BOSE: Block-Wise Federated Learning in Heterogeneous Edge Computing

Lun Wang, Yang Xu, *Member, IEEE*, Hongli Xu, *Member, IEEE*, Zhida Jiang, Min Chen, *Student Member, IEEE*, Wuyang Zhang, *Member, IEEE*, and Chen Qian, *Senior Member, IEEE, Member, ACM*

***Abstract*— At the network edge, federated learning (FL) has gained attention as a promising approach for training deep learning (DL) models collaboratively across a large number of devices while preserving user privacy. However, FL still faces specific challenges related to the limited, heterogeneous and dynamic resources of devices. In most FL systems, all devices train the same model, while the devices with constrained resources, referred to as stragglers, will significantly slow down overall training process. It is intuitive to alleviate computation and communication load on the stragglers by training and transmitting a part of the model. Inspired by multi-exit models, we divide an original DL model into several non-overlapping blocks, which can be trained separately on the low-capability devices. Furthermore, we propose BOSE, a novel FL system that performs adaptive block-wise model training under resource constraints. Considering the diverse impacts of different blocks on model convergence and the varying training loads they incur, a naive block assignment strategy, *e.g.*, uniformly random assignment, may not yield optimal model performance and fail to fully utilize available resources. To this end, we introduce two metrics, including *learning speed* and *device-wise divergence*, to measure the potential of blocks in promoting model convergence. Given resource budget, BOSE initially identifies a set of candidate blocks for each device and subsequently selects specific training blocks based on their potential for promoting model convergence. In general, blocks with higher potential are more likely to be chosen for training. Extensive experiments on a physical platform show that BOSE provides a 1.4×∼3.8× speedup without sacrificing model accuracy, compared to the baselines.**

Manuscript received 22 November 2022; revised 11 May 2023; accepted 8 September 2023; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor C. Joe-Wong. Date of publication 5 October 2023; date of current version 18 April 2024. This work was supported in part by the National Key Research and Development Program of China under Grant 2021YFB3301500; in part by the National Science Foundation of China (NSFC) under Grant 62132019, Grant 61936015, and Grant 62102391; and in part by the Jiangsu Province Science Foundation for Youths under Grant BK20210122. *(Corresponding author: Hongli Xu.)*

Lun Wang, Yang Xu, Hongli Xu, and Zhida Jiang are with the School of Computer Science and Technology, University of Science and Technology of China, Hefei, Anhui 230027, China, and also with the Suzhou Institute for Advanced Research, University of Science and Technology of China, Suzhou, Jiangsu 215123, China (e-mail: wanglun0@mail.ustc.edu.cn; xuyangcs@ustc.edu.cn; xuhongli@ustc.edu.cn; zdjiang@mail.ustc.edu.cn).

Min Chen is with Huawei Cloud Computing Technology Company Ltd., Hangzhou 310052, China (e-mail: chenmin148@huawei.com).

Wuyang Zhang is with Meta Inc., Menlo Park, CA 94025 USA (e-mail: wuyangz@meta.com).

Chen Qian is with the Department of Computer Science and Engineering, University of California at Santa Cruz, Santa Cruz, CA 95064 USA (e-mail: cqian12@ucsc.edu).

This article has supplementary downloadable material available at https://doi.org/10.1109/TNET-XU-3316421, provided by the authors.

Digital Object Identifier 10.1109/TNET.2023.3316421

***Index Terms*— Edge computing, federated learning, heterogeneous devices, block-wise training.**

I. INTRODUCTION

WITH the rapid development of edge computing in the past few years [1], [2], a great number of computation tasks are pushed from the cloud to the network edge, which avoids massive data transmission across the Internet. In edge computing, federated learning (FL) [3] has become an indispensable training paradigm for efficient on-device model training without exposing sensitive raw data. For example, FL has been applied for predicting clinical outcomes in patients with COVID-19 [4] and keyword spotting of the voice assistant on smartphones [5]. In FL, edge devices are coordinated by a parameter server to perform model training on their local datasets. By aggregating local models periodically, the parameter server obtains an updated global model and then distributes it to devices for further training.

Despite the distinct advantages of FL, several characteristics of on-device resources hinder FL systems from performing efficient model training. 1) *Limitation.* Different from the resource-rich data centers [6], the capabilities (including computation and communication) of edge devices are typically constrained. Limited resources on devices may result in long latency for training over-parameterized DL models. 2) *Heterogeneity.* Edge devices usually exhibit capability heterogeneity, due to different types of CPU, GPU and RAM [7]. The low-capability devices may become the system stragglers, which slows down the training process. 3) *Dynamics.* There are usually multiple applications running on a device and they may compete for various resources. Besides, the network conditions may also vary due to the mobility of devices. As a result, the capability gap is further widened among heterogeneous devices.

Many works have made efforts to improve training efficiency in FL. To save communication cost, model compression techniques [8], [9], [10] are employed. For example, *Quantization* [9] reduces the number of bits of each model parameter (*e.g.*, from float32 to float16) while *Sparsification* [10] transmits a sparse vector including only a subset of the original model parameters. However, these methods only focus on communication savings but overlook computation efficiency. To fill the gap, some works [11], [12] propose to reduce the complexity of local models on devices to save both computation and communication cost. For example, ProgFed [11] divides a model into several partitions and progressively

expands the training model from the first partition to the full model. As all devices train the same model at one time, ProgFed fails to deal with resource heterogeneity. Given a specific resource budget, HeteroFL [12] always assigns the same submodel for devices without considering the convergence statuses of different parts in the global model. Thus, it is a waste of resources to train those converged submodels. Besides, through the structured pruning, Hermes [13] first finds sparse structures of local models for different devices and then trains the pruned models with fixed structures, which lacks the flexibility to adapt to resource dynamics. **As a result, none of the existing methods can adequately address all the above three challenges.**

To conquer these challenges, herein, we allow edge devices to adaptively train different parts of a model. Inspired by recent researches on multi-exit models [14], we divide a conventional DL model into several non-overlapping blocks by adding auxiliary classifiers (shown in Figure 2). Each block contains one or multiple consecutive layers, and can be trained independently. The multi-exit model is originally used to save inference cost [15], but its potential to achieve training efficiency in FL remains rarely explored. In this paper, we propose a novel FL system, called BOSE, to perform adaptive ***blo***ck-wi***se*** model training on edge devices. Specifically, we adaptively assign different blocks for heterogeneous devices to perform local training, which aims to alleviate the training cost on the relatively resource-constrained devices as well as promote model convergence.

However, there are two critical difficulties on the way to building an effective and efficient block assignment strategy: 1) The computation load and parameter size of different layers in a model are not the same, and thus those of blocks are also distinct. For example, for a VGG model [16], the computation load and parameter size of the second layer are about 8× and 20× as that of the first layer [17], respectively. Thus, instead of only considering the number of blocks, our system will assign blocks based on their specific training and transmission cost on heterogeneous devices. 2) The convergence statuses of different blocks in a model are not always the same. On one hand, different blocks converge at inconsistent speeds [18]. The low-speed blocks indicate that they are close to the optimum, and training these blocks makes little improvement on model performance [19]. On the other hand, the divergence degree of local gradients is different across blocks [20]. For the blocks whose derived gradients are similar, aggregating blocks from a few devices is enough for the global model. On the contrary, for the divergent blocks, we have to collect local updating from more devices to obtain the high-quality global updating. Due to the different convergence statuses across blocks, it is unwise to spend the same resources on training and transmitting different blocks.

The main contributions of this paper are as follows:

- We propose an efficient FL system, where a full model is split into several blocks through auxiliary classifiers, and devices can perform block-wise training under resource constraints.
- To achieve efficient training, we propose two metrics (*i.e.*, learning speed and device-wise divergence) to measure the blocks' potential for enhancing model convergence. Considering both devices' resources and blocks' potential, our system will adaptively assign different blocks for heterogeneous devices.
- We implement our system on a physical platform with 50 devices and perform extensive evaluations. The results demonstrate the efficiency of BOSE, which provides a 1.4×∼3.8× speedup when achieving the same model accuracy, compared with other baselines.

The rest of the paper is organized as follows. Section II introduces the background of FL and the motivation of our research. Section III presents the design of our system. Convergence of our training method is given in Section IV. We perform extensive experiments in Section V to evaluate our system. Section VI reviews related works and the whole paper is concluded in Section VII.

II. Background and Motivation

In this section, we first briefly introduce federated learning (FL), and then analyze several challenges in FL. Furthermore, we present the motivation to utilize multi-exit models in our FL system.

