A Unified Federated DNNs Framework for Heterogeneous Mobile Devices

Xiaoli Li, *Student Member, IEEE*, Yuzheng Li, Shixuan Li, Yuren Zhou, *Member, IEEE*, Chuan Chen, *Member, IEEE*, and Zibin Zheng, *Senior Member, IEEE*

***Abstract*—Mobile devices can generate a tremendous amount of unique data, and thus, create countless opportunities for deep learning tasks. Due to the concerns of data privacy, it is often impractical to log all the data to a central server for training a satisfactory model. In federated learning, the participating devices can train a shared global model collaboratively while keeping their data locally. However, it is not a trivial task to train the deep neural networks (DNNs) with millions and billions of parameters on resource-constrained mobile devices in a federated manner. We replace each fully connected (FC) layer with two low-rank projection matrices to compact the DNNs model, and establish a global error function to recover the outputs of the compressed DNNs model. Then, we design a communication-efficient federated optimation Algorithm to reduce communication cost further. Considering that the heterogeneous devices may run different models at the same time, we devise three different training patterns to integrate the heterogeneous devices running different models. We conduct extensive experiments on both independently identically distribution (IID) and non-IID data sets. The experimental results demonstrate that the proposed framework can significantly reduce the number of parameters and communication cost while maintaining performance.**

***Index Terms*—Communication efficient, deep neural networks (DNNs), federated learning, heterogeneous, resource constrained.**

I. Introduction

SMART mobile devices, such as smartphones, tablets, wearable devices, and autonomous vehicles, are becoming more and more popular. These devices generate a tremendous amount of valuable data every day. Deep neural networks (DNNs) models learned based on these data, such as face recognition, medical diagnosis, and natural language processing, are gaining extensive attention [1]. The conventional training process of DNNs usually requires logging all the data to a central cloud as it needs a great number of computations on large data. However, due to the concerns of data privacy, it is often impractical to send users' raw data to the central cloud. On the other hand, once the training of DNNs is merely based on the local data of a single device, it will lead to non-precise results due to insufficient data. Hence, how to train a DNNs high-quality centralized model without violating user privacy is a challenge.

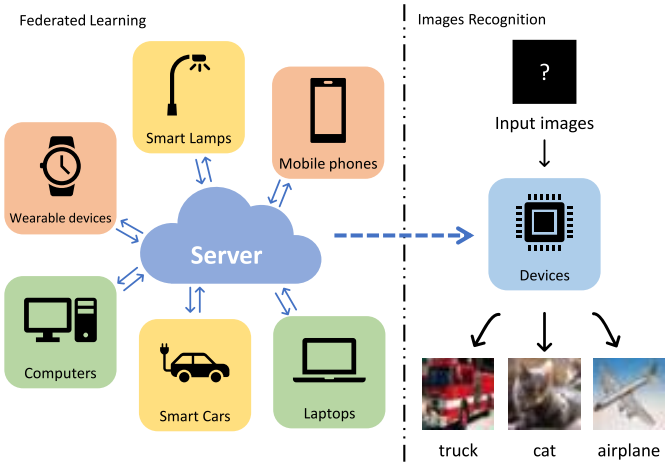

Fig. 1. Federated learning for image recognition on mobile devices: the mobile devices collaboratively train a model, and any device does not expose its local data to others. Each device in the federated framework obtains the global model and uses it locally for image recognition. The accuracy of the federated model is very close to the performance of the central model.

Recently, federated learning [2], [3] has been proposed. The participating devices (clients) in federated learning perform computation based on their local data and exchange information with the central server periodically to train a shared global model collaboratively. Different from conventional distributed machine learning, in federated Learning, computing devices hold their data and the central server cannot access the data on the devices directly or indirectly. Fig. 1 shows an example application of federated learning for image recognition on mobile devices. It works as follows: mobile devices communicate with a central server to download a global model, improve it by performing local training on their own user data, and send their local updates to the server. After receiving these updates, the server immediately aggregates them into a global update. After a certain number of iterative training, each device obtains the global model, and uses it locally for the tasks of image recognition. In addition, the accuracy of the federated model can be approximated close

Manuscript received February 22, 2021; revised May 24, 2021; accepted June 7, 2021. Date of publication June 14, 2021; date of current version January 24, 2022. This work was supported in part by the Key-Area Research and Development Program of Guangdong Province under Grant 2018B010109001; in part by the National Natural Science Foundation of China under Grant 11801595; in part by the Guangdong Basic and Applied Basic Research Foundation under Grant 2019A1515011043; in part by the Natural Science Foundation of Guangdong under Grant 2018A030310076; and in part by the Tencent Wechat Rhino-Bird Project under Grant 2021321. *(Corresponding author: Chuan Chen.)*

The authors are with the School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou 510275, China (e-mail: lixli27@mail2.sysu.edu.cn; liyzh23@mail2.sysu.edu.cn; lishx7@mail2.sysu.edu.cn; zhouyuren@mail.sysu.edu.cn; chenchuan@mail.sysu.edu.cn; zhzibin@mail.sysu.edu.cn).

Digital Object Identifier 10.1109/JIOT.2021.3088867

to the performance of the central model, which is trained by putting all data together. Federated learning provides globally model training while ensuring privacy, so it is increasingly attractive.

However, federated learning on mobile devices is a quite difficult task, as it requires integrating DNNs into smart mobile devices to obtain intelligence. The storage cost of DNNs is unaffordable [4], [5] for mobile devices due to the hundreds of millions of parameters. For example, an 8-layer-AlexNet with more than 600 000 parameters costs about 240-MB storage. In contrast, some mobile devices have strict constraints in terms of memory capacity. It is very problematic that DNN models are becoming more and more complex in terms of memory, and most common hardware platforms are not able to keep up with the exponential growth of the neural networks' sizes [6].

Model compression is considered a solution that can fill the computing gap between mobile devices and DNNS. Gradient sparsification [7], [8] and gradient quantization [9] have been dedicated to reducing the high network communication cost for distributed deep learning. However, these methods focus on the compression model in the communication process, and still need to learn a full model update in the training process. By combining pruning, training, quantization, and Huffman coding, Han *et al.* [10] reduced the storage requirement of DNNs by 35× to 49× without affecting their accuracy. However, these methods require additional definitions of new operations. The devices in federated learning need to exchange their models or updates with others, and these new operations need to be unified and shared by all devices. Thus, it is difficult to perform pruning, training quantization, and Huffman coding in federated learning. Another model compression method is to design lightweight DNN directly, such as Mobilenetv [11], ShuffleNet [12], etc. However, these mobile-friendly deep architectures require pretrain and fine-tune the network parameters based on a huge amount of data. This is not suitable for federated learning, which has no centralized data. Konecný *et al.* [13] proposed two structured updates methods for communication-efficient federated learning, which can directly learn an update from a restricted space. Structured updates methods retain the information obtained in the training process, which are better than the unstructured update (such as sparsification and quantization). However, the structured updates methods require generating a low-rank reconstruction matrix randomly or a predefined sparse pattern for each client independently, and the global model obtained by these methods cannot be reused by other devices due to the lack of local low-rank reconstruction matrices or sparse patterns. For the new devices to join the federated learning, the DNNs model needs to be retrained.

Different from the above methods, we define a federated DNNs compression (FDNNC) framework for resource-constrained mobile devices. The proposed compression model uses a low-rank representation of the weight matrices of the fully connected (FC) layers to compact the DNNs models. Each FC layer is replaced by two low-rank projection matrices. Instead of minimizing the reconstruction error between the original FC layers parameters and the projection matrices, FDNNC directly establishes an objective function to recover the outputs of the compressed DNNs. In this way, the weight parameters of the interlayer and intralayer relationship can be compressed jointly without the tedious retraining procedure. In addition, the global DNNs model generated by FDNNC is generalized for other devices, and the new devices only need to do a few times of training based on the global model to get a well-performed model.

After the compression of the weight matrices of the FC layers, the communication cost is reduced at a certain. Considering that mobile devices typically have minimal upload bandwidth or expensive connections, we further proposed a communication-efficient federated optimation (CEFO), to reduce the communication cost further. Different from the federated averaging algorithm [14], which sends all the updates of each client to the server, CEFO only sends a part of the updates of each client. In this way, not only communication cost can be reduced but also update gradients can be protected from exposure [15], [16].

