learning (EPPFL) system for training machine learning model with resource-constrained edge devices. In our system, we deploy the first several layers for data feature extraction of the deep neuron network (DNN) on edge devices and the remaining layers on the edge server. An edge device extracts basic features from private raw data, then adds noise to the extracted features based on the differential privacy (DP) scheme. The processed data has a smaller size and preserves privacy compared with raw data. The edge server will take as input the processed features received from the edge device to perform the resource-consuming training process on the remaining layers. For the back-propagation process, the edge server will compute the gradients of its input and send them back to the edge device to continue the back-propagation on the feature extraction layers. After several iterations of interactions between edge server and edge device, an aggregation server will merge the local models computed by multiple edge device/server pairs into a global model. The global model will be further transmitted back to each edge device/server pair.

During the process, we will judiciously reduce the communication between the edge device and edge server by selectively dropping data. In addition, for model merging, local models will be transmitted to the aggregation server securely through a simple cryptographic scheme so that sensitive information will not be disclosed. To this end, our system can reduce communication and computation on edge devices, preserve edge devices' privacy.

To the best of our knowledge, none of the prior research has explored privacy-preserving, computation efficient, and communication efficient model training for FL on resource-constrained devices. Our model training architecture spanning over edge devices and edge servers are well-positioned to train on the data collected through edge devices. Our proposed techniques move most of the computation to edge servers while making an effort in reducing communication and preserving privacy. Our approach is adaptable for general DNN models. Moreover, the model can be partitioned based on the requirements of resource consumption and privacy protection of the task.

The contributions of this paper are threefold:

1) We propose a novel FL approach for resource-constrained devices, which reduces the communication & computation cost on edge devices.

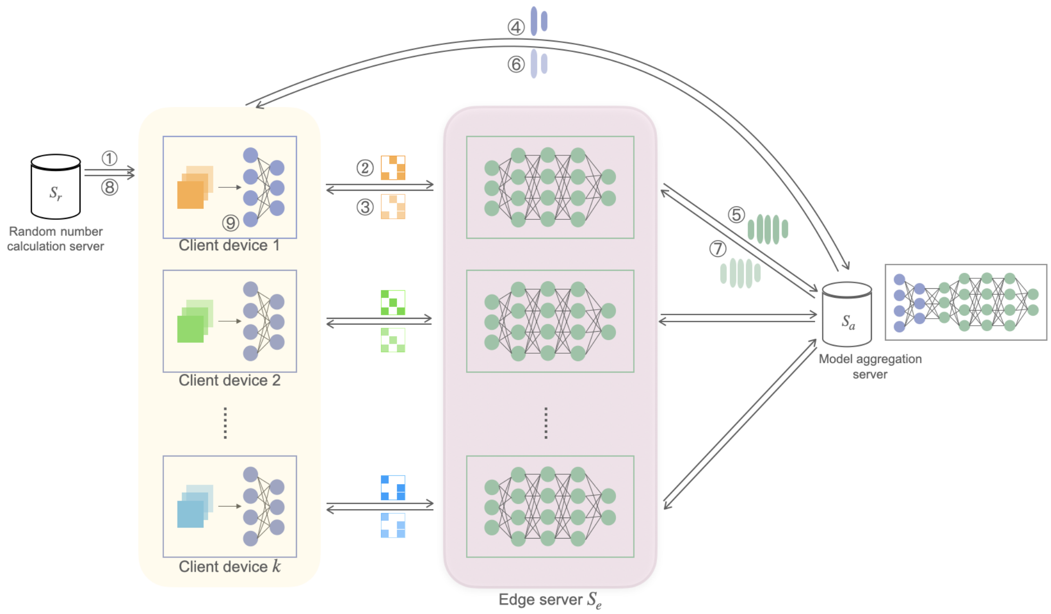

Figure 1. EPPFL system architecture. Local model training is performed by secure and efficient communication between clients and edge servers. Random number calculation server S_r is used to assist simple client-side model encryption and decryption. And the model aggregation server S_a merges the local models and broadcasts the global model.

2) We propose a secure and efficient model training framework, which reduces the response latency by offloading the computation to nearby edge servers and protects data on both data and model level.
3) We conduct a thorough experimental evaluation and analysis of the proposed approach.

2. The Proposed Framework

This section describes the proposed framework for the DNN training across multiple resource-constrained edge devices and edge servers with privacy protection and computation & communication cost reduction. The details of each component in the framework are provided in the following. And the important notations used in this paper are listed in Table 1.