A. Federated Learning

An FL system usually includes a parameter server and a set of M edge devices (*e.g.*, IoT clients and personal phones) $\mathcal{V} = \{v_1, v_2, \ldots, v_M\}$, which collaboratively train DL models over widespread locations. Each device $v_m \in \mathcal{V}$ trains a local model on its own private dataset $\mathcal{D}_m$ with $|\mathcal{D}_m| = n_m$, and only needs to synchronize the updated local models with the parameter server rather than sharing original data, which prevents the exposure of personal privacy.

Let $l(w, x, y)$ denote the loss function over the data (x, y), where w is the model, x is the data sample and y is the label. For device v_m, its optimization objective is:

$$F_m(w_m) = \mathbb{E}_{(x,y)\sim\mathcal{D}_m} l(w_m, x, y), \tag{1}$$

where w_m is the local model of device v_m. The global optimization objective can be formulated as:

$$\min_w F(w) := \min_w \sum_{m=1}^{M} \frac{n_m}{n} F_m(w), \tag{2}$$

where $n = \sum_{m=1}^{M} n_m$ is the total number of data samples and $w = \frac{1}{n}\sum_{m=1}^{M} n_m w_m$ is the global model.

B. Challenges of FL on Edge Devices

Although FL is gaining increasing attention, the unique characteristics of resources on edge devices give rise to three major challenges for implementing efficient FL systems.

Limitation. As a DL model usually contains a large number of parameters (*e.g.*, VGG [16] and ResNet [21] with over hundreds of millions of parameters), training these models remains a resource-hungry task, resulting in over 10^6s training time on Raspberry Pi devices [22]. Unlike data centers, where dedicated GPUs and networks are deployed with a stable power supply [6], edge devices are usually equipped with limited computation resources and battery life due to the hardware restriction. Besides, massive edge devices communicate with the parameter server through edge networks and the typical bandwidth is only about 5∼25Mb/s [3], which is much less than that within data centers (*e.g.*, over 10Gb/s [6]). Consequently, limited resources on devices lead to significant latency for training DL models.

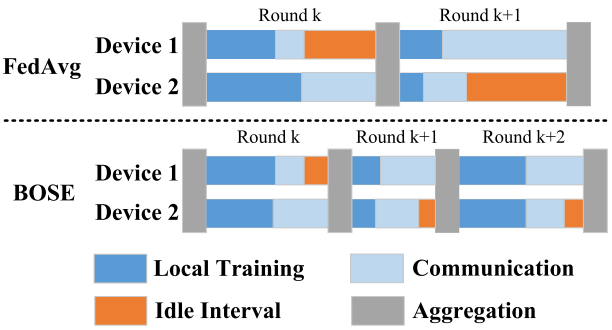


Fig. 1. Illustration of FedAvg and BOSE.

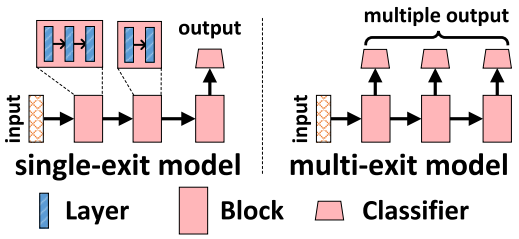


Fig. 2. Architectures of the conventional (*i.e.*, single-exit) model and the multi-exit model.

Heterogeneity. The capabilities of edge devices usually exhibit great divergence in both computation and communication [7], [23], [24]. According to AI-Benchmark,[1] the computation capability varies significantly across different mobile CPUs. For instance, compared with Snapdragon 625, HiSilicon Kirin 9000 can speed up the model training by about 10×. Besides, according to the report of Cisco [25], by 2023, there are still about 30% of edge devices that communicate through 2G or 3G (the bandwidth is less than 10 Mb/s). Meanwhile, the 5G connection, which is over 50 times faster than 2G or 3G connections, accounts for 10%. Due to the computation and communication heterogeneity among devices, the stragglers may slow down the training process, resulting in a long waiting time of fast devices and thus inefficient utilization of available resources.

Dynamics. Since there are usually multiple applications running on an edge device [26] and they may compete for resources, the available resources for FL could change over time. For instance, when FL tasks are executed in the background, the user may play handy games, occupying computation capacity, or watch online videos, occupying bandwidth. Moreover, network conditions will change occasionally due to the mobility of devices and the obstacles between the transceivers [27]. As a result, the available resources on devices are time-varying, which widens the performance gap among heterogeneous edge devices.

In Figure 1, we illustrate how these challenges hinder previous methods (*e.g.*, FedAvg) from performing efficient FL, and show the improvement by our system. In FedAvg, since all devices train the same local models across different rounds, the fast devices always have to wait for the stragglers, resulting in long idle time and resource waste. Moreover, due to resource dynamics, the fast device (*e.g.*, device 1) at round k may become the straggler at the next round. Thus, an efficient FL system should consider the heterogeneity among devices as well as resource dynamics over time. To achieve this goal, BOSE will assign different training loads for devices based on their current capabilities, so as to fully utilize available resources.

C. Multi-Exit Models

Multi-exit models have been proven effective to accelerate inference [15], but remain rarely explored for accelerating model training. Herein, we first introduce the architecture of the multi-exit model, consisting of multiple blocks, and then show the benefits of training blocks on resource-constrained devices. As a supplement, we finally introduce the methods to achieve inference efficiency using multi-exit models.

Multi-exit Model Architecture. A conventional model can be regarded as a function f that maps the input x into the prediction $f(x)$. We decouple it into B blocks and a classifier, which are represented as $h_1, \ldots, h_B$ and c, respectively:

$$\hat{y} = f(x) = (c \circ h_B \circ h_{B-1} \circ \cdots \circ h_1)(x), \tag{3}$$

where x is the input and $\hat{y}$ is the prediction from the model. The operator $\circ$ denotes composition, *i.e.*, $(h_b \circ h_{b'})(\cdot) = h_b(h_{b'}(\cdot))$. The output of block b is denoted as $z_b, 1 \leq b \leq B$. We denote $z_0 = x$, $z_b = h_b(z_{b-1})$ and $\hat{y} = c(z_B)$. As shown in Figure 2, a block contains any single or several consecutive layers (*e.g.*, convolutional layers).

To transform a conventional model into a multi-exit model, we add an auxiliary classifier after each block as in Figure 2. Then, we feed the output of the block b to the corresponding classifier (*i.e.*, c_b) to obtain an early prediction:

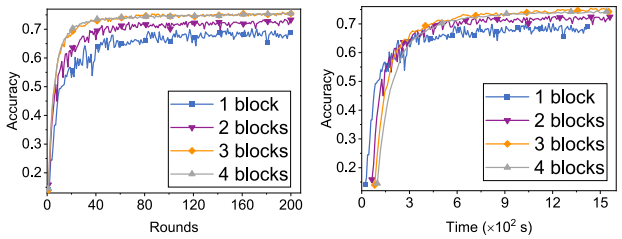
$$\hat{y}_b = c_b(z_b), 1 \leq b \leq B, \tag{4}$$

where z_b $(1 \leq b \leq B-1)$ is the input of both the auxiliary classifier and the original block $b+1$, and z_B is the input of the original classifier, *i.e.*, $c_B = c$. For convenience, we call the combination of original block b and its corresponding classifier as block b of the multi-exit model. We use w_b to denote the parameters of block b. Note that the transformation from a conventional DL model to the multi-exit model can be easily applied on any CNN-based model [14], *e.g.*, VGG [16] and ResNet [21], which have been widely used in various FL applications [4], [5].

Training Blocks on Devices. Due to the sequential nature of gradient updating in DL models [28], training inconsecutive blocks cannot necessarily save resources. For example, the training cost of blocks 1 and 3 is the same as that of blocks 1∼3, because the forward and backward propagations of block 2 are inevitable even if only blocks 1 and 3 need to be trained. To explore the feasibility of block training for saving resources, we conduct a preliminary experiment, where each device only trains a *block group* (not a full model) containing one or several consecutive blocks. A CNN model with four blocks is trained on 10 Jetson Tx2 devices to perform a speech recognition task, *i.e.*, KWS [29]. At each round, the parameter server will assign a block group randomly for each device to perform local training. In an experiment, we fix the number of blocks trained on devices, and change it (from 1 to 4) across different experiments. More details of the experiment are described in Section V.