Furthermore, previous work only considered the scenario of running the compression DNNs model or original DNNs model on all devices, but there is possible that devices in federated learning may run different models at the same time due to system heterogeneity. At present, a lot of works have proposed the issue of system heterogeneity [17]–[22], that is, the devices in federated learning are heterogeneous, which vary in terms of hardware (i.e., CPU and memory). For example, there are some mobile devices with larger memory, such as intelligent sensors or intelligent monitoring devices, which train together based on the original DNNs model at the beginning. After a while, a group of small mobile devices running the compression DNNs model with limited storage resources join the training process. However, the original large mobile devices may not update the training model in time and still upload the updates of the original DNNs model; or these large mobile devices may not want to replace the model because they have gained a well-performed original DNNs model. Therefore, how to design a unified federated deep learning framework for heterogeneous mobile devices is a challenging problem. In this article, we present three different training patterns to train the heterogeneous mobile devices running the original DNNs model and the compression DNNs model at the same time.

Our contributions can be summarized as follows.

1) To compact the DNNs model for resource-constrained mobile devices, we replace each FC layer by two low-rank projection matrices, and establish an objective function of the global error to recover the outputs of the compressed DNNs model. The proposed compression DNNs model compresses the weight parameters of interlayer and intralayer relationship jointly, and greatly reduces the number of parameters without significantly reducing the prediction accuracy.
2) Based on the compression model, we propose a CEFO approach, which only sends a part of the updates of each client to the server. The mobile devices in federated learning are typically large in number and require significant communication bandwidth for updates exchange. However, mobile devices frequently

have minimal upload bandwidth or expensive connections. In this way, not only communication cost can be reduced further but also update gradients can be protected from exposure.

3) Unlike previous studies that only consider the devices running the same model, we present three different training patterns to train the heterogeneous mobile devices running the original model and the compression model at the same time. Due to variability in hardware (CPU and memory), the storage memory of each device in federated learning may be heterogeneous. Furthermore, there is a possibility that the heterogeneous devices run different models at the same time. To our best knowledge, we are the first to integrate the heterogeneous devices running different models in federated learning.
4) To evaluate the advantages of our framework, we take convolutional neural networks (CNNs), one of the most popular DNNs, as an instance. We conduct extensive experiments on both independently identically distribution (IID) and non-IID data sets. The experimental results show that the proposed framework can significantly reduce the number of parameters and communication cost while maintaining performance.

The remainder of this article is organized as follows. Section II summarizes the related work. Section III introduces the proposed framework. Section IV summarizes the experimental results and analysis. Finally, Section V concludes the article.

II. Related Work

System Heterogeneity: Different from traditional distributed machine learning, federated learning involves the participation of an extremely large number of heterogeneous devices. The participating devices vary in terms of hardware, such as computation capabilities, storage, and memory. Yang *et al.* [23] proposed that the low-end devices are the major reason for client failure, which may have negative impacts on the performance of federated learning. They suggested that the federated learning platform should consider device heterogeneity that is ignored in existing platforms. Nishio and Yonetani [18] proposed a client selection protocol to address the training bottleneck of the devices with limited computational resources. Wang *et al.* [20] considered the computation resource constraint, and proposed a control Algorithm to the adaptive tradeoff between local update and global aggregation under a resource budget constraint. Nguyen *et al.* [21] presented that the devices with limited resources and low computation capacity may take more than one communication block to complete one global round; thus, they formulated a resource allocation problem of total energy consumption and completion time. Kang *et al.* [19] proposed that the heterogeneous computation resource of devices leads to varying resource cost; thus, it is necessary to design economic compensation to stimulate devices to participate in model training. Abdelmoniem and Canini [22] proposed an adaptive model quantization federated learning method (AQFL), which can reduce the degree of device heterogeneity by uniformizing the computing resources of the clients. However, AQFL requires the server to collect the clients' computational profiles, which is unrealistic under strict privacy constraints. Moreover, the server has to customize the quantized model of each client, and dequantizes the model updates of each client in each iteration. In contrast, our framework is simpler and more practical, in which the server does not need to initialize a different model for each client, and the workload is much less than AQFL.

Model Compression: Some mobile devices have strict constraints in terms of memory capacity and storage. The deployment of DNN models on these devices is gaining more and more attention [24]–[27]. Various optimization techniques have been proposed to reduce the model size of deep learning on resource-constrained mobile devices, such as vector quantization [28], hashing techniques [29], circulant projection [30], parameter pruning [31], and sparsity [32]. These methods greatly reduced the model size without impacting the effectiveness. However, these methods require additional definitions of new operations; thus, it is difficult to perform them in federated learning. In addition to these methods, using low-rank methods to compress deep learning has a long history. Denton *et al.* [33] applied truncated singular value decomposition (SVD) to compress the weight matrix of the FC layer. Kim *et al.* [26], Novikov *et al.* [34], and Tai *et al.* [35] used a tensor decomposition to approximate the parameters of the pretrained weights and then fine-tuned them to compensate for the performance loss. The above low-rank approaches only considered approximating the parameters between compressed weights with pretrained weights by minimizing their Euclidean distance. This setting is indeed problematic for federated learning without pretrained weights. Konecný *et al.* [13] proposed a low-rank structured updates approach, which directly learns an update from a restricted space parameterized using a smaller number of variables in federated learning. However, the structured updates approach requires generating predefined patterns afresh in each round and for each client independently, so resulting in low adaptive ability and efficiency.

Communication Efficiency: In federated learning, mobile devices are required to exchange parameters with the central server frequently, which leads to huge communication overhead. Many effective communication methods have been proposed to reduce the communication cost. For example, McMahan *et al.* [14] focused on reducing the rounds of communication, and proposed that clients should perform multiple iterations locally to calculate the weight update instead of communicating after every iteration. However, this requires more complex calculations on the clients to evaluate gradients. Xie *et al.* [36] proposed an asynchronous update strategy, in which the central server does not need to wait for a certain number of responding clients, as long as there are updates sent by the client, the server can carry out aggregation. The asynchronous update strategy greatly reduces network congestion, but it may reduce the convergence speed due to staleness. Besides, gradient quantization and sparsification, which reduce communication data size, are also extensively studied. Seide *et al.* [37] used only 1-bit to represent the updates. Wen *et al.* proposed TernGrad [9], which stochastically quantizes gradients to ternary values. DoReFa-Net [38] quantizes

the weights with 1-bit and gradients with 2-bit. Strom [39] presented a sparsification approach, which only sends gradient updates that are greater than a certain predefined threshold to the server. Instead of using a fixed threshold, Aji and Heafield [7] used a fixed sparsity rate to communicate small subset parameters with the biggest magnitude. These communication-efficient approaches reduce the communication cost, but they do not consider the resource-constrained problem of the mobile devices, which make it impractical for the devices to run the excessive model with millions and billions of parameters in the local training process; thus, they are not applicable to the mobile devices with limited memory.

III. Method

A. Preliminaries

We define a training batch $X = [x_1, x_2, \ldots, x_n] \in \mathbb{R}^{d\times n}$, where d is the dimension of feature vectors, and n is the batch size. The forward propagation of the lth FC layer in a DNNs model can be written as

$$a^l = f\left(z^l\right), \text{ where } z^l = W^l a^{l-1} + b^l \tag{1}$$

where the weight matrix $W^l \in \mathbb{R}^{M^l\times N^l}$ and the bias vector $b^l \in \mathbb{R}^{M^l}$ define the transformation. The vectors z and a denote the activation units before and after transformation, and $f(\cdot)$ represents the transition function, such as sigmoid, tanh, and ReLU.

When M^l and N^l are large, the memory cost is unaffordable. An intuitive way is to compress W^l. The state-of-the-art low-rank decomposition methods [40]–[42] only consider approximating the pretrained parameters W^l with compressed one $\hat{W}^l$ by minimizing their Euclidean distance $\|W^l - \hat{W}^l\|_F^2$. However, in federated learning, each device computes the update gradients locally with their own data. It is indeed problematic to pretrain and perform parameters decomposition on the federated scenario.