2.1. Overview

The EPPFL system is composed of multiple resource-constrained client devices (edge devices) $C = c_1, c_2, ..., c_k$, multiple edge servers S_e, a model aggregation server S_a, and a third-party random number calculation server S_r (see Fig. 1). Each client has a private dataset d_i, $i \in \{1, 2, ..., k\}$, known as *local dataset*. These local datasets are disjoint with each other. Initially, given a large DNN M that needs to be trained, we partition it from a certain layer into two parts M_c and M_s, and maintain k copies of the neural network by deploying M_c on each client device and deploying the M_s that corresponds to each M_c on edge server S_e.

We assume all components and communication channels in the system are honest-but-curious, which means every component will honestly perform the assigned task but attempt to learn more information about others from received data. Hence, one of the aims of our method is to ensure that the private information of the client will not be leaked during the transmission and on any components in the system during model training and use.

TABLE 1. NOTATION LIST

Notation	Description
C	The set of clients
M_{c_i}	The client-side model deployed on the i-th client device C_i
M_{s_i}	The server-side model deployed on the edge server S_e and pairs to M_{c_i} on i-th client device
$W_{c_i}^t$	The weights of M_{c_i} in t-th global round
$W_{s_i}^t$	The weights of M_{s_i} in t-th global round
S_a	The model aggregation server
S_e	The set of edge servers
S_r	The random number calculation server
f_c	The process of model forward-propagation on client-side model
f_s	The process of model forward-propagation on server-side model
f_r	The random number generation function
A_i	The output of f_c on i-th client device C_i
$\bar{r}^t$	The average of random numbers generated by f_r in t-th global round
θ_c	The proportion of clients selected in each global round
θ_a	The proportion of activations selected to send from client to server in each local iteration
θ_g	The proportion of gradients selected to send from server to client in each local iteration

Each client in the system first trains a model with its private dataset, known as *local model*. Then FL aggregates n local models into a global model according to the weight η of the local model M_{local} by

$$M_{global} = \sum_{i=1}^{n} \eta_i M_{local_i} \tag{1}$$

We define the process between two model aggregations Eq. (1) as a *global round*, a local model update on a batch of private data as a *local iteration* of a client, and a learning process on the entire local dataset as a *local epoch*.

We describe our system by following the data flow in Fig. 1. The federated model training starts from the first *global round*: ① To initialize the random number generation function for clients, the third-party random number calculation server S_r sends clients a function f_r that generates a random number using a given random seed, client id, and global round id. The random number generated by f_r will be used in step ④ to encrypt model weights.

② Then a fraction of clients are randomly selected to make contributions to the current global round training. In a *local iteration*, each selected client trains a local model by using a batch of data from its private dataset to perform forward-propagation on M_c on the local device. An activation selection strategy is used to reduce the size of activations obtained from M_c and the (ϵ, δ)-DP mechanism is used to protect sensitive information contained in the activations. Then truth labels and the processed activations of the batch of data are transmitted to edge server S_e. ③ The edge server S_e continues the remaining forward-propagation on M_s and back-propagates M_s according to the training loss calculated from the objective function, then updates M_s using Stochastic Gradient Descent (SGD). We use the gradient selection strategy to reduce the size of the partial derivative of the M_s inputs that are also the outputs of M_c. Then the selected gradients are passed back to the client device to finish the optimization on M_c. And a *local iteration* is finished.

After finishing several epochs of local training, ④ the client adds a random number generated by f_r to W_c, the weights of M_c, to obscure the sensitive information in model weights, then transmits obscured client-side model to the aggregation server S_a. ⑤ And the edge server S_e sends M_s with trained weights W_s to the aggregation server S_a. The aggregation server S_a aggregates received local models into a global model by Eq. (1) and broadcasts the global model to ⑥ each client and ⑦ edge server S_e. ⑧ The third-party random number calculation server S_r computes $\bar{r}$, the average of the random numbers generated by presented clients in the current *global round*, and broadcasts to all clients. ⑨ Every presented client subtracts $\bar{r}$ from the received global client-side model weights and uses the result to replace its local M_c. Then the system is ready for the following global training rounds by repeating step ② to ⑨. The complete pseudo-code is given in Algorithm 1.