The experimental results are presented in Figure 3. At the early stage (*i.e.*, before achieving 60% model accuracy), training fewer blocks on each device makes the model converge faster in terms of training time. The reason lies in that training

[1] https://ai-benchmark.com



(a) Rounds vs. Accuracy (b) Time vs. Accuracy

Fig. 3. Different number of blocks on each device on KWS dataset.

a subset of blocks saves both computation and communication cost at each round. Compared to training a full model, training a block saves about 50% computation time and also reduces the transmitted data size (more details in Tables I-III of Section V). However, at the later stage, training 1 or 2 blocks fails to achieve higher model accuracy, compared with training 4 blocks (*i.e.*, a full model). That is because the model still needs end-to-end fine-tuning to improve its generalization performance [30]. Thus, we are expected to design an adaptive block assignment strategy to achieve fast convergence as well as high model accuracy.

Efficient and Accurate inference. The multi-exit models obtained from FL can also perform adaptive inference and make ensemble predictions, saving resource consumption and improving prediction accuracy. 1) *Adaptive Inference*. When performing inference, the multi-exit model will terminate execution if it is confident on its early output. The carefully designed early-exit techniques lead to at most 50% reduction of computation cost while only degrading the model accuracy by about 1% [15], [31], [32]. 2) *Ensemble Prediction.* Since multiple outputs are generated from different exits of the multi-exit model, we can further boost the model by performing ensemble prediction with negligible extra computation. Specifically, the predictions can be obtained based on the averaged outputs of all model exits. Compared with the single-exit model, the ensemble prediction improves the model accuracy by about 1%∼5% while achieving almost the same inference speed [33].

III. Design of BOSE

In this section, we propose a novel system, called BOSE, which implements ***blo***ck-wi***se*** model training on edge devices. We first present the overview of BOSE and further elaborate the detailed design of key modules.

A. System Overview

Our system contains two key roles, *i.e.*, a parameter server and edge devices. The whole training process involves a certain number of rounds. At each round, the parameter server first evaluates the blocks' potential for improving model performance and estimates the available resources on devices. In addition, we adjust the round deadline so as to accommodate system dynamics. Based on the deadline and blocks' potential, our system will adaptively assign different blocks for devices to perform block-wise local training. The illustration of system modules is presented in Figure 4.

Block-wise Training. Based on the local dataset, this module on devices trains the model in a block-wise manner. The computation and communication cost are both reduced by training and transmitting a subset of blocks.

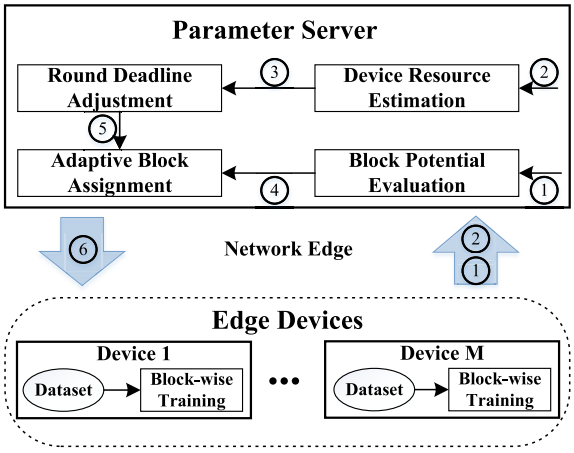


Fig. 4. System Overview. After local training, the updated blocks (①) and device statuses (②) are uploaded to the parameter server. Based on ① and ②, we estimate the training time of blocks (③) and measure the scores of blocks (④). Combining ④ and the round time (⑤), BOSE adaptively determines and assigns different blocks for devices (⑥).

Block Potential Evaluation. Different blocks in a model usually have distinct impacts on the training convergence [18], [20]. To indicate the blocks' potential for promoting the convergence, we compute a score for each block (④) based on the convergence statuses of the updated blocks (①).

Device Resource Estimation. For block assignment, we need to obtain the training time of different blocks on devices. However, due to resource dynamics, the training time of the same block on a device may be different over time. Alternatively, we use the recent device statuses (②) to estimate on-device resources and further calculate the training time of blocks (③) at the current round.

Round Deadline Adjustment. To avoid delaying by stragglers, a round deadline needs to be determined before assigning blocks for devices. However, a fixed deadline cannot guarantee fast training as well as high model accuracy, due to resource dynamics and varied deadline requirements at different training stages [34], [35], [36]. Thus, based on the estimated training time of blocks (③) and the current model convergence condition, this module will adaptively adjust the round deadline.

Adaptive Block Assignment. Considering both blocks' scores (④) and the round deadline (⑤), the parameter server adaptively assigns different blocks (⑥) for heterogeneous devices to perform local training, which alleviates the impact of stragglers as well as promotes model convergence.

B. Block-Wise Training

In most FL methods, *e.g.*, FedAvg, all devices always train the same model on their local datasets, which results in stragglers due to heterogeneous resources. Thus, we propose to assign one or several consecutive blocks (*i.e.*, a block group) for devices to perform local training in correspondence to their current available resources. Although the previous work ProgFed [11] partitions a model into several blocks for reducing training cost, we give an example to illustrate the difference from ours. As shown in Figure 5, ProgFed [11] initially trains the first block and then progressively adds more blocks until a full model is reached. When a block is added, the original classifier is discarded and a new classifier will be supplied after the final block. However, in ProgFed, all

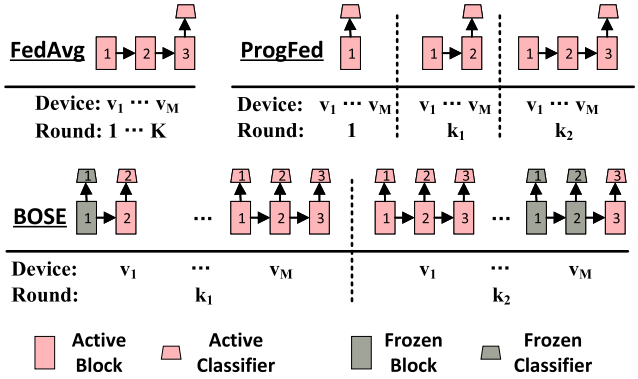
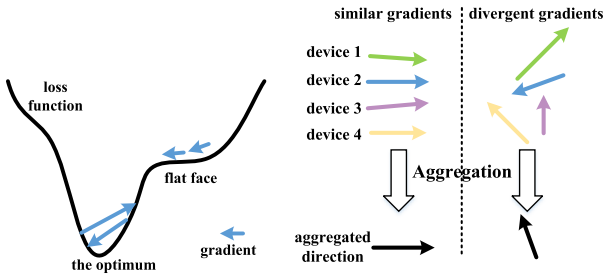


Fig. 5. Illustration of training mechanisms. Local models are: always the same (FedAvg); the same across devices but different across rounds (ProgFed); different across devices and rounds (BOSE).



(a) Flat face v.s. the optimum point. (b) Similar gradients v.s. divergent gradients.

Fig. 6. Two observations of gradients.

devices train the same model in a round, failing to deal with the heterogeneity among devices.

To achieve training efficiency among heterogeneous devices, we utilize the multi-exit model to implement block-wise training on devices. To be specific, a multi-exit model consists of several non-overlapping blocks, and one or several consecutive blocks form a block group, which can be trained separately. On devices, the blocks (or classifiers) that need to be trained are regarded as active, while others are frozen and remain unchanged during the training. We should note that those preceding frozen blocks of the active blocks still need to be transmitted from the parameter server to devices, and play the role of extracting data features when performing local updating. However, there is no need to transmit the subsequent frozen blocks of the active blocks and all frozen classifiers to devices. After local training, only those updated blocks and classifiers on devices should be uploaded to the parameter server. For example, in Figure 5, when device v_1 trains block 2 at round k_1, block 1, block 2 and classifier 2 will be transmitted to the device, and only the updated block 2 and classifier 2 need to be sent back to the parameter server for global aggregation. Thus, the communication cost is significantly reduced. Since the updating of parameters (*i.e.*, backward propagation) is computationally expensive, *e.g.*, accounting for 65∼75% of the total training time [37], the computation cost is also largely saved without updating frozen blocks. As listed in Tables I-III of Section V, if a device only trains the first block of ResNet-18, compared to training a full model, the communication and computation cost can be reduced by 99% and 89%, respectively. In addition, we can see that the active blocks are different across devices and rounds due to the possible dynamics of resources. The details of the block assignment will be elaborated in Section III-F.