B. Compression DNNs Model

Denil *et al.* [40] showed that DNNs models are overparameterized, and the features can be learned given only a small fraction of their parameters. The weights of the FC layers dominate memory consumption, which reached the memory limit with 89% [43] or even 100% [42]. Therefore, an intuitive insight is to discover and discard the parameter redundancy in FC layers while maintaining the performance. In this section, we introduce a compression DNNs model for resource-constrained mobile devices, which uses an SVD-based low-rank decomposition to replace the original weight matrix of the FC layer. Then, we establish an objective function of the global error to recover the outputs of the compressed DNNs model and adopt a stochastic gradient descent to optimize the parameters. Fig. 2 shows the comparison between the original DNNs model and the compression DNNs model.

1) Low-Rank Decomposition: We first adopt SVD to replace W^l with low-rank decomposition: USV^T, where $U \in \mathbb{R}^{M^l\times r}$, $V \in \mathbb{R}^{N^l\times r}$, and $S \in \mathbb{R}^{r\times r}$ (S is a diagonal matrix and r is the SVD-rank). We obtain the low-rank decomposition of W^l as $A^l B^{lT}$, where $A^l = US^{(1/2)}$ and $B^l = VS^{(1/2)}$.

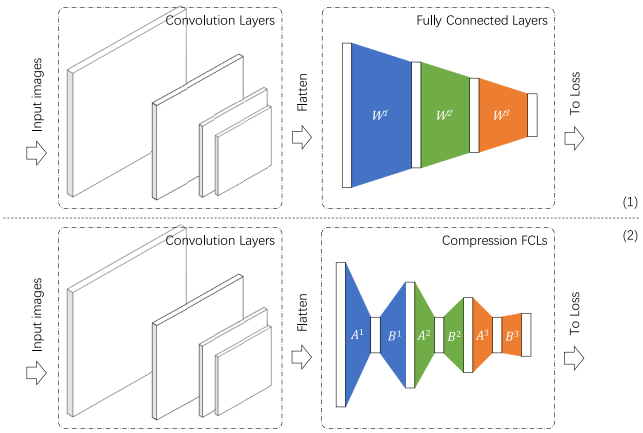

Fig. 2. Comparison between the original DNNs model and the compression DNNs model. 1) In the original DNNs model, the weight matrices of the FC layers dominates memory consumption. 2) In the compression DNNs model, we use two low-rank projection matrices to replace the FC layers to compact the DNNs model.

2) Forward Propagation: The forward propagation of the lth layer in (1) can be can be rewritten as

$$a^l = f\left(z^l\right), \text{ where } z^l = A^l B^{lT} a^{l-1} + b^l. \tag{2}$$

In particular, if the DNNs model has m fully connected layers, the output of the last layer a^m is as follows:

$$a^m = f \circ \left(A^m \times B^{mT}\right) \circ \cdots \circ f \circ \left(A^1 \times B^{1T}\right) x + b^m + \cdots + b^1. \tag{3}$$

3) Backpropagation: Instead of minimizing the divergence of $\hat{W}^l$ and W^l, much attention should be put on the predictions of the model; thus, we try to minimize the global error of $\|y - a^m\|_F^2$ to recover the outputs of the compressed DNNs model, where y is the true label set of the training set. Subsequently, we use the stochastic gradient descent to solve the nonconvex optimization problem, and the compression DNNs model is further jointly optimized among layers via backpropagation. In this way, the proposed compression DNNs model compresses the weight parameters of interlayer and intralayer relationship jointly. The objective function can be written as

$$\min_{l=1\cdots m} J\left(A^l, B^{lT}, b^l\right) = \frac{1}{2}\left\|y - a^m\right\|_F^2. \tag{4}$$

The error signals of the objective function can be calculated as

$$\delta^l = \begin{cases} \frac{\partial J}{\partial a^l} \odot f'\left(z^l\right), & l = m \\ \left(A^{l+1} B^{l+1^T}\right)^T \delta^{l+1} \odot f'\left(z^l\right), & \text{otherwise} \end{cases} \tag{5}$$

where $\odot$ indicates elementwise multiplication.

The gradient of the objective function with respect to all parameters (A^l, B^{l^T}, b^l for all layers) is calculated as

$$\frac{\partial J}{\partial A^l} = \delta^l \left(a^{l-1}\right)^T B^l, \quad \frac{\partial J}{\partial B^{lT}} = A^{lT} \delta^l \left(a^{l-1}\right)^T, \quad \frac{\partial J}{\partial b^l} = \delta^l. \tag{6}$$

TABLE I
COMPARISON OF THE MEMORY USAGE OF SVD-FC LAYER AND FC LAYER

Operation	Memory
FC forward pass	$O(M^l \times N^l)$
SVD-FC forward pass	$O(r \times max(M^l, N^l))$
FC backward pass	$O(M^l \times N^l)$
SVD-FC backward pass	$O(r^3 \times max(M^l, N^l))$

The gradients in (6) are used to update the parameters. At the kth iteration, the new parameters are as follows:

$$A^l_{k+1} = {A_k}^l + \alpha_k \frac{\partial J}{\partial A^l}$$
$$B^{lT}_{k+1} = {B^l_k}^T + \alpha_k \frac{\partial J}{\partial B^{lT}}$$
$$b^l_{k+1} = b^l_k + \alpha_k \frac{\partial J}{\partial b^l} \tag{7}$$

where α is the learning rate.

We summarize the comparison of the memory usage of the SVD FC layer (SVD-FC) and the original FC layer in Table I. When $r \prec \min(M^l, N^l)$, our method would greatly reduce the number of parameters of the FC layers. In addition, the compression DNNs model has the following important advantages.

1) `Ease of Implementation:` SVD, as one of the standard tools in matrix decomposition, is easy to implement. The decomposition of FC layers is easy to insert into a DNN learning process.
2) `Ease of Fine-Tuning:` Once the FC layers are compressed, it is straightforward to fine-tune the entire network using backpropagation.

C. Communication-Efficient Federated Optimation

Communication is a key bottleneck in federated learning. On the one hand, mobile devices in federated learning are typically large in number and frequently exchange the parameters with the remote server, which requires an enormous amount of network bandwidth. However, the network bandwidth of the server is usually restricted and shared among all running devices, which leads to a vast communication overhead. On the other hand, mobile devices frequently have narrow upload bandwidth or expensive connections. The uplink is typically slow, and all the devices need to wait for the slowest one in each synchronous round.

After the compression of the weight matrices of the FC layers, the size of the messages communicated at each round of the lth FC layer is reduced from $M^l N^l$ to $(M^l + N^l) \times r$. However, for the mobile devices that have narrow upload bandwidth, there is much room to be improved. As shown in Fig. 3, we propose a CEFO approach to reduce the communication cost further.

The kth iteration consists of the following steps.

1) *Model Selection at Clients:* A subset of existing clients (the number of clients included in the subset is c, and $c \succ 0$) is selected, each of which downloads the current weight matrices of the global model from the server. The weight matrices covering all layers are denoted as $A_k, {B^T}_k$, and b_k

$$A_k = \left[A^m_k, A^{m-1}_k, \ldots, A^1_k\right] \tag{8}$$
$${B_k}^T = \left[B^{mT}_k, {B^{m-1}_k}^T, \ldots, {B^1_k}^T\right] \tag{9}$$
$$b_k = \left[b^m_k, b^{m-1}_k, \ldots, b^1_k\right]. \tag{10}$$

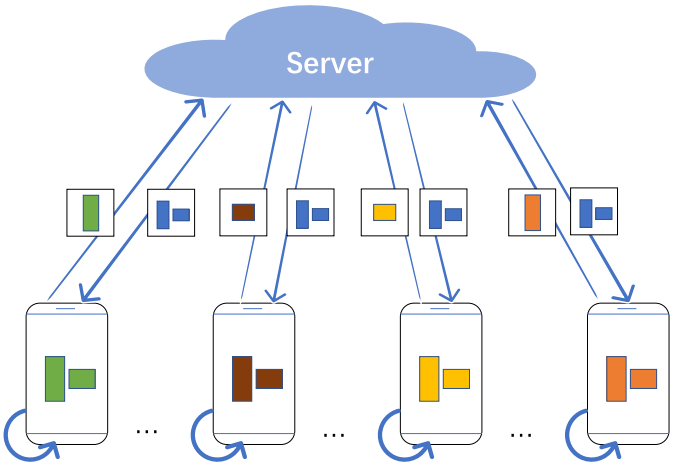

Fig. 3. CEFO: the devices running the compression DNNs model obtain two low-rank weight matrices in each FC layer. Different from the previous works, which send all the two weight matrices to the server, half of the clients send a low-rank weight matrix, and the other half send another weight matrix to the server.