Algorithm 1: Efficient Privacy-preserving Federated Learning (EPPFL) Algorithm

Require: K clients $\{c_1, c_2, ..., c_k\}$, Private dataset d_i, $i \in [i, k]$, Model aggregation server S_a, Random number calculation server S_r, Edge server S_e
Ensure : Optimal global client-side and server-side model

Initialization;
for *global round* t *from 1 to* G **do**
  $C_t \leftarrow$ RandomChoose(C, θ_c);
  for *client index* i *from 1 to* $|C_t|$ **do**
    if $t \neq 1$ **then**
      c_i downloads random number average $\bar{r}^t$ from S_r;
      c_i downloads global client-side model weights $W_c'^t$ from S_a;
      c_i decrypts $W_c'^t$ by $W_{c_i}^t \leftarrow W_c'^t - \bar{r}^t$;
      S_e downloads global server-side model weights W_e^t from S_a;
    for *local epoch* e *from 1 to* E_{loc} **do**
      for *batch* (X_b, Y_b) *in* d_i **do**
        $M_{c_i}^t$ forward propagation by Eq.(2);
        c_i applies *activation selection* to A_b;
        c_i applies (ϵ, δ)-*DP* to kept A_b;
        $M_{s_i}^t$ forward propagation by Eq.(3);
        S_e calculates batch loss;
        $M_{s_i}^t$ backward propagation and update by Eq.(4);
        S_e applies *gradient selection* to $\mathrm{d}A_b'$;
        $M_{c_i}^t$ backward propagation and update by Eq.(5);
    c_i protects $W_{c_i}^t$ by Eq. (8) to get ${W'}_{c_i}^{t}$;
    c_i uploads ${W'}_{c_i}^{t}$ to S_a;
    S_e uploads $W_{s_i}^t$ to S_a;
  S_a aggregates local client-side models into $W_c'^{t+1}$ by (9);
  S_a aggregates local server-side models into W_s^{t+1} by (10);

2.2. Computation Offloading

In this subsection, we will explain the process of offloading the training computation from resource-constrained edge devices to edge servers. It is hard for edge devices to support the training of a large DNN model locally due to their limited computation resources, so we offload some computation from the end devices to edge servers. We partition a large DNN at a certain layer into two parts, each of which is composed of layers in the DNN model. We place the first part on the local device and another part on the edge server. In our EPPFL system, each client and its associated edge server train a DNN model cooperatively.

At the beginning of the training (*Initialization* in Alg. 1), the local client device c_i, $i \in [1, k]$, initializes the client-side model M_{c_i} with weights $W_{c_i}^1$; its associate edge server S_e initializes the server-side model M_{s_i} with weights $W_{s_i}^1$; and the random number calculation server S_r sends a secret random seed $seed$ and a random number generation function f_r to client devices.

In each global round, θ_c fraction of clients will be randomly selected to participate in the current global round training (*RandomChoose* in Alg. 1). We denote the active client set as C_t. Assuming we are seeing t-th global round, the active client c_i in C_t will train E_{loc} local epochs on his private dataset in mini-batch training style. In each local iteration, client c_i uses a batch of private training data (X_b, Y_b), where X_b is training instances and Y_b is truth labels of X_b, to perform forward propagation process f_c on M_{c_i} on local device by

$$A_b = f_c(X_b, W_{c_i}^t). \tag{2}$$

where A_b is the activations of extracted features by M_{c_i} from raw data X_b, and will be transmitted to the edge server for following resource-consuming training. To further reduce the communication cost and protect clients' privacy, we apply communication cost reduction methods and DP mechanism to A_b and get A_b' (see Section 2.3 and 2.4).

The associated edge server S_e of the client c_i takes A_b' as the input of the server-side model M_{s_i}, and performs the forward propagation f_s on M_{s_i} by

$$\hat{Y}_b = f_s(A_b', W_{s_i}^t) \tag{3}$$

where $\hat{Y}_b$ is the predicted label of X_b. Then the edge server S_e uses the predicted label $\hat{Y}_b$ and the ground truth Y_b to calculate the training loss L_b of the batch using the objective function ℓ.

The back-propagation phase follows the differentiation chain rule. The edge server S_e calculates the gradients of $W_{s_i}^t$ and the partial of L_b with respect to A_b', denote by $\mathrm{d}A_b'$. The server-side model is updated with learning rate η and the gradients of $W_{s_i}^t$ by

$$W_{s_i}^t = W_{s_i}^t - \eta \bigtriangledown \ell(W_{s_i}^t). \tag{4}$$

$\mathrm{d}A_b'$ should be sent back to client device for the model updating on M_{c_i}. To reduce the client receiving cost and accelerate the training process, we apply the gradient reduction method to $\mathrm{d}A_b'$ (see Section 2.3). The client device receives and uses the reduced $\mathrm{d}A_b'$ to continue the model updating of client-side model $M_{c_i}^t$ by

$$W_{c_i}^t = W_{c_i}^t - \eta \bigtriangledown f_c(W_{c_i}^t) \tag{5}$$

With the help of the edge server, only a small amount of computation is completed on client devices with limited resources, while following the same process of conventional local device training. So our approach can be applied to the training of general DNN models.