Herein, we present formal descriptions of the block-wise training and aggregation process. At round k, w_b^k and $w_{b,m}^k$ are the parameters of block b in the global model and the local model of device v_m, respectively. For block b on device v_m, we calculate its loss value as:

$$F_m(w_{b,m}^k) = \mathbb{E}_{(x,y)\sim\mathcal{D}_m} l(w_{b,m}^k, x, y). \tag{5}$$

Given a block assignment $\mathcal{G}_m^k$, our overall objective is to minimize the following expression:

$$\sum_{b\in\mathcal{G}_m^k} F_m(w_{b,m}^k), \tag{6}$$

which is optimized by the widely-used SGD algorithm. After finishing local training, we perform block-wise aggregation at the server. For each block b, we first calculate s_b^k:

$$s_b^k = \sum_{m=1}^{M} \mathbb{I}_{b,m}^k \cdot n_m, \tag{7}$$

where $\mathbb{I}_{b,m}^k = 1$ if block b is assigned to device v_m at round k, and 0 otherwise. n_m is the number of data samples on device v_m. Then, the parameter of block b in the global model is updated as:

$$w_b^k = \frac{1}{s_b^k} \sum_{m=1}^{M} w_{b,m}^k \cdot \mathbb{I}_{b,m}^k \cdot n_m. \tag{8}$$

As a result, an updated global model w^{k+1} is generated for further training.

C. Block Potential Evaluation

In most training methods of FL, all blocks (*i.e.*, a full model) are trained on different devices during the whole training process. However, previous researches [18], [38] have demonstrated that it is unnecessary, since different blocks have diverse impact on the model convergence. To indicate blocks' potential for promoting model convergence, two metrics are adopted to measure the convergence statuses of blocks based on the following two observations.

Observation 1: Different blocks in a model converge at inconsistent speeds. For example, [38] shows that the parameters in the shallow layers converge faster than those in the deep layers. Prior works use gradient magnitude to indicate whether the parameters in a block converge or not, *i.e.*, small gradient magnitude means well-trained parameters and vice versa. However, gradient magnitude is not a good indicator [39]. On one hand, small gradients do not mean that the parameters are close to the optimum. As shown in Figure 6(a), saddle points at the flat face will lead to small gradients that are far from the optimum. On the other hand, the relatively large gradients do not indicate fast convergence. The parameters may oscillate around the optimal values due to the variance of SGD optimization [19].

Due to the above reasons, for block b at round k, we adopt the *learning speed* as an important indicator:

$$P_b^k = \frac{\| \sum_{i=0}^{r-1} \Delta_b^{k-i} \|}{\epsilon + \sum_{i=0}^{r-1} \| \Delta_b^{k-i} \|}, \tag{9}$$

where $\Delta_b^k = w_b^k - w_b^{k-1}, k \geq 2$ is the updating of block b at round k and r is the observation window. ϵ is a small

positive value, which helps to avoid the division-by-zero error. Obviously, the range of P_b^k is between 0 and 1. In general, the closer to its optimum a block is, the smaller its learning speed is, and vice versa. If the gradients are always of the same direction within the rounds $[k-r+1, k]$, then P_b^k is 1, indicating the block is far from the optimum. If the updates in the observation window completely cancel each other (*i.e.*, parameters oscillate around a point), then P_b^k would be 0, meaning that the block is close to the optimum.

Observation 2: The divergence degree of the local gradients is different across blocks. The variance of gradients derived from SGD optimization may lead the local models to different directions during the local training, *i.e.*, divergent updating across devices. It is known that the divergence degree across blocks may be different [20]. In fact, collecting similar gradients from many devices causes resource waste while contributing little to model convergence. As Figure 6 shows, choosing any one from these similar gradients is enough for global aggregation and also achieves training efficiency. For divergent blocks, we have to collect the gradients from more devices to obtain high-quality global updating. To measure the differences between local blocks and the aggregated global block, we compute the *device-wise divergence* of block b at round k as:

$$D_b^k = \frac{1}{M}\sum_{m=1}^{M} \|w_b^k - w_{b,m}^k\|_2^2, \tag{10}$$

where $w_{b,m}^k$ is the parameter of block b on device v_m while w_b^k is the parameter of the aggregated block b. Larger D_b^k indicates more diverse blocks.

Combining the above two metrics, we measure the convergence status of block b at round k as the following score:

$$S_b^k = P_b^k \cdot D_b^k. \tag{11}$$

The higher score S_b^k means a larger potential to improve the model performance by training block b at round k.

D. Device Resource Estimation

To efficiently assign blocks for devices with heterogeneous resources, the training time of different blocks on devices is required. However, due to resource dynamics, the training time of the same block on a device may vary over rounds. Alternatively, we use the recent device statuses (*i.e.*, the historical training time on devices) to estimate on-device resources. Further, based on blocks' parameter sizes and measured computation load, the estimated training time of blocks at the current round can be obtained.

For device v_m, the computation capability at round k can be estimated as:

$$\widehat{\psi}_m^k = \frac{\Pi_m^k}{t_{m,cp}^k}, \tag{12}$$

where Π_m^k is the computation load on device v_m at round k, and $t_{m,cp}^k$ is the actual measured training time. Herein, the computation load is expressed as the normalized training time of blocks, which can be easily measured in advance. We further calculate the moving average with $\alpha \in [0,1]$ (*e.g.*, 0.9 in our experiments) as:

$$\psi_m^k = \alpha\psi_m^{k-1} + (1-\alpha)\widehat{\psi}_m^k, \tag{13}$$

which makes the estimation more robust [5]. In the same way, we estimate the download bandwidth as:

$$\widehat{\gamma}_m^{k,in} = \frac{\Omega_m^{k,in}}{t_{m,cm}^{k,in}}, \tag{14}$$

where $\Omega_m^{k,in}$ is the size of the model transmitted to device v_m at round k, and $t_{m,cm}^{k,in}$ is the corresponding transmission time. Then, the moving average download bandwidth of device v_m at round k can be calculated as:

$$\gamma_m^{k,in} = \alpha\gamma_m^{k-1,in} + (1-\alpha)\widehat{\gamma}_m^{k,in}. \tag{15}$$

Similarly, we obtain the upload bandwidth $\gamma_m^{k,out}$. For device v_m, based on the estimated resources at round k, *i.e.*, ψ_m^k, $\gamma_m^{k,in}$ and $\gamma_m^{k,out}$, the time for training block group $\mathcal{G}$ at round $k+1$ can be estimated as:

$$\hat{t}_m^{k+1}(\mathcal{G}) = \frac{\Pi(\mathcal{G})}{\psi_m^k} + \frac{\Omega^{in}(\mathcal{G})}{\gamma_m^{k,in}} + \frac{\Omega^{out}(\mathcal{G})}{\gamma_m^{k,out}}, \tag{16}$$

where $\Pi(\mathcal{G})$, $\Omega^{in}(\mathcal{G})$ and $\Omega^{out}(\mathcal{G})$ are block group $\mathcal{G}$'s training load, download size and upload size, respectively. The information of blocks can be measured in advance.

E. Round Deadline Adjustment

At a round, all devices are expected to finish local training and model transmission before a predefined round deadline. However, a fixed deadline cannot guarantee fast training as well as high model accuracy, due to resource dynamics and varied deadline requirements at different training stages [34]. A long deadline makes the fast devices always wait for the stragglers, resulting in long idle time and resource waste. On the contrary, a short deadline guarantees fast convergence at the early stage of the training but hinders the model from achieving higher accuracy at the later stage. The reason lies in that the short deadline makes some devices continue training a few blocks (*i.e.*, small-scale models) and thus the models underfit local datasets. The similar phenomenon can also be observed in the works [34], [35], [36]. Thus, as in [34], we gradually increase the deadline during the training process.

Algorithm 1 Deadline Adjustment at Round k

Input: Device percentile ρ, percentile step $\hat{\rho}$.

1: **if** Model accuracy has not improved for $\hat{k}$ rounds **then**
2: $\quad\rho = \min(\rho + \hat{\rho}, 80\%)$;
// $\hat{k} = 5$ and $\hat{\rho} = 10\%$ in BOSE
3: Estimate the time for training all blocks on devices using Equation (16);
4: Sort the estimated time as $t'_1, \cdots, t'_M$ in ascending order;
5: Set the round time $T^k = t'_{\lceil M\cdot\rho\rceil}$;

Output: Round deadline T^k.