2) *Local Gradients Updates:* Each client in the subset computes the gradient $G_{A^k}, G_{B^{T^k}}$, and G_{b_k} of the weight matrices $A_k, {B^T}_k$, and b_k for one minibatch of their local data

$$G_{A_k} = \left[g\left(A^m_k\right), g\left(A^{m-1}_k\right) \cdots g\left(A^1_k\right)\right] \tag{11}$$
$$G_{B^T_k} = \left[g\left(B^{mT}_k\right), g\left({B^{m-1}_k}^T\right) \cdots g\left({B^1_k}^T\right)\right] \tag{12}$$
$$G_{b_k} = \left[g\left(b^m_k\right), g\left(b^{m-1}_k\right) \cdots g\left(b^1_k\right)\right]. \tag{13}$$

Different from the previous works, which require to send all the updates $G_{A^k}, G_{B^{T^k}}$, and G_{b^k} to the server, we randomly select half of clients to send G_{A^k} and G_{b^k}, and the other half send $G_{B^{T^k}}$ and G_{b^k} to the server. In this way, not only communication cost of the lth FC layer can be reduced to $(M^l + N^l) \times r/2$ but also the gradient updates can be protected from exposure, as the server only has a part of the gradient updates.

3) *Federated Averaging:* The server aggregates the gradient updates $G_{A^k}, G_{B^{T^k}}$, and G_{b_k} of the selected clients, and averages them as

$$\overline{G_{A_k}} = \frac{1}{i}\left({G_{A_k}}^{(1)} + {G_{A_k}}^{(2)} + \cdots + {G_{A_k}}^{(i)}\right) \tag{14}$$
$$\overline{G^T_{B_k}} = \frac{1}{c-i}\left(G^{T\ (i+1)}_{B_k} + G^{T\ (i+2)}_{B_k} + \cdots + G^{T\ (c)}_{B_k}\right) \tag{15}$$
$$\overline{G_{b_k}} = \frac{1}{c}\left({G_{b_k}}^{(1)} + {G_{b_k}}^{(2)} + \cdots + {G_{b_k}}^{(c)}\right) \tag{16}$$

where (i) represents the ith client. Then, the gradient updates are used to update the parameters as follows:

$$A_{k+1}^{l} = A_{k}^{l} + \alpha_k \overline{G_{A_k}^{l}} \tag{17}$$

$$B_{k+1}^{l\,T} = B^{l\,T}_{k} + \alpha_k \overline{G_{B_k}^{l\,T}} \tag{18}$$

$$b^{l}{}_{k+1} = b^{l}{}_{k} + \alpha_k \overline{G_{b_k}^{l}}. \tag{19}$$

D. Training Patterns

Due to system heterogeneity, the storage memory of each device in federated learning may differ. Some mobile devices have enough memory to train complex DNN models, while other mobile devices have strict constraints in terms of memory capacity. Furthermore, there is a possibility that devices in federated learning run different models at the same time. For example, we assume that all the mobile devices perform the original DNNs model initially. As resources are occupied more and more by other programs, some devices cannot continue to train the complex DNNs, so they replace the original model with the compression one. Meanwhile, a group of small mobile devices running the compression DNNs model with limited storage resources join the training process. However, some of the original large mobile devices may not update the training model in time or do not want to replace the model as they have gained a well-performed original DNNs model; thus, they still upload the updates of the original DNNs model. Therefore, how to design a unified federated deep learning framework for heterogeneous mobile devices is a challenging problem. We call the devices that are running the original model and the compression one as big devices and small devices, respectively. We present three training patterns in our framework, synchronized pattern, pretrain fine-tuning pattern, and alternative pattern, to integrate the heterogeneous devices running different models. The three patterns are introduced as follows.

1) Synchronized Pattern: The original model and the compression model train together synchronously. A typical round of federated learning in synchronized pattern consists of the following steps.

1) `Local Training`: In each iteration, each big device and each small device simultaneously computes the weight matrix W and the low-rank weight matrices A and B, respectively. Then, the big devices send their W to the server (Fig. 4①), and the small devices send their A and B to the server synchronously (Fig. 4⑥).
2) `Server Process`: The server computes $W^{+} = (A \times B + W)/2$ (Fig. 4④), and then decomposes W^{+} into A^{+} and B^{+} (Fig. 4③).
3) `Update Weight Matrix`: Small devices download A^{+} and B^{+} from the server (Fig. 4⑤), and big devices get W^{+} from the server (Fig. 4②).

This is an intuitive way for federated learning; however, the decomposition of the original model may lose information, thus leading to great drop-in accuracy.

2) Pretrained Fine-Tuning Pattern: The previous works usually use decomposition to approximate the parameters of the pretrained weights and then fine-tune to compensate for the performance loss. According to this idea, we present a pretrained fine-tuning pattern. The process of the pretrained fine-tuning pattern is as follows.

1) `Pretraining on Big Devices`: In each iteration, each big device computes its update matrices W and sends to the server (Fig. 4①). The server aggregates these updates, averages them, and sends back the global improved weight matrix W^{+} to big devices for the next iteration (Fig. 4②). After some loops of (Fig. 4①) and (Fig. 4②), we obtain the global weight W^{+}.
2) `Server Process`: The server decomposes the pretrained weight matrix W^{+} into two lower dimensional matrices A^{+} and B^{+} (Fig. 4③), and then sends them to the small devices (Fig. 4⑤).
3) `Fine-Tuning on Small Devices`: Small devices fine-tune the weight matrices in federated manner to obtain two global weight matrices A^{+} and B^{+}.
4) `Update Weight Matrix`: The server computes $W^{+} = A^{+} \times B^{+}$ (Fig. 4④), and the big devices get W^{+} from the server (Fig. 4②).

3) Alternative Pattern: In alternative pattern, at each iteration, big devices train a global weight matrix first, and then each small devices computes their updates based on the decomposition of the global weight matrix trained by the big devices from the previous round. The process of the alternative pattern is as follows.

1) `Local Training on Big Devices`: In each iteration, each big device computes its weight matrix W and sends it to the server (Fig. 4①).
2) `Server Decomposition`: The server averages the weight matrices to obtain a global weight matrix W^{+}, and then decomposes W^{+} to A^{+} and B^{+} (Fig. 4③).
3) `Local Training on Small Devices`: The small devices download A^{+} and B^{+} from the server (Fig. 4⑤), and compute their gradient updates A and B on their local data, and send them to the server (Fig. 4⑥).
4) `Server Composition`: The server averages A and B of all small devices to get matrices A^{+} and B^{+}, and then computes $W^{+} = A^{+} \times B^{+}$ (Fig. 4④).
5) `Update Weight Matrix`: The big devices download W^{+} for the next iteration.

IV. Experimental Results and Analysis

In this section, we introduce the real-world data sets and report our experimental results.

A. Data Sets

We employ two image classification data sets for the proposed framework, one is the CIFAR-10 data set [44], which is IID, balanced, and have been widely used in DNNs research. The other is federated extended MNIST (FEMNIST) [45], which is non-IID, unbalanced, and large-scale, built by partitioning the data in Extended MNIST [46] based on the writer of the digits, lowercase, and uppercase letters.