2.3. Communication Cost Reduction

When the DNN model is complicated and needs lots of iterations and epochs to train, or when the private training dataset is large, the communication cost is non-trivial for resource-constrained devices. To reduce the communication cost for edge devices, we propose two strategies for the forward-propagation and backward-propagation phases.

Activation selection policy is designed to reduce the message transmission for client devices while retaining selected features of training data in the forward-propagation phase. θ_a is a predefined meta-parameter, which determines what ratio of activations will be kept to transmit. We denote the number of activations in each channel as N_c. For each channel of activations, the activation selection policy randomly keeps $N_c\theta_a$ activations to transmit to the edge server as the input of the server-side model. We denote the selected subset of activations as A_{sub}.

For each channel of the partial derivative of training loss with respect to activations, $\mathrm{d}A$, **Gradient selection policy** finds out the ($N_c\theta_g$)-th largest number from the absolute values of $\mathrm{d}A$ and drops the values whose absolute values are smaller than the $N_c\theta_g$-th largest number. Intuitively, a gradient with a large absolute value means that the model will update relatively drastically in the specific direction, so the $\mathrm{d}A$ with large absolute values are selected to update the model first. The neurons that have small gradients will be updated in later iterations. This strategy not only contributes to the rapid convergence of the model but also helps to reduce the receiving cost of the devices.

During the training process without communication reduction methods, the client device needs to upload all the activations of the partition layer to the edge server and receives all the gradients of the activations with respect to the batch loss from the edge server during each iteration. Suppose a CNN model M is partitioned into M_c and M_s that are deployed on the edge device and the edge server respectively. Take a training image x as an example. In the forward passing phase, the size of A, the activations of x on M_c, is $N = kN_c$, where k is the number of convolutional kernels in the last layer of M_c and N_c is the size of the output of a convolutional kernel.

In each global round, we assume the local model will be trained E_{loc} local epochs, each local epoch contains T_{loc} iterations, and the client private dataset consists of D images. We use 32 bits float numbers to represent activations, so the size of the activations that need to be transmitted to the server is $4DE_{loc}N$ bytes. In addition, the labels of training data, which are set as 32 bits integer numbers, also need to be sent to the edge server. Therefore, the data amount transmitted from the client device to the edge server in a global round is $4DE_{loc}(N + 1)$ bytes. Similarly, in the back-propagation phase, the data amount the client device received from the edge server is $4T_{loc}E_{loc}N$ bytes. With the proposed communication cost strategies, however, the communication cost of an individual client device could be reduced to $4DE_{loc}(\theta_a N + 1)$ and $4\theta_g T_{loc}E_{loc}N$ bytes in sending and receiving phases, respectively. In the evaluation section, we show that the meta-parameters θ_a and θ_g can be set to 0.5.

2.4. Differentially Private Communication

In our approach, a general DNN model M could be partitioned into M_c and M_s at a selected layer. The model deployed on the client device M_c is composed of all the layers before the partition layer and their activation functions, e.g., ReLU. The input of M_c is the client's private training data and the output is the activations of the activation function on extracted features. The activations will be transmitted to the associate edge server and fed to the server-side model M_s for the remaining learning process. The privacy of the clients' training data, however, might be breached in the communication channel or stolen by the curious edge server. We apply the differential privacy scheme to the output of the client-side model after applying the activation selection policy.

We sample the activations according to the above-mentioned activation reduction policy and use $\tilde{a}^k_{i,j}(x)$ to denote the selected activation of the input x in k-th channel at position (i, j), which will be kept to transmit to the edge server for the following processing.

With the Laplace (ϵ, δ)-DP mechanism for client device selected output, the adversary should not tell if a specific instance exists in the client's private training dataset. Hence, the noise should be generated according to the sensitivity of the output of the $f^k_c(x, W_c)$, where f^k_c is the composition of k-th kernel in the last layer of M_c and the kernels in previous layers.