To this end, the deadline in our system is adaptively adjusted, as shown in Algorithm 1. To deal with resource dynamics, we set the deadline as the time of the fastest $\rho \in (0, 100\%]$ of devices that finish training all blocks. To achieve fast convergence without sacrificing model accuracy, ρ is initialized as 10% and increased by 10% when the accuracy of the global model has not improved for a certain number (*e.g.*, 5) of rounds (Line 1-2). To avoid the stragglers

(*i.e.*, devices with very limited resources), we empirically set the maximum ρ as 80%. In other words, the slowest 20% of devices are always allowed to train a subset of blocks. Then, we estimate the time for training all blocks on devices using Equation (16) (Line 3). The estimated training time of devices is sorted in the ascending order: $t'_1 \leq \cdots \leq t'_M$, and choose $t'_{\lceil M \cdot \rho \rceil}$ as the round deadline T^k (Line 4-5). By gradually increasing the deadline, fast convergence speed as well as high model accuracy are both achieved.

F. Adaptive Block Assignment

At each round, the parameter server will adaptively assign blocks for devices to perform local training. We aim to optimize the following loss function with resource constraints:

$$\min_{w} \sum_{b=1}^{B} F(w_b) \tag{17}$$

$$\textbf{s.t.} \quad t_m^k \leq T^k, \forall v_m \in \mathcal{V}, \tag{18}$$

where B is the number of blocks, $F(\cdot)$ is the global loss function defined in Equation (2) and T^k is the derived deadline at round k, as described in Section III-E. w is the union of the parameters of all blocks, *i.e.*, $w = \{w_1, \ldots, w_B\}$. t_m^k is the training time of device v_m at round k:

$$t_m^k = t_{m,cp}^k + t_{m,cm}^{k,in} + t_{m,cm}^{k,out}, \tag{19}$$

where $t_{m,cp}^k$, $t_{m,cm}^{k,in}$ and $t_{m,cm}^{k,out}$ are the time of local updating, model download and model upload, respectively. Since the actual values of the variables in Equation (19) are unknown before local training and model transmission, we estimate these variables by Equation (16) in Section III-D.

Although obtaining the optimal solution, *i.e.*, w, for Equation (17) is the ultimate goal of our system, the complexity of deep learning models renders it infeasible to derive a closed-form solution directly. Consequently, we employ adaptive block assignment to optimize Equation (17), with the objective of effectively utilizing the available resources on all devices to enhance model convergence. In Sections II-C and III-C, we provide observations and analyses that serve as the basis for two principles to guide block assignment, while considering the constrained on-device resources: 1) Maximizing the number of blocks trained on each device within the round deadline constraint; 2) Training the blocks with higher potential scores more frequently.

The process of block assignment is presented in Algorithm 2. As a block group contains one or several consecutive blocks, we use $\mathbb{B}$ to denote the set of all $\frac{B(B+1)}{2}$ possible groups. For device v_m, we initialize its feasible set of block groups $\overline{\mathbb{B}} = \emptyset$. For each group $\mathcal{G} \in \mathbb{B}$, we estimate its completion time $t_m^k(\mathcal{G})$ on device v_m at round k using Equation (16) (Line 3-4). If the estimated time is not longer than the round deadline, we add the block group into the feasible set (Line 5-6). However, it is possible to have redundant block groups in the set, which could lead to suboptimal resource utilization. For example, the group consisting of blocks 1-3 is included in the set as long as the group consisting of blocks 1-4 is in the set. It is clear that the completion time of the former group will be shorter than the latter. In this case, if a device is capable of training blocks 1-4, it should train blocks 1-4 instead of blocks 1-3 to maximize the utilization of available resources. To eliminate redundancy, we introduce a step to remove any block groups from the set that are subsumed by another block group with a larger block range (Line 7). As a result, a feasible group set $\overline{\mathbb{B}}$ is obtained. Then, we calculate the score of each group $\mathcal{G} \in \overline{\mathbb{B}}$ as:

$$S(\mathcal{G}) = \sum_{b \in \mathcal{G}} S_b. \tag{20}$$

The higher score $S(\mathcal{G})$ means larger potential to promote model convergence by training block group $\mathcal{G}$.

Based on the scores, an intuitive method is to only train the blocks with high scores and always freeze others. However, the blocks with low scores still need to be trained to adapt to the changes of other blocks [19]. Thus, given the set $\overline{\mathbb{B}}$ and the scores of each block group, we propose a probability-based strategy for block assignment (Line 8-10). Specifically, the probability of choosing the group $\mathcal{G} \in \overline{\mathbb{B}}$ is set as:

$$p(\mathcal{G}) = \frac{S(\mathcal{G})}{\sum_{\mathcal{G}' \in \overline{\mathbb{B}}} S(\mathcal{G}')}. \tag{21}$$

Based on the probabilities, we randomly choose one block group for each device to perform local training. As a result, considering the training statuses of blocks and available on-device resources, different block groups are assigned for heterogeneous edge devices, which achieves training efficiency. At each round, our algorithm will calculate the potential scores of B blocks and $\frac{B(B+1)}{2}$ block groups. For each device, we will estimate the completion time of these block groups on it. Thus, the time complexity of our block assignment is $O(MB^2)$, where M and B are the number of devices and the number of blocks, respectively. As in most works of multi-exit models [15], [33], B is usually smaller than 10 (typically 4∼6). Thus, the additional overhead for block assignment is negligible, which is demonstrated in Section V-E.

Algorithm 2 Adaptive Block Assignment at Round k

Input: Estimated computation capability ψ_m^{k-1}, download/upload bandwidth $\gamma_m^{k-1,in}/\gamma_m^{k-1,out}$, $\forall v_m \in \mathcal{V}$; Round deadline T^k; All possible block groups $\mathbb{B}$.

1: **for** $v_m \in \mathcal{V}$ **do**
2: Initialize the set of candidate block groups: $\overline{\mathbb{B}} = \emptyset$
3: **for** Block group $\mathcal{G} \in \mathbb{B}$ **do**
4: Compute $\hat{t}_m^k(\mathcal{G})$ as Equation (16);
5: **if** $\hat{t}_m^k(\mathcal{G}) \leq T^k$ **then**
6: Add $\mathcal{G}$ in the set $\overline{\mathbb{B}}$;
7: Remove the redundant groups in $\overline{\mathbb{B}}$;
8: Compute $S(\mathcal{G}), \forall \mathcal{G} \in \overline{\mathbb{B}}$ as Equation (20);
9: Set $p(\mathcal{G}), \forall \mathcal{G} \in \overline{\mathbb{B}}$ as Equation (21);
10: Randomly select one group $\mathcal{G}_m^k$ based on the probabilities $p(\mathcal{G}), \forall \mathcal{G} \in \overline{\mathbb{B}}$;

Output: Block assignments $\mathcal{G}_m^k, \forall v_m \in \mathcal{V}$.

G. The Whole Training Process

Herein, we summarize the whole block-wise training procedure of BOSE in Algorithm 3. At the beginning of each round, each device trains the assigned blocks on its local dataset by minimizing the loss function defined in Equation (6) (Line 4). Upon finishing local training, devices transmit the

Algorithm 3 BOSE: Block-Wise Federated Learning

Input: Participating devices $\mathcal{V}$, Time budget T.

1: Initialize round number $k = 1$, block assignment $\mathcal{G}_m^k = \{\text{Whole model}\}, \forall v_m \in \mathcal{V}$;
2: Measured training time $t_{m,cp}^k$, download/upload time $t_{m,cm}^{k,in}/t_{m,cm}^{k,out}$, round time t^k; Estimated computation capability ψ_m^k, download/upload bandwidth $\gamma_m^{k,in}/\gamma_m^{k,out}$, block potential S_b^k; Round deadline T^k;
3: **while** $T > 0$ **do**
4: Devices train assigned blocks by minimizing Equation (6);
5: Trained blocks in $\mathcal{G}_m^k$ and local statuses $t_{m,cp}^k, t_{m,cm}^{k,in}, t_{m,cm}^{k,out}$ are sent to the server;
6: Local blocks are aggregated as Equations (7) and (8);
7: S_b^k is calculated as Equations (9)–(11);
8: $\psi_m^k, \gamma_m^{k,in}, \gamma_m^{k,out}$ are estimated as Equations (12)–(15);
9: T^{k+1} is obtained by calling Algorithm 1;
10: $\mathcal{G}_m^{k+1}$ is obtained by calling Algorithm 2;
11: $T \leftarrow T - t^k$, $k \leftarrow k + 1$;

Output: Trained global model w.

updated blocks, along with the measured training and communication time, to the server (Line 5). Then, the server performs block-wise aggregation as Equations (7) and (8) (Line 6). Subsequently, the potential score of each block is calculated based on the locally trained blocks and the globally aggregated blocks using Equations (9)–(11) (Line 7), while the available resources on the devices are estimated using Equations (12)–(15) (Line 8) based on the measured training and communication time. Besides, the round deadline for the next round is determined by calling Algorithm 1 (Line 9). Based on the block potential, available resources and round deadline, block assignments for the next round are determined by calling Algorithm 2 (Line 10). Then, time budget and round number are updated (Line 11). Finally, when the time budget is exhausted, the training process is finished and a final trained model is obtained.