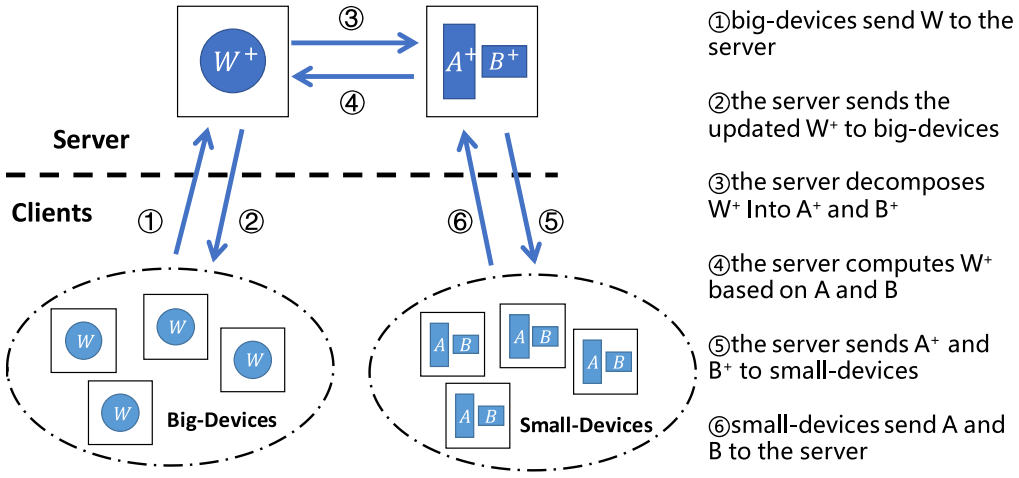

Fig. 4. Process of training pattern: Different training patterns have different processes. The process of the synchronized pattern is ① ⑥ → ④ → ③ → ⑤ ②; the process of the pretrained fine-tuning pattern is ① → ② → ① → ② → ⋯ → ① → ② → ③ → ⑤ → ⑥ → ⑤ → ⑥ → ⋯ → ⑤ → ⑥ → ④ → ②; and the process of the alternative pattern is ① → ③ → ⑤ → ⑥ → ④ → ②.

TABLE II
STATISTICS OF DATA SETS

Dataset	Number of devices	Total samples	Samples per device	
			mean	stdev
CIFAR10	200	50000	250	0
FEMNIST	900	278033	308.93	74.13

1) `CIFAR-10 Data~Set`: The data set consists of ten classes of 32 × 32 images with three RGB channels. There are 50 000 training examples and 10 000 testing examples. We shuffle and partition the training set into 200 subsets, representing 200 clients, while the testing set is globally shared.
2) `FEMNIST Data~Set`: The data set consists of 62 different classes (10 digits, 26 lowercase, and 26 uppercase), and images are 28 by 28 pixels. We generate 900 clients from FEMNIST for training, and randomly select 10 000 images as the testing set. The amount of local training data typically varies across devices. Besides, the local data of any device cannot be regarded as a representative sample of the overall distribution.

The statistics of these data sets are listed in Table II.

B. Compared Methods

Training Patterns: In this article, we present three different patterns to train the heterogeneous mobile devices in Section III-D.

1) `BOTH`: Our synchronized pattern mentioned in Section III-D1 is named *BOTH*. In each iteration, the original model and the compression model are updated synchronously.
2) `BFIRST`: Our pretrained fine-tuning pattern mentioned in Section III-D2 is named *BFIRST*. The big devices pretrain a model collaboratively through multiple iterations. Then, the small devices fine-tune the pretrained model.
3) `SWITCH`: Our alternative pattern mentioned in Section III-D3 is named *SWITCH*. In each iteration, each big device computes its updates first, and then each small device computes its updates based on the updates of the big devices from the previous round.

Federated Upload Pattern: We proposed a novel CEFO approach to reduce the communication cost further for the small devices in Section III-C. We consider the following two upload patterns to compare.

1) `FULL`: The small-devices send all their updates to the server in each iteration.
2) `HALF`: Different from *FULL*, half of the small-devices send G_{A^k} and G_{b^k}, and the rest send $G_{B_k^T}, G_{b^k}$ to the server at iteration k.

Systems Heterogeneity: Due to system heterogeneity, there is a possibility that devices in federated learning run different models at the same time. We simulate the coexistence of the two models in a real-world scenario by adjusting the proportion of big devices. We set the proportion of big devices to be $\{0.0, 0.2, 0.5, 0.7, 1.0\}$, respectively. We represent our method in the following form: ⟨*big-devices proportion*⟩-⟨*training pattern* ⟩-⟨*upload pattern* ⟩. For example, the method *1.0-BOTH-FULL* means that the federated learning system consists of 100% big devices, the devices train in the *BOTH* pattern, and use the *FULL* federated upload pattern. *1.0-BOTH-FULL* is the same as the federated averaging algorithm [14], so we will take it as a baseline method.

C. Experimental Results

To evaluate the advantages of our framework, we take CNNs, one of the most popular DNNs, as an instance. We deploy experiments on a widely used CNNs, AlexNet [47], which contains five convolutional layers and three FC layers. For other DNNs, which include FC layers, such as VGGNet [48], our proposed method is also workable.

In each iteration, we randomly sample 10% big devices and 10% small devices to participate in the training. It simulates

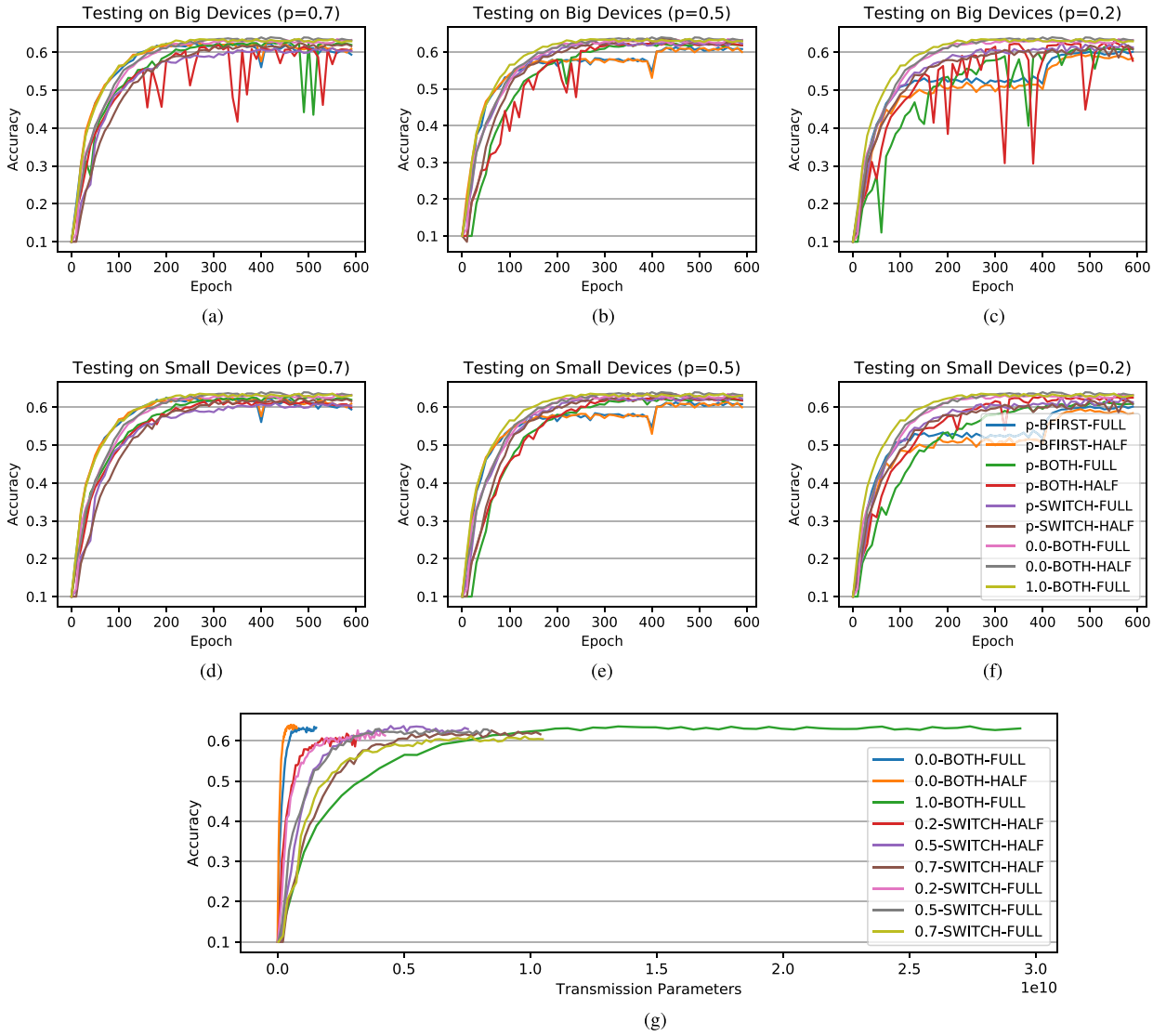

Fig. 5. Results on CIFAR-10 data set. (a)–(f) Convergence speed in terms of iterations, and (g) convergence speed in terms of the total number of uploaded parameters. "Testing on Big-Devices" and "Testing on Small-Devices" represent the testing results on big-devices and small-devices separately. p represents the proportion of big devices.

the real-world situation that devices will only participate in a small number of update rounds per day due to the limitation of battery and network. In such settings, the learning process would utilize the data of only a small number of devices per round. Thus, applying federated learning is possible without affecting the user experience.