The instances x and x' from client's private dataset D and its neighboring dataset D' that has only one different instance from D, respectively. We have the sensitivity of the output of client device at position (i, j) with kernel k

$$\Delta f^k_c = max_{D,D'} \left\| \tilde{a}^k_{i,j}(x) - \tilde{a}^k_{i,j}(x') \right\| \tag{6}$$

So the final processed activations that are ready to be sent is

$$out^k_{i,j}(x) = \tilde{a}^k_{i,j}(x) + Lap\left(\frac{\Delta f^k_c}{\epsilon}\right) \tag{7}$$

We construct a (ϵ, δ)-DP mechanism for the client device output when partitioning a CNN model from the first convolutional layer and considering to apply K kernels of the convolutional layer on each pixel of image x. The training loss calculated on the edge with the activations that satisfy (ϵ, δ)-DP is ϵ_1-DP, where $\epsilon_1 = N_c\epsilon + c$. The model updating of M_s and M_c is $(O(\frac{|b|}{|d|}\epsilon_2\sqrt{T}), \delta_0)$-DP, where $\epsilon_2 = c' + c$ [6]. The privacy budget for local model updating increases as the number of iterations increases, which raises the sensitive information leakage risk. M_s is safe to be transmitted to the model aggregator for model merging because it is complex for post-analysis. However, M_c is vulnerable to model inversion attacks because its structure is simpler, and it has learned many basic features from raw private data. So, we propose an efficient secure model protection method for M_c in Section 2.5.

2.5. Efficient Secure Model Aggregation

We present a novel efficient secure model aggregation method. The third-party random number calculation server S_r defines a secret random seed $seed$ and constructs a random number generation function f_r, which generates a unique random number for each client in every global round by taking as input $seed$, the id of clients, and the current global round id t. Then the function f_r and the protected random seed $seed$ are send to every client.

After the local training in t-th global round, an active client c_i generates a random number r^t_i using f_r and protects the trained client-side model weights $W^t_{c_i}$ by

$$W'^t_{c_i} = W^t_{c_i} + r^t_i \tag{8}$$

where $W'^t_{c_i}$ is the obfuscated weights of client-side model M_{c_i}. The client c_i sends $W'^t_{c_i}$ to the model aggregator S_a for model merging. The edge server S_e directly sends the corresponding trained server-side model weights $W^t_{s_i}$ to S_a for aggregation since the $W^t_{s_i}$ satisfies DP as we showed before.

Assuming C_t is the set of clients who participate in the t-th global round and they are the same weighted, S_a aggregates their local models into global models by

$$W'^{t+1}_c = \frac{1}{|C_t|} \sum_{i=1}^{|C_t|} W'^t_{c_i} \tag{9}$$

$$W^{t+1}_s = \frac{1}{|C_t|} \sum_{i=1}^{|C_t|} W^t_{s_i} \tag{10}$$

The edge servers S_e can directly use the aggregated weights W^{t+1}_s to update the server-side model for next global round training. Nevertheless, W'^{t+1}_c is not the ready model weights for client-side model because it is protected by adding random numbers. To clean the global client-side model W'^{t+1}_c, the random number calculation server S_r calculates the average value $\bar{r}^t$ of the random numbers r^t_i generated by c_i, $c_i \in C_t$, and exposes $\bar{r}^t$ to all clients. After receiving global model weights W'^{t+1}_c, client c_i performs a model cleaning by $W^{t+1}_{c_i} = W'^{t+1}_c - \bar{r}^t$, and use the cleaned $W^{t+1}_{c_i}$ to replace its local model (line 6-8 in Alg. 1).

With this configuration, the aggregation server S_a and the random number calculation server S_r cannot infer any information about clients involved in the learning from their accessible data. The value of W_c can only be obtained by clients. In addition, W_c is protected by DP with a large privacy budget since it is updated with DP-protected activations. And a malicious client with limited observed data and resources is hard to retrieve sensitive information from the client-side model weights W_c by model inversion attacks. Compared with another popular model encryption method homomorphic encryption (HE), our secure aggregation method is faster and simpler in calculating resource-constrained edge devices, while ensuring the security of users' privacy.