IV. Theoretical Analysis

In this section, we study the convergence of the multi-exit model in a block-wise manner. It is worth noting that except for the input data of the first block, the inputs of remaining blocks change as the training proceeds [28]. This implies that the input data distribution of block b ($b > 1$) depends on its preceding blocks. We denote the input data distribution of block b on device v_m at round k as $p_{m,b}^k$ and assume the converged distribution to be p_b^*.

To analyze the convergence property of the block-wise training, we make the following assumptions:

Assumption 1 (L-Smoothness): For a constant $L > 0$ and the block b, we have:

$$\|\nabla F_m(w_b) - \nabla F_m(w_b')\| \leq L\|w_b - w_b'\|, \forall v_m \in \mathcal{V}, \quad (22)$$

where w_b and w_b' are the parameters of the block b at different rounds.

Assumption 2 (Convergence of the Previous Blocks): Following [28], we adopt the variation distance between the local distribution and the converged distribution: $c_{m,b}^k \triangleq \int |p_{m,b}^k - p_b^*| dz$, where z represents the data samples drawn from the distribution p, *i.e.*, $z \sim p$. To simplify the analysis, we use $c_b^k = \max_m c_{m,b}^k$, and assume the input of block b will eventually converge:

$$\sum_k c_{b-1}^k < \infty. \quad (23)$$

Assumption 3 (Bounded Gradient): For a constant G, we have:

$$\|F_m(w_{m,b}^k)\|^2 \leq G, \forall m, b, k. \quad (24)$$

Lemma 1: Under Assumptions 1-3, we have:

$$\mathbb{E}[F(w_b^{k+1})] \leq F(w_b^k) + \frac{LG\eta^2}{2M} - \eta\left(\frac{1}{2}\|\nabla F(w_b^k)\|^2 - \frac{Gc_{b-1}^k}{M}\right), \quad (25)$$

where η is the learning rate. In APPENDIX A, we give the proof of Lemma 1.

Theorem 1: Under Assumptions 1-3 and Lemma 1, we obtain the following convergence bound:

$$\frac{1}{K}\sum_{k=0}^{K-1}\|\nabla F(w_b^k)\|^2 \leq \frac{2(F(w_b^0) - F(w_b^K))}{K\eta} + \frac{2G}{MK}\sum_{k=0}^{K-1} c_{b-1}^k + \frac{LG\eta}{M}. \quad (26)$$

The proof is given in APPENDIX B.

Corollary 1: Referring to the analyses in the previous works [11], [40], we define:

$$q^k = \max_b \frac{\|\nabla F(w^k)\|^2}{\|\nabla F(w_b^k)\|^2} \quad \text{and} \quad q' = \max_k q^k. \quad (27)$$

Besides, we also define:

$$F' = \max_b F(w_b^0) \quad \text{and} \quad c' = \max_{k,b} c_{b-1}^k. \quad (28)$$

Then, we will obtain the convergence of the full model:

$$\frac{1}{K}\sum_{k=0}^{K-1}\|\nabla F(w^k)\|^2 \leq \frac{2q'F'}{K\eta} + \frac{2q'Gc'}{M} + \frac{q'LG\eta}{M}, \quad (29)$$

which is proved in APPENDIX C.

Theorem 1 shows that the convergence of block b is affected by the input data distribution c_{b-1}^k, which, in turn, depends on the convergence of all the preceding blocks of block b. This suggests that if block b is far from its optimum, it should be trained on more devices (*i.e.*, larger M) to reduce the term $\frac{2G}{MK}\sum_{k=0}^{K-1} c_{b-1}^k$, which will promote the convergence of all subsequent blocks of block b.

V. Evaluation

In this section, we conduct extensive experiments on a physical edge platform to evaluate the efficiency of our system. The experiments are performed based on PyTorch [41], which is a widely used deep learning framework.

A. Experimental Settings and Baselines

System Deployment. We employ an AMAX deep learning workstation as the parameter server and 50 NVIDIA Jetson kits as devices, including 20 TX2, 20 Xavier NX and 10 AGX Xavier. The parameter server is equipped with an Intel(R) Core(TM) i9-10900X CPU, 4 NVIDIA GeForce RTX 2080Ti GPUs and a 128GB RAM. Each TX2 has a 256-core Pascal GPU and a CPU cluster consisting of a 2-core Denver2 and a 4-core ARM CortexA57. Each NX (or AGX) is equipped with a 384-core (or 512-core) NVIDIA Volta GPU and a 6-core (or 8-core) NVIDIA Carmel CPU.

Device Heterogeneity and Dynamics. To set various edge devices with dynamic computation and communication capabilities, we give the configuration of devices as follows.

- **For Computation.** TX2 and NX can work in one of four computation modes while AGX has eight modes. Devices working in different modes exhibit diverse capabilities. Specifically, the fastest mode (*i.e.*, mode 0 of AGX) performs training 5× faster than the slowest one (*i.e.*, mode 1 of TX2). To mimic time-varying on-device resources, we randomly change the modes of devices every 20 rounds.
- **For Communication.** All devices are connected to the parameter server via 2.4GHz WiFi. We arrange 50 devices in 3 groups (each of which contains 20, 20 and 10 devices, respectively). Then, the devices in the groups are placed at different locations that are 2m, 10m and 20m away from the WiFi router. Due to random noises and competition among devices, the bandwidth between the parameter server and devices varies dynamically. During the training, the measured bandwidth of devices changes within the range of [1Mb/s, 30Mb/s].

Tasks, Datasets and DL Models. To verify the performance of BOSE across applications, we evaluate BOSE on two popular tasks: image classification and speech recognition, which are widely deployed on edge devices.

- **Image Classification.** We adopt two classical datasets in computer vision areas, *i.e.*, CIFAR-10 and CIFAR-100 [42]. Concretely, CIFAR-10 contains 60,000 32×32 color images labeled in 10 classes with 50,000 samples for training and 10,000 samples for test. CIFAR-100 has the same number of images but consists of 100 classes, which is more challenging to train models for classification. We train two well-known DL models to perform image classification tasks, *i.e.*, VGG-16 [16] on CIFAR-10 and ResNet-18 [21] on CIFAR-100. We add an auxiliary classifier after each pooling layer of VGG-16 and residual block of ResNet-18, respectively. For both VGG-16 and ResNet-18, a full model is divided into 5 blocks.
- **Speech Recognition.** For the task of speech recognition, the Google speech dataset [29] is adopted, which aims to recognize voice commands from 35 target words (*e.g.*, "yes", "left" and "seven") and is often called keyword spotting (KWS). The dataset includes 85,511 and 4,890 audio clips for training and test, respectively. The model trained on KWS is a CNN network with 4 1-D convolution layers. Each layer is followed by an auxiliary classifier. Thus, there are 4 blocks in the CNN model.

We present the parameter size, peak memory footprint and training time of blocks (or block groups) in Tables I-III, respectively. Memory footprint is measured by the code in [43]. Since computation capabilities of edge devices are distinct, the training time of blocks is normalized by that of the corresponding first block. From Tables I-III, we can see that training a part of the model significantly reduces training cost, compared to training the full model.

TABLE I

PEAK MEMORY FOOTPRINT (MB) OF BLOCK GROUPS

	Block						Classifier					
No.	1	2	3	4	5	**sum**	1	2	3	4	5	**sum**
VGG-16	0.15	0.85	5.64	22.53	27.03	**56.02**	0.04	0.04	0.04	0.02	0.02	**0.16**
ResNet-18	0.01	0.57	2.00	8.00	32.00	**42.58**	0.39	0.39	0.44	0.39	0.20	**1.81**
CNN (×0.1)	0.10	0.12	0.25	0.48	-	**0.95**	0.17	0.17	0.09	0.09	-	**0.52**

×0.1 denotes that the actual parameter size is 0.1 times the listed value.
- denotes that there is no such block or block group (the same in Tables 2 and 3).