From a general view, Figs. 5 and 6 show results on the CIFAR-10 and FMINIST data sets, respectively. Figs. 5(a)–(f) and 6(a)–(f) show convergence speed in terms of iterations, while Figs. 5(g) and 6(g) show convergence speed in terms of the total number of uploaded parameters. The figures of "Testing on Big Devices" and "Testing on Small Devices" show the testing results on big devices and small devices separately. For *1.0-BOTH-FULL*, due to the lack of the compression model, the original model needs to be decomposed into lower dimensional models so that it can be deployed on small devices for testing. On the contrary, for *0.0-BOTH-FULL* and *0.0-BOTH-HALF*, the compression model should be composed for testing on big devices. In the following, we will analyze the impact of the compression model, training patterns, federated upload patterns, and the proportion of big devices on the performance of our framework.

1) Compression Mode Versus Original Model: We first focus on the comparison between pure original mode (denoted as *1.0-BOTH-FULL*) and pure compression mode (denoted as *0.0-BOTH-FULL*). The goal of compression mode is to address the resource-constrained issues of small devices while maintaining the model's performance. In Fig. 5(a) and (d), we observe that the prediction accuracy of *0.0-BOTH-FULL* is almost closed to *1.0-BOTH-FULL* when testing on CIFAR-10. Fig. 6(d) also shows that *0.0-BOTH-FULL* has little or no drop in accuracy on small devices compared with *1.0-BOTH-FULL* when testing on the FEMNIST data set. Furthermore, Fig. 6(a) shows that *0.0-BOTH-FULL* can achieve a higher accuracy compared to *0.0-BOTH-FULL* when testing on

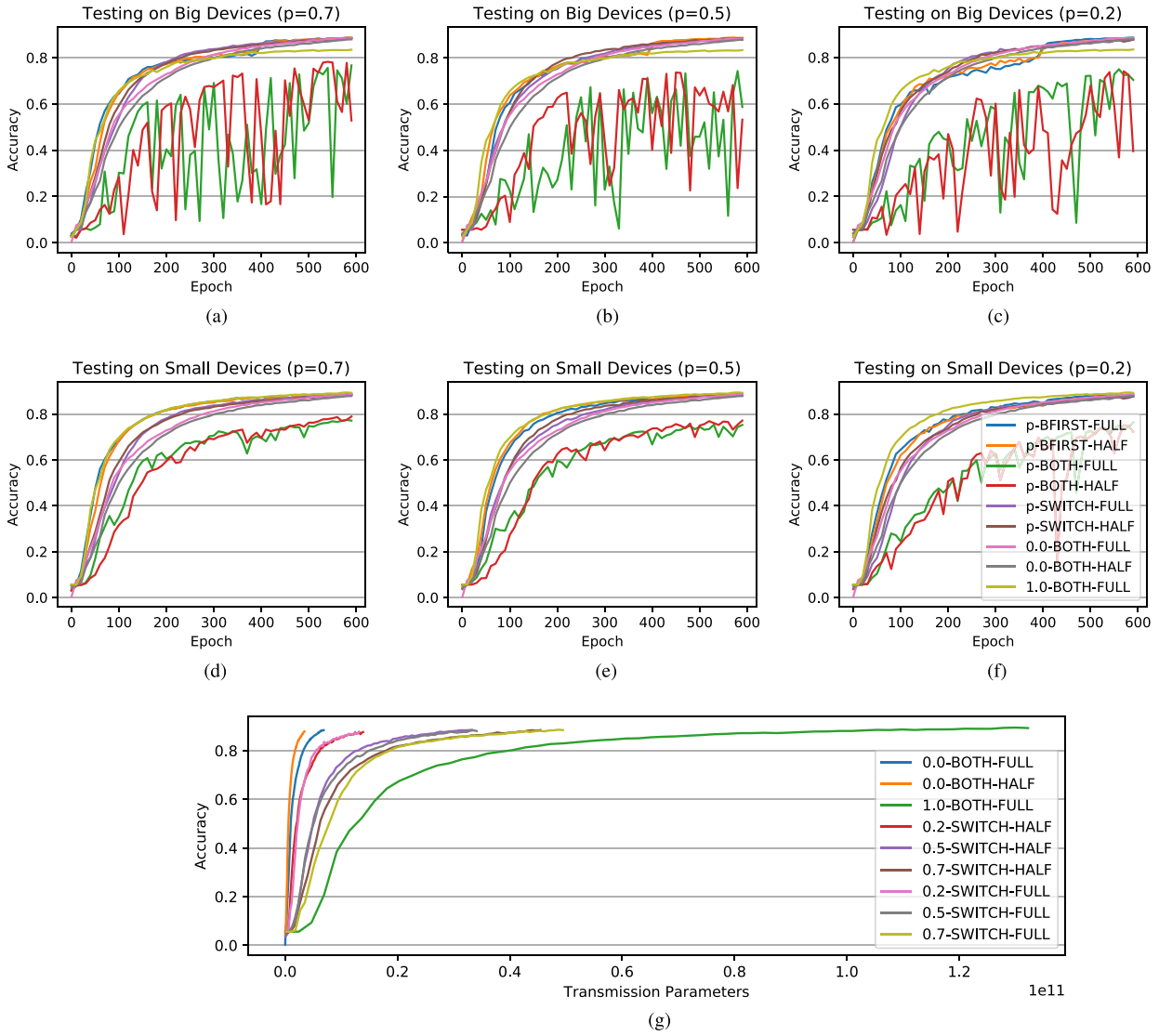

Fig. 6. Results on FEMNIST data set. (a)–(f) Convergence speed in terms of iterations, and (g) convergence speed in terms of the total number of uploaded parameters. "Testing on Big-Devices" and "Testing on Small-Devices" represent the testing results on big devices and small devices separately. p represents the proportion of big devices.

big devices. This is because the redundancy of parameters in *1.0-BOTH-FULL* leads to overfitting when testing on the federated scenario, which contains non-IID, unbalanced, and large-scale data. Our proposed compression model (*0.0-BOTH-FULL*) is more suitable for the federated tasks. In addition, in Figs. 5(g) and 6(g), we observe that the compression model can achieve significant reductions in network communication cost on both CIFAR-10 and FEMNIST.

Overall, the experimental results show that the compression approach can achieve significant reductions in model size without any accuracy loss and can even improve accuracy in the federated data set. Practically, this reduction in the number of parameters is important for mobile devices with limited storage memory.

2) HALF Versus FULL: Then, we compare the two federated upload patterns. From Figs. 5(a)–(f) and 6(a)–(f), the lines of the *HALF* methods and the *FULL* methods of the same training pattern roughly coincide; thus, it is clear that the *HALF* methods can obtain the same accuracy as the *FULL* methods and even outperform the *FULL* methods without a drop in convergence. Besides, Figs. 5(g) and 6(g) show that the *HALF* methods can reduce the communication cost further than the *FULL* methods within the same training pattern.

In summary, CEFO can reduce the communication cost further with little or no drop in accuracy and convergence. This attribute is beneficial for federated learning as not only communication cost can be reduced further but also update gradients can be protected from exposure.