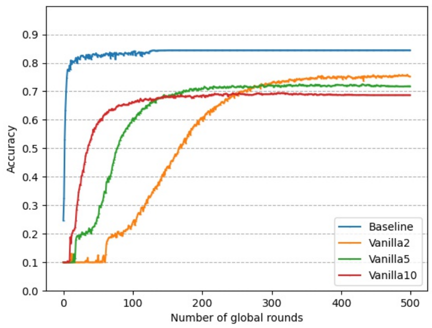

Figure 2. Accuracy in Vanilla settings

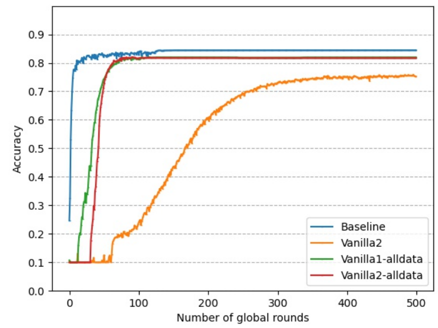

Figure 3. Accuracy in Vanilla and Vanilla-alldata settings

3. Evaluation

We implement EPPFL with the Pytorch framework [1]. We will analyze the evaluation results obtained from VGG-19 on the benchmark CIFAR-10 [7]. CIFAR-10 includes 60,000 colored images of size 32 $\times$ 32 from 10 classes. 50,000 of them are used for training and the remaining 10,000 images are reserved for testing. We equally split CIFAR-10 training images into 100 disjoint subsets of size 500, and assume 100 clients are available to participate in FL and each of them has 500 private images. At the beginning of each global round, $\theta_c = 1/10$ clients are randomly selected to train their local models with the same number of local epochs. In general, the model can be partitioned from any layer to satisfy different work offloading and privacy protection requirements. In our implementation, we deploy the first convolutional layer and ReLU function of VGG-19 on client devices and the remaining layers on the edge server. We will use the same hyperparameters in each setting. We set the mini-batch size to 16, use SGD to optimize the model with a learning rate of 0.01 and momentum of 0.5.

We will compare and analyze the impact of computation, communication, and privacy of the system on the accuracy of the VGG-19. The *Baseline* of our experiments is the setting in which 50,000 training images are centralized to train one-piece VGG-19 on a server.

3.1. Computation Cost

With the fixed size of the private dataset, hyperparameters, and model partition from the input layer, the number of local epochs a client trains directly impacts the computation cost on the client and the quality of the global model. Less local epochs require less computation power from resource-constrained devices. To evaluate the impact of the number of local epochs on model quality, we compare the model test accuracy in different *Vanilla* settings that are the standard FL settings in which the differential privacy scheme and communication cost reduction strategies are not applied. The only difference is the number of local epochs a client device trains in a global round.

1. GitHub: https://github.com/Jindi0/EPPFL

Fig. 2 shows the model accuracy with the number of local epochs of 2, 5, and 10 in each global round. The result shows that the global model converges more rapidly with more local epochs in a global round. However, the test accuracy is degraded. That is caused by the small size of the local dataset and the complexity of VGG-19. For each client, only 500 private images are used to train a large-scale VGG-19 model, which leads to model overfitting. In a typical real-world scenario, the local training data is always limited and does not match the scale of the target model. Therefore, fewer local epochs are preferred to avoid overfitting and improve model accuracy. For example, a relatively high model test accuracy of 75.92% is achieved in *Vanilla2* setting where each client trains two local epochs on the private dataset and submits its local model for model aggregation. The setting *Vanilla5* with 5 local epochs and *Vanilla10* with 10 local epochs achieve accuracy 72.34% and 69.46%, respectively.

In addition, *Vanilla2* requires 420 global rounds to make VGG-19 get converge, which is more than the number of global rounds needed by *Vanilla5* and *Vanilla10*, but it will not burden the individual clients because the clients are randomly selected in each global round. So the required time and the amount of computation for the participants are acceptable for resource-constrained devices.

Note that the model accuracy in *Vanilla* settings is much lower than *Baseline*, probably because less training data is used during each global round. In order to confirm this speculation, we split 50,000 training images into 10 clients, then make all of 10 clients participant in every global round, known as *Vanilla-alldata* setting. Fig. 3 illustrates that with a larger private dataset on each client and all training data are used in each global round, both *Vanilla1-alldata* and *Vanilla2-alldata* have improved accuracy and require fewer global rounds compared with *Vanilla* settings. However, they have similar accuracy, which proves that the size of the private dataset is an important factor in model accuracy.

3.2. Privacy

The privacy budget of DP is highly determined by the balance between the privacy protection and model quality