TABLE II

PEAK MEMORY FOOTPRINT (MB) OF BLOCK GROUPS

Block Group	1	2	3	4	5	1-2	2-3	3-4	4-5	1-3	2-4	3-5	1-4	2-5	1-5
VGG-16	4.2	2.5	3.3	12.3	17.3	6.6	4.8	14.4	25.7	8.9	14.8	27.8	14.8	28.2	28.3
ResNet-18	2.2	9.3	6.1	5.0	17.8	10.3	14.1	9.4	21.0	15.2	17.4	21.9	18.5	22.6	23.7
CNN	0.80	0.33	0.21	0.16	-	0.94	0.39	0.23	-	1.00	0.41	-	1.03	-	-

TABLE III

NORMALIZED TRAINING TIME OF BLOCK GROUPS

Block Group	1	2	3	4	5	1-2	2-3	3-4	4-5	1-3	2-4	3-5	1-4	2-5	1-5
VGG-16	1.0	1.1	1.2	1.7	2.1	1.5	1.6	2.1	2.7	2.0	2.4	3.1	2.9	3.5	3.9
ResNet-18	1.0	2.7	2.5	2.9	3.9	2.9	4.8	4.1	4.8	5.0	6.4	6.0	6.7	8.3	8.7
CNN	1.0	0.9	1.0	1.0	-	1.2	1.2	1.3	-	1.7	1.6	-	2.1	-	-

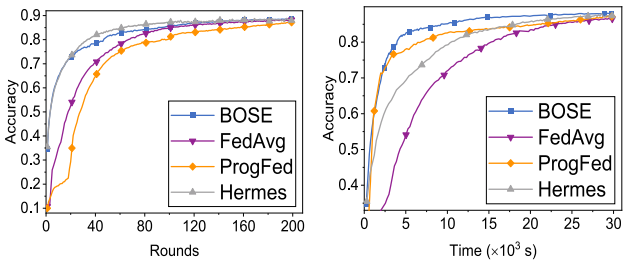
Baselines. We compare BOSE with the following baselines.

- **FedAvg** [3] is a widely-used method in FL. The parameter server broadcasts the full global model to edge devices for local training and then aggregates the updated local models from devices.
- **ProgFed** [11] also involves partitioning a full model into several parts, similar to the blocks in BOSE. ProgFed starts with the first block and gradually adds more blocks until the full model is trained. However, ProgFed does not consider device heterogeneity, as all devices in each round train the same local model.
- **Hermes** [13] also addresses device heterogeneity in FL. It tailors a small submodel for each device by applying structured pruning. However, the structures of the submodels obtained by Hermes are fixed, which limits its ability to adapt to the dynamic resource variations.

Evaluation Metrics. We employ three metrics to evaluate the performance of different FL systems:

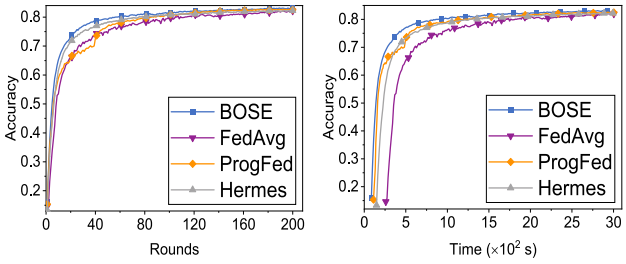
- **Test accuracy**. At each round, we evaluate the global model on the test dataset at the parameter server and use test accuracy to measure the performance of DL models.
- **Completion time**. We will report the time to achieve the target test accuracy, which is used for evaluating the training speed.
- **Communication traffic**. The total communication cost for model (or block) transmission through the network is also recorded when achieving the target test accuracy.

Training Details. If not specified, all experiments are conducted with the following default settings. Adam optimizer is adopted to update the parameters of DL models, and the learning rate is set to 0.001 [44]. The batch size and local epoch are 32 and 5 [35], respectively. For CIFAR-10/100, each device holds 500 images for local training while for KWS, there are 1,000 audio clips on each device. These local datasets are disjoint and selected from the original dataset randomly



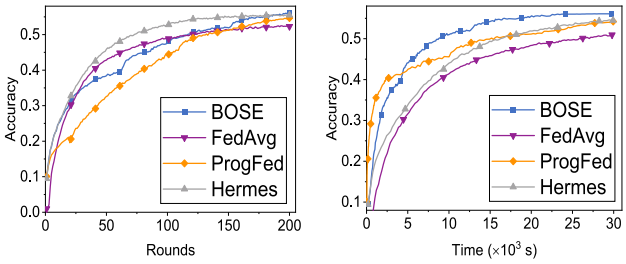
(a) Accuracy vs. Rounds (b) Accuracy vs. Time

Fig. 7. Performance for VGG-16 on CIFAR-10.



(a) Accuracy vs. Rounds (b) Accuracy vs. Time

Fig. 8. Performance for ResNet-18 on CIFAR-100.



(a) Accuracy vs. Rounds (b) Accuracy vs. Time

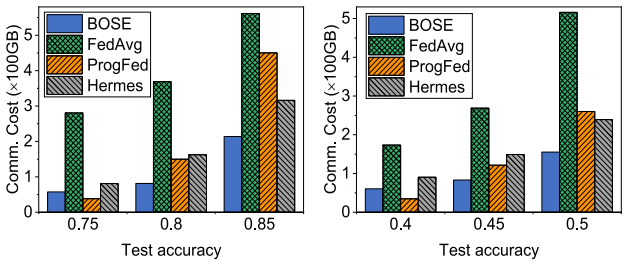
Fig. 9. Performance for CNN on KWS.

and uniformly. In each experiment, 50 devices participate in the training.

B. Overall Performance

We first compare the performance between our system and the baselines. We present how fast the training converges in terms of both rounds and time and measure the communication cost during the training process. Note that we focus on the comparison of training efficiency among different systems given the resource constraints, training models for state-of-the-art accuracy is beyond the scope of this work.

Convergence Speed. Figures 7-9 present the training performance of VGG-16 on CIFAR-10, ResNet-18 on CIFAR-100 and CNN on KWS, respectively. For small models (*e.g.*, CNN on KWS) and/or simple tasks (*e.g.*, VGG-16 on CIFAR-10), Figures 7 and 9 show that all systems can achieve similar final accuracy. While some baseline systems may converge faster than our system in terms of the number of rounds, BOSE consistently outperforms them in terms of overall completion time. That is because the baseline systems ignore the heterogeneity among devices, resulting in longer round time due to the presence of stragglers. Specifically, on CIFAR-10, the completion time of BOSE to achieve 80% accuracy is 3,971s, while that of FedAvg, ProgFed and Hermes is 15,308s, 6,925s, and 10,584s, respectively. In other words, BOSE provides a speedup of



(a) VGG-16 on CIFAR-10 (b) ResNet-18 on CIFAR-100

Fig. 10. Communication cost to achieve the target accuracy.

about 3.8×, 1.7× and 2.6×. For CNN on KWS, the corresponding speedup is 1.9×, 1.4× and 1.5×, when achieving 80% accuracy. As for the challenging task (*e.g.*, ResNet-18 on CIFAR-100), BOSE achieves higher accuracy given the same time budget, compared with baselines. Specifically, in 30,000s, BOSE improves the accuracy by about 5.1%, 1.9% and 1.5%, compared with FedAvg, ProgFed and Hermes, respectively. These experiments demonstrate that BOSE always converges fast without sacrificing model accuracy.

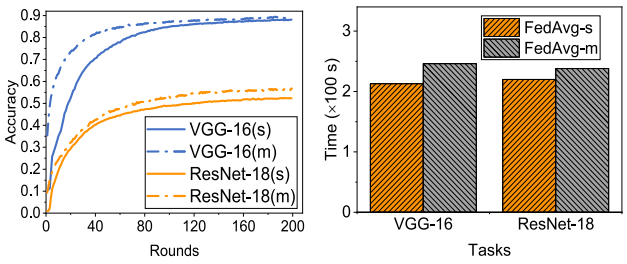
Communication Cost. In Figure 10, we compare the communication cost of different systems when training models on CIFAR-10 and CIFAR-100. When achieving the low target accuracy (*e.g.*, 0.75 for CIFAR-10 and 0.4 for CIFAR-100), ProgFed consumes less communication cost than others as it only transmits the shallow layers in the early stage. For example, to achieve 75% accuracy, the communication cost of ProgFed is 38GB while that of BOSE, FedAvg and Hermes is separately 57GB, 281GB, and 81GB. However, the heuristic block growing strategy of ProgFed makes it difficult to achieve higher test accuracy, resulting in more training rounds and thus higher communication cost. When achieving 85% test accuracy on CIFAR-10, BOSE saves the communication cost by about 72%, 52% and 32%, compared with FedAvg, ProgFed and Hermes, respectively. The corresponding reduction on CIFAR-100 to achieve 50% accuracy is 70%, 40% and 35%. Thus, BOSE is also able to save communication cost during training.