3) SWITCH Versus BOTH BFIRST: We compare three different training patterns. From Figs. 5 and 6, we observe that the performances of the *0.2-BOTH*, *0.4-BOTH*, and *0.7-BOTH* patterns are poor. This is because the decomposition loss happens at every iteration, and cause a serious drop in accuracy. Also, we observe that the *BFIRST* pattern and *SWITCH* pattern achieve relatively better performance.

TABLE III
ACCURACY ON CIFAR-10 DATA SET

Methods	proportion of big-devices									
	100%		70%		50%		20%		0%	
	B-ACC	S-ACC	B-ACC	S-ACC	B-ACC	S-ACC	B-ACC	S-ACC	B-ACC	S-ACC
BFIRST-FULL	—	—	0.6292	0.6292	0.6195	0.6195	0.6029	0.6029	—	—
BFIRST HALF	—	—	**0.6336**	**0.6336**	0.6165	0.6165	0.5959	0.5959	—	—
BOTH FULL	0.6383	0.6391	0.6321	0.6314	0.6272	0.6275	0.6173	0.6183	0.6376	0.6376
BOTH HALF	—	—	0.6215	0.6206	0.6282	0.6286	0.629	**0.6313**	0.6419	0.6419
SWITCH FULL	—	—	0.6164	0.6164	0.6309	0.6309	**0.6295**	0.6295	—	—
SWITCH HALF	—	—	0.6258	0.6261	**0.6444**	**0.6444**	0.6218	0.6218	—	—

TABLE IV
ACCURACY ON FEMNIST DATA SET

Methods	proportion of big-devices									
	100%		70%		50%		20%		0%	
	B-ACC	S-ACC	B-ACC	S-ACC	B-ACC	S-ACC	B-ACC	S-ACC	B-ACC	S-ACC
BFIRST-FULL	—	—	**0.8879**	**0.8879**	0.8883	0.8883	**0.8892**	**0.8892**	—	—
BFIRST HALF	—	—	0.8869	0.8869	**0.8907**	**0.8907**	0.8832	0.8832	—	—
BOTH FULL	0.8368	0.8946	0.7696	0.7856	0.7634	0.7651	0.7609	0.7648	0.8856	0.8856
BOTH HALF	—	—	0.7899	0.7923	0.7645	0.7788	0.7457	0.7529	0.8807	0.8807
SWITCH FULL	—	—	0.8855	0.8858	0.8838	0.8837	0.8817	0.8817	—	—
SWITCH HALF	—	—	0.8848	0.8848	0.8874	0.8874	0.8818	0.8818	—	—

From the figures of "Testing on Big Devices," there is a leap around the 400th epoch of the line of *BFIRST* pattern. This leap explains that the performance can be improved rapidly via the fine-tuning of the compression model. Besides, from Figs. 5(g) and 6(g), the *SWITCH* pattern reduces the communication cost on both CIFAR-10 and FEMNIST significantly.

In a word, among these three patterns, *SWITCH* achieves communication efficiency and effectiveness superiority.

4) Proportion of Big Devices: In this section, we discuss the influence of the proportion of big devices. We display the accuracy of different methods under various proportions of big devices in Tables III and IV for two data sets. First, we observe that our methods *BFIRST-FULL*, *BFIRST-HALF*, *SWITCH-FULL*, and *SWITCH-HALF* are efficient under all the proportion of big devices. If a mobile device replaces its original model with the compression model, the federated learning framework will continue to operate steadily without significantly reducing the prediction accuracy. In such settings, we can gradually replace the original model in the federated framework with the compression model. Alternatively, leave the original model unchanged and let the new devices added to the framework run the compression model.

V. CONCLUSION AND FUTUREWORK

Federated learning on mobile devices is not a trivial task, which has three significant challenges: 1) resource constrained; 2) high communication overhead; and 3) systems heterogeneity. To address these challenges, we propose a unified DNNs federated framework. First, we present a compression DNNs model for resource-constrained mobile devices, which uses a low-rank representation of the weight matrices of the FC layers. Second, we propose a novel CEFO approach, which only sends a part of the updates of each client to the server. Third, we present three different patterns to federated train the heterogeneous mobile devices. We test the performance of the proposed framework on CIFAR-10 and FEMNIST, and the experimental results show that our framework can achieve significant effectiveness. To our best knowledge, we are the first to integrate the heterogeneous devices running different models in federated learning. Under any proportion of different models, our federated learning framework will operate steadily without significantly reducing the prediction accuracy.

REFERENCES

[1] J. Kang, Z. Xiong, D. Niyato, Y. Zou, Y. Zhang, and M. Guizani, "Reliable federated learning for mobile networks," *IEEE Wireless Commun.*, vol. 27, no. 2, pp. 72–80, Apr. 2020.

[2] J. Konečný, H. B. McMahan, D. Ramage, and P. Richtárik, "Federated optimization: Distributed machine learning for on-device intelligence," 2016. [Online]. Available: https://arxiv.org/abs/1610.02527.

[3] J. Konečný, "Stochastic, distributed and federated optimization for machine learning," 2017. [Online]. Available: http://arxiv.org/abs/1707.01155.

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[5] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *Proc. 13th Eur. Conf. Comput. Vis. (ECCV)*, Zurich, Switzerland, Sep. 2014, pp. 818–833.

[6] X. Xu *et al.*, "Scaling for edge inference of deep neural networks," *Nat. Electron.*, vol. 1, no. 4, pp. 216–222, 2018.

[7] A. F. Aji and K. Heafield, "Sparse communication for distributed gradient descent," in *Proc. Conf. Empir. Methods Nat. Lang. Process. (EMNLP)*, Copenhagen, Denmark, Sep. 2017, pp. 440–445.

[8] Y. Lin, S. Han, H. Mao, Y. Wang, and B. Dally, "Deep gradient compression: Reducing the communication bandwidth for distributed training," in *Proc. 6th Int. Conf. Learn. Represent. (ICLR)*, Vancouver, BC, Canada, 2018, pp. 1–14.

[9] W. Wen *et al.*, "Terngrad: Ternary gradients to reduce communication in distributed deep learning," in *Proc. 31st Annu. Conf. Neural Inf. Process. Syst.*, Long Beach, CA, USA, Dec. 2017, pp. 1509–1519.

[10] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," 2015. [Online]. Available: arXiv:1510.00149.

[11] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Salt Lake City, UT, USA, Jun. 2018, pp. 4510–4520.

[12] N. Ma, X. Zhang, H. Zheng, and J. Sun, "ShuffleNet V2: Practical guidelines for efficient CNN architecture design," in *Proc. 15th Eur. Conf. Comput. Vis. (ECCV)*, Munich, Germany, 2018, pp. 122–138.

[13] J. Konecný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," 2016. [Online]. Available: https://arxiv.org/abs/1610.05492.

[14] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. Y. Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. 20th Int. Conf. Artif. Intell. Stat. (AISTATS)*, Fort Lauderdale, FL, USA, Apr. 2017, pp. 1273–1282.

[15] D. Chai, L. Wang, K. Chen, and Q. Yang, "Secure federated matrix factorization," 2019. [Online]. Available: https://arxiv.org/abs/1906.05108.

[16] K. Bonawitz *et al.*, "Practical secure aggregation for privacy-preserving machine learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security (CCS)*, Dallas, TX, USA, 2017, pp. 1175–1191.

[17] W. Y. B. Lim *et al.*, "Federated learning in mobile edge networks: A comprehensive survey," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 3, pp. 2031–2063, 3rd Quart., 2020.

[18] T. Nishio and R. Yonetani, "Client selection for federated learning with heterogeneous resources in mobile edge," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Shanghai, China, May 2019, pp. 1–7.

[19] J. Kang, Z. Xiong, D. Niyato, S. Xie, and J. Zhang, "Incentive mechanism for reliable federated learning: A joint optimization approach to combining reputation and contract theory," *IEEE Internet Things J.*, vol. 6, no. 6, pp. 10700–10714, Dec. 2019.

[20] S. Wang *et al.*, "Adaptive federated learning in resource constrained edge computing systems," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 6, pp. 1205–1221, Jun. 2019.

[21] V.-D. Nguyen, S. K. Sharma, T. X. Vu, S. Chatzinotas, and B. E. Ottersten, "Efficient federated learning Algorithm for resource allocation in wireless IoT networks," *IEEE Internet Things J.*, vol. 8, no. 5, pp. 3394–3409, Mar. 2021.