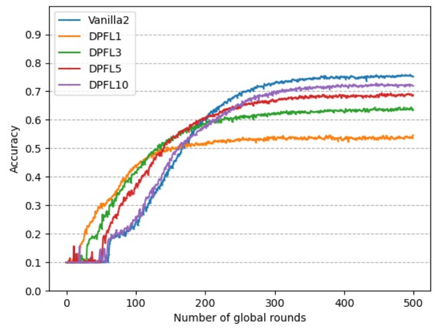

Figure 4. Accuracy in DPFL settings

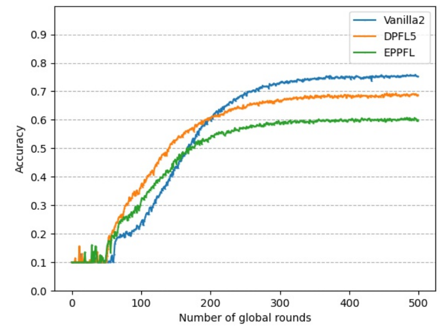

Figure 5. Accuracy in EPPFL setting

requirement of the application. To evaluate the impact of privacy budget on model quality in model-partitioned FL settings, we conduct four experiments in *DPFL* settings where the DP scheme is applied to the output of the client-side model. In these experiments, we make every client runs two local epochs per global round and record the test accuracy with privacy budget $\epsilon = 1, 3, 5, 10$ in Fig. 4. The experimental results show the model test accuracy gets degraded from 72.53% to 54.59% along with the privacy budget decreasing from 10 to 1. Compared with the model accuracy of *Vanilla2*, only 3.57% of the accuracy drops in *DPFL10*.

In addition, we notice that the accuracy difference between $\epsilon = 5$ and 10 is 3.15%, which is much smaller than that between $\epsilon = 1$ and 5, 14.8%. Therefore, to guarantee model accuracy, it is safe to choose a relatively large privacy budget ϵ. Note that we will reduce the transmitted activations and gradients for reducing communication costs, which further reduces the risks of sensitive information leakage.

3.3. Communication

In this subsection, we analyze the model accuracy of the proposed method *EPPFL* (see Fig. 5). We first set the local epoch to 2 and privacy budget $\epsilon = 5$. Then we set the proportion of kept activations $\theta_a = 0.5$ and the proportion of kept gradients $\theta_g = 0.5$. By reducing 50% of communication data between client and edge server with activation selection polity and gradient selection policy, the model accuracy of *EPPFL* drops to 60.78%, which is 8.6% lower than *DPFL5* and 15.14% lower than *Vanilla2*.

For the number of global rounds required to train VGG-19 in the settings, we have *EPPFL* $=$ *DPFL5* $<$ *Vanilla2*. Our approach not only reduces the communication cost between the client device and the edge server in each local iteration, but also reduces the overall communication cost of the FL system during the training process. Yet *EPPFL* drops the model accuracy by about 15.14% due to the data utility is impacted by the DP scheme and communication reduction strategies. The edge-side models have to learn from a fraction of perturbed image features and the client-side models only use a fraction of gradients to update. The learning is performed on data that is much less than the original private dataset. So, the accuracy drop of 15.14% is still acceptable.

In the future, we will conduct more experiments in EPPFL settings to explore the impact of θ_a and θ_g on model performance in communication cost and accuracy aspects.

4. Related Work

For DNN training in the federated learning scenario, there has been a large body of literature on privacy & security, and computation & computation efficiency.

To make a participant train a machine learning model without sharing private data while benefiting from other participants' models, Shokri et al. [8] first bring privacy in DNN joint learning in 2015. They proposed a secure distributed learning technique, in which each participant maintains a local model by uploading a fraction of model updates and perturbs them with the DP mechanism then updating his local model with a fraction of merged model updates. Unlike other differentially private approaches that aim at hiding a single instance of a participant's private dataset [9], Geyer et al. introduce client-level differential privacy in federated optimization to hide a participant in the training process [10].

Secure Multiparty Computation (SMC) framework has been used to train ML models with two non-colluding servers [11]. Three computing participants (3PC) models [12] allow participants to share data secretly among non-colluding servers. Truex et al. present a hybrid approach of DP and SMC to guarantee the privacy and performance of ML models trained with FL [13]. In addition to the privacy budget and sensitivity of the randomization algorithm, the approach takes the trust level into account when adding noise to the secret so that it alleviates the vulnerability of SMC and improves the model performance when using DP. Then the perturbed secret will be encrypted with HE and sent to the model aggregator, and the merged model is broadcast back to clients. However, the method is expensive in encrypting with HE. To solve this problem, BatchCrypt reduces the communication and computation cost of HE in

FL by encoding a batch of quantized gradients into a long integer and encrypt it in one go [14].

Moreover, in the distributed learning system, the attacker may also be malicious clients who upload the poisonous local updates to prevent the global model from converging. Xia et al. propose a fast aggregation algorithm FABA to remove the outliers from local updates and maintain the model performance [15], and VBOR to remove outliers with one-pass iteration [16]. However, the works mentioned above require edge devices to perform a great deal of computation and transmit a large volume of data, such as the gradients of each neuron of a DNN model, which is prohibitively expensive.

To reduce the training cost of end devices, Mao et al. split a DNN model into two parts and deploy them on edge device and edge server and apply DP in the communication channel. In this way, the clients offload the computing pressure to a server without violating privacy and only need to transmit the intermediate result and gradients of the cut layer [6]. Later, the approach was extended to adjust parallel training [17]. Our work further reduces the communication cost between clients and servers and proposes an efficient and simple encryption method to protect the privacy of data and models.

5. Conclusion

An efficient privacy-preserving federated deep learning system run on resource-constrained devices is proposed in this paper. Locally an edge device and an edge server collaborate to learn a local model. The local model training is partitioned on the local edge devices and the edge servers. This way, our system can learn users' private data without collecting them and offload the computation from the edge devices to the edge servers. The private features extracted from private data are protected with the differential privacy mechanism and transmitted from edge devices to the edge servers, and the untrusted components in the system will not learn about the protected information from differentially private features. Communication cost reduction strategies are also applied to save transmission bandwidth and further protect user privacy from leakage. In addition, to protect the local model from model inversion attacks, instead of using expensive homomorphic encryption, we use a simple and secure method with the help of a third-party random number calculation server. We evaluate our efficient and secure FL system with VGG-19 on CIFAR-10. The results show that our approach can preserve users' privacy with much-reduced communication and computation costs.

Acknowledgment

This project was supported in part by US National Science Foundation grant CNS-1816399. This work was also supported in part by the Commonwealth Cyber Initiative, an investment in the advancement of cyber R&D, innovation and workforce development. For more information about CCI, visit cyberinitiative.org.

References

[1] Z. Tao, Q. Xia, Z. Hao, C. Li, L. Ma, S. Yi, and Q. Li, "A survey of virtual machine management in edge computing," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1482–1499, 2019.

[2] X. Wang, Y. Han, C. Wang, Q. Zhao, X. Chen, and M. Chen, "In-edge ai: Intelligentizing mobile edge computing, caching and communication by federated learning," *IEEE Network*, vol. 33, no. 5, pp. 156–165, 2019.

[3] M. Hao, H. Li, G. Xu, S. Liu, and H. Yang, "Towards efficient and privacy-preserving federated deep learning," in *ICC 2019-2019 IEEE International Conference on Communications (ICC)*. IEEE, 2019, pp. 1–6.

[4] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 10, no. 2, pp. 1–19, 2019.

[5] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," *IEEE Signal Processing Magazine*, vol. 37, no. 3, pp. 50–60, 2020.

[6] Y. Mao, S. Yi, Q. Li, J. Feng, F. Xu, and S. Zhong, "Learning from differentially private neural activations with edge computing," in *2018 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 2018, pp. 90–102.

[7] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[8] R. Shokri and V. Shmatikov, "Privacy-preserving deep learning," in *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, 2015, pp. 1310–1321.

[9] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep learning with differential privacy," in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, 2016, pp. 308–318.

[10] R. C. Geyer, T. Klein, and M. Nabi, "Differentially private federated learning: A client level perspective," *arXiv preprint arXiv:1712.07557*, 2017.

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

[12] P. Mohassel and P. Rindal, "Aby3: A mixed protocol framework for machine learning," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 35–52.

[13] S. Truex, N. Baracaldo, A. Anwar, T. Steinke, H. Ludwig, R. Zhang, and Y. Zhou, "A hybrid approach to privacy-preserving federated learning," in *Proceedings of the 12th ACM Workshop on Artificial Intelligence and Security*, 2019, pp. 1–11.

[14] C. Zhang, S. Li, J. Xia, W. Wang, F. Yan, and Y. Liu, "Batchcrypt: Efficient homomorphic encryption for cross-silo federated learning," in *2020 {USENIX} Annual Technical Conference ({USENIX}{ATC} 20)*, 2020, pp. 493–506.

[15] Q. Xia, Z. Tao, Z. Hao, and Q. Li, "Faba: an algorithm for fast aggregation against byzantine attacks in distributed neural networks," in *IJCAI*, 2019.

[16] Q. Xia, Z. Tao, and Q. Li, "Defenses against byzantine attacks in distributed deep neural networks," *IEEE Transactions on Network Science and Engineering*, 2020.

[17] Y. Mao, W. Hong, H. Wang, Q. Li, and S. Zhong, "Privacy-preserving computation offloading for parallel deep neural networks training," *IEEE Transactions on Parallel and Distributed Systems*, 2020.