[22] A. M. Abdelmoniem and M. Canini, "Towards mitigating device heterogeneity in federated learning via adaptive model quantization," in *Proc. 1st Workshop Mach. Learn. Syst. Virtual Event (EuroMLSysEuroSys)*, Edinburgh, U.K., Apr. 2021, pp. 96–103.

[23] C. Yang, Q. Wang, M. Xu, S. Wang, K. Bian, and X. Liu, "Characterizing impacts of heterogeneity in federated learning upon large-scale smartphone data," 2020. [Online]. Available: https://arxiv.org/abs/2006.06983.

[24] H. Lan *et al.*, "FeatherCNN: Fast inference computation with TensorgEMM on ARM architectures," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 3, pp. 580–594, Mar. 2020.

[25] M. Xu, J. Liu, Y. Liu, F. X. Lin, Y. Liu, and X. Liu, "A first look at deep learning apps on smartphones," in *Proc. World Wide Web Conf. (WWW)*, San Francisco, CA, USA, May 2019, pp. 2125–2136.

[26] Y. Kim, E. Park, S. Yoo, T. Choi, L. Yang, and D. Shin, "Compression of deep convolutional neural networks for fast and low power mobile applications," in *Proc. 4th Int. Conf. Learn. Represent. (ICLR)*, San Juan, Puerto Rico, May 2016, pp. 1–16.

[27] A. G. Howard *et al.*, "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017. [Online]. Available: https://arxiv.org/abs/1704.04861.

[28] Y. Gong, L. Liu, M. Yang, and L. D. Bourdev, "Compressing deep convolutional networks using vector quantization," 2014. [Online]. Available: https://arxiv.org/abs/1412.6115.

[29] W. Chen, J. T. Wilson, S. Tyree, K. Q. Weinberger, and Y. Chen, "Compressing neural networks with the hashing trick," in *Proc. 32nd Int. Conf. Mach. Learn. (ICML)*, Lille, France, Jul. 2015, pp. 2285–2294.

[30] Y. Cheng, F. X. Yu, R. S. Feris, S. Kumar, A. N. Choudhary, and S. Chang, "Fast neural networks with circulant projections," 2015. [Online]. Available: arXiv:1502.03436.

[31] S. Srinivas and R. V. Babu, "Data-free parameter pruning for deep neural networks," in *Proc. Brit. Mach. Vis. Conf. (BMVC)*, Swansea, U.K., Sep. 2015, pp. 1–12.

[32] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *Proc. Annu. Conf. Neural Inf. Process. Syst.*, Barcelona, Spain, Dec. 2016, pp. 2074–2082.

[33] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, "Exploiting linear structure within convolutional networks for efficient evaluation," in *Proc. Annu. Conf. Neural Inf. Process. Syst.*, Montreal, QC, Canada, Dec. 2014, pp. 1269–1277.

[34] A. Novikov, D. Podoprikhin, A. Osokin, and D. P. Vetrov, "Tensorizing neural networks," in *Proc. Annu. Conf. Neural Inf. Process. Syst.*, Montreal, QC, Canada, Dec. 2015, pp. 442–450.

[35] C. Tai, T. Xiao, X. Wang, and W. E, "Convolutional neural networks with low-rank regularization," in *Proc. 4th Int. Conf. Learn. Represent. (ICLR)*, San Juan, Puerto Rico, May 2016, pp. 1–11.

[36] C. Xie, S. Koyejo, and I. Gupta, "Asynchronous federated optimization," 2019. [Online]. Available: http://arxiv.org/abs/1903.03934.

[37] F. Seide, H. Fu, J. Droppo, G. Li, and D. Yu, "1-bit stochastic gradient descent and application to data-parallel distributed training of speech DNNs," in *Proc. 15th Annu. Conf. Int. Speech Commun. Assoc. (INTERSPEECH)*, Singapore, Sep. 2014, pp. 1058–1062.

[38] S. Zhou, Z. Ni, X. Zhou, H. Wen, Y. Wu, and Y. Zou, "DoReFa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients," 2016. [Online]. Available: https://arxiv.org/abs/1606.06160.

[39] N. Strom, "Scalable distributed DNN training using commodity GPU cloud computing," in *Proc. 16th Annu. Conf. Int. Speech Commun. Assoc. (INTERSPEECH)*, Dresden, Germany, Sep. 2015, pp. 1488–1492.

[40] M. Denil, B. Shakibi, L. Dinh, M. Ranzato, and N. de Freitas, "Predicting parameters in deep learning," in *Proc. 27th Annu. Conf. Neural Inf. Process. Syst.*, Dec. 2013, pp. 2148–2156.

[41] T. N. Sainath, B. Kingsbury, V. Sindhwani, E. Arisoy, and B. Ramabhadran, "Low-rank matrix factorization for deep neural network training with high-dimensional output targets," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process. (ICASSP)*, Vancouver, BC, Canada, May 2013, pp. 6655–6659.

[42] J. Xue, J. Li, and Y. Gong, "Restructuring of deep neural network acoustic models with singular value decomposition," in *Proc. 14th Annu. Conf. Int. Speech Commun. Assoc. (INTERSPEECH)*, Lyon, France, Aug. 2013, pp. 2365–2369.

[43] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. 3rd Int. Conf. Learn. Represent. (ICLR)*, San Diego, CA, USA, May 2015, pp. 1–14.

[44] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Dept. Comput. Sci., Univ. Toronto, Toronto, ON, Canada, Rep. TR-2009, 2009.

[45] S. Caldas *et al.*, "LEAF: A benchmark for federated settings," 2018. [Online]. Available: https://arxiv.org/abs/1812.01097.

[46] G. Cohen, S. Afshar, J. Tapson, and A. van Schaik, "EMNIST: An extension of MNIST to handwritten letters," 2017. [Online]. Available: https://arxiv.org/abs/1702.05373.

[47] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. 26th Annu. Conf. Neural Inf. Process. Syst.*, Dec. 2012, pp. 1106–1114.

[48] L. Wang, S. Guo, W. Huang, and Y. Qiao, "Places205-VGGNet models for scene recognition," 2015. [Online]. Available: https://arxiv.org/abs/1508.01667.

Xiaoli Li (Student Member, IEEE) received the master's degree in computer architecture from the University of Electronic Science and Technology of China, Chengdou, China, in 2011. She is currently pursuing the Ph.D. degree with the School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou, China.

Her research interests include services computing, software engineering, cloud computing, machine learning, and federated learning.

Yuzheng Li received the B.E. degree from the Sun Yat-sen University, Guangdong, China, in 2018, where he is currently pursuing the M.S. degree.

His current research interests include federated learning, blockchain, statistical machine learning, multiview learning, and optimization.

Shixuan Li received the bachelor's degree in network engineering from Sun Yat-sen University, Guangzhou, China, in 2019, where she is currently pursuing the master's degree with the School of Computer Science and Engineering.

Her research interests include machine learning, federated learning, and software engineering.

Yuren Zhou (Member, IEEE) received the B.Sc. degree in mathematics from Peking University, Beijing, China, in 1988, and the M.Sc. degree in mathematics and the Ph.D. degree in computer science from Wuhan University, Wuhan, China, in 1991 and 2003, respectively.

He is currently a Professor with the School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou, China. His current research interests include design and analysis of algorithms, evolutionary computation, and social networks.

Chuan Chen (Member, IEEE) received the B.S. degree from Sun Yat-sen University, Guangzhou, China, in 2012, and the Ph.D. degree from Hong Kong Baptist University, Hong Kong, in 2016.

He is currently an Associate Professor with the School of Computer Science and Engineering, Sun Yat-sen University. He published over 50 international journal and conference papers. His current research interests include machine learning, numerical linear algebra, and numerical optimization.

Zibin Zheng (Senior Member, IEEE) received the Ph.D. degree from the Chinese University of Hong Kong, Hong Kong, in 2011.

He is currently a Professor with the School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou, China. He has published over 150 international journal and conference papers, including three ESI highly cited papers. According to Google Scholar, his papers have more than 13 590 citations, with an H-index of 54. His research interests include blockchain, smart contract, services computing, and software reliability.