C. Evaluation of Key Modules

In BOSE, there are three key modules that are designed to improve training efficiency, including block-wise training, round deadline adjustment and adaptive block assignment. Herein, we conduct several sets of experiments to evaluate the effectiveness of these critical modules.

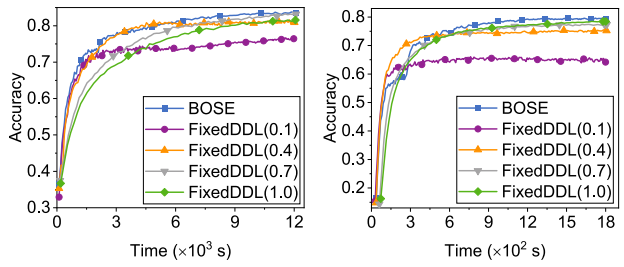
Block-wise Training. Different from end-to-end model training, BOSE performs block-wise training by utilizing multi-exit models. To investigate the impact of two training mechanisms, we conduct a set of experiments to compare end-to-end training (*i.e.*, single exit, denoted as FedAvg-s) and block-wise training (*i.e.*, multiple exits, denoted as FedAvg-m). In FedAvg-m, each edge device will train all blocks simultaneously. The results are shown in Figure 11.

As we can see, FedAvg-m improves the model accuracy by about 0.8% for VGG-16 on CIFAR-10 and 4.0% for ResNet-18 on CIFAR-100. The average per-round time is presented in Figure 11(b), and we observe that the computation overhead of FedAvg-m increases about 8% for VGG-16 and 15% for ResNet-18, compared to FedAvg-s. In terms of per-round communication cost, as shown in Table I, the parameter size of auxiliary classifiers (*i.e.*, classifiers 1-4) only accounts for



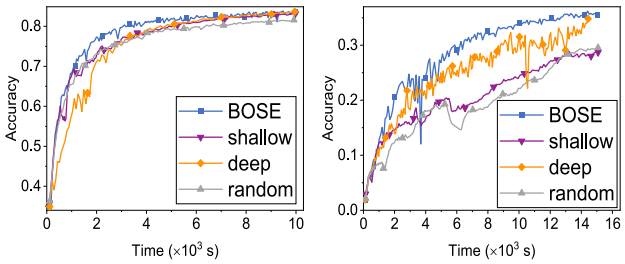
(a) Accuracy vs. Rounds (b) Average Per-round Time

Fig. 11. Training on FedAvg-s (s for short) and FedAvg-m (m for short). Task: VGG-16 on CIFAR-10 and ResNet-18 on CIFAR-100.



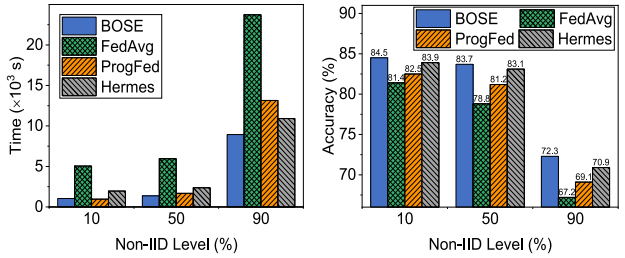
(a) VGG-16 on CIFAR-10 (b) CNN on KWS

Fig. 12. Training with adaptive and fixed deadlines.



(a) VGG-16 on CIFAR-10 (b) ResNet-18 on CIFAR-100

Fig. 13. Comparison of block selection strategies.



(a) Time to reach 70% accuracy (b) Accuracy within 15,000s

Fig. 14. Training on non-IID CIFAR-10.

0.2% and 3.6% of the total size of VGG-16 and ResNet-18, respectively. The improvement of FedAvg-m against FedAvg-s lies in that the auxiliary classifiers provide gradients information close to the shallow layers, which alleviates the vanishing gradient problem in the end-to-end model training. This set of experiments confirms the benefits of multi-exit models for FL training.

Round Deadline Adjustment. In BOSE, when the model accuracy fails to increase for 5 rounds, the round deadline will be adjusted. On CIFAR-10 and KWS with 20 devices, we evaluate the performance of adaptive (*i.e.*, in BOSE) and fixed deadline, denoted as FixedDDL(ρ), $\rho = 0.1, 0.4, 0.7$ and 1.0. For FixedDDL(ρ), the round deadline is always set as the completion time of the fastest ρ devices that finish training all blocks.

As shown in Figure 12, for both tasks, FixedDDL(0.1) and FixedDDL(0.4) converge fast at the initial stage of the training but fail to achieve higher model accuracy. While FixedDDL(0.7) and FixedDDL(1.0) can reach high accuracy but at the expense of a slow speed of the model convergence. In contrast, BOSE adjusts the deadline adaptively and achieves the best trade-off between convergence speed and model accuracy. Compared to FixedDDL(ρ), BOSE achieves 0.5%∼7.0% and 1.2%∼13.9% accuracy improvement on CIFAR-10 (in 12,000s) and KWS (in 1,800s), respectively.

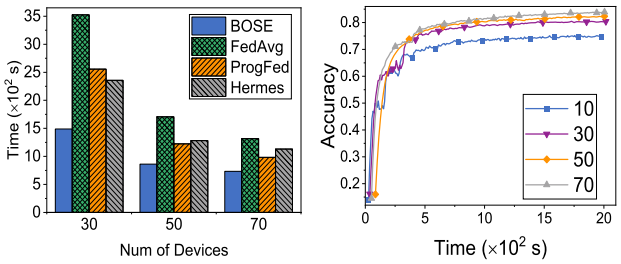
Adaptive Block Assignment. Given the available resources on devices and the training statuses of blocks, BOSE assigns blocks adaptively for devices to perform local training. To evaluate the effectiveness of adaptive block assignment strategy in BOSE, we use VGG-16 on CIFAR-10 and ResNet-18 on CIFAR-100 as examples. BOSE is compared with the other three strategies, *i.e.*, *shallow*, *deep*, and *random*. Specifically, *shallow* always assigns the blocks starting from the first block (*e.g.*, block groups 1-2 and 1-5) while *deep* always selects the blocks ending in the last block (*e.g.*, block groups 3-5 and 4-5). For *random*, block groups are selected randomly without considering the resources of devices and the convergence statuses of blocks.

There are 20 devices in this set of experiments and the results are presented in Figure 13. BOSE consistently outperforms the other three block assignment strategies. When achieving 80% accuracy on CIFAR-10, BOSE is 1.6×, 1.4× and 2.0× faster than *shallow*, *deep*, and *random*, respectively. Correspondingly, BOSE improves model accuracy by about 9.6%, 1.7% and 11.8%, within 10,000s time budget on CIFAR-100. The *random* strategy always behaves the worst since it doesn't take the on-device resources and blocks' convergence statuses into consideration, resulting in many stragglers and significant resource waste. The results verify the importance of the adaptive block assignment strategy.

D. Effect of Data Distribution

In the practical scenario, the local datasets among edge devices are usually not independent and identically distributed (non-IID) as users have different usage patterns [45]. Herein, we perform a set of experiments on CIFAR-10 with 20 devices to investigate the effect of non-IID data on training performance. We partition the original dataset into local datasets with different non-IID levels χ (*e.g.*, 10%, 50% and 90%) as in the previous work [46]. The non-IID level χ denotes that χ of data samples in a local dataset are in the same class while the remaining samples belong to other classes. Note that $\chi = 10\%$ on CIFAR-10 is a special case, which denotes the IID setting.

In Figure 14(a), we present the completion time to achieve the target accuracy on different systems across three non-IID levels (*i.e.*, $\chi = 10\%, 50\%$ and 90%). We observe that as the non-IID level increases, all systems take more time to reach the target accuracy. For example, on different systems, the time to achieve 70% accuracy with $\chi = 90\%$ is 2.9×∼12.5× more than that with $\chi = 10\%$ and 50%. Nevertheless, BOSE is always the fastest among all systems. Compared with the baselines, BOSE provides a speedup of 1.2×∼4.7× across different data distributions. Furthermore, in Figure 14(b), we present the accuracy achieved by various systems within



(a) Time to reach 80% accuracy (b) Accuracy v.s. Time on BOSE

Fig. 15. Training CNN on KWS with different number of devices.

TABLE IV
AVERAGE TIME (S) FOR THREE RUNNING STEPS OF THE TRAINING ACROSS DEVICES AND ROUNDS WHEN TRAINING CNN ON KWS

Number of devices	10	30	50	70
Local training	8.54	8.93	9.31	9.57
Communication	1.20	1.58	1.99	2.31
Block assignment	0.062	0.091	0.119	0.135

15,000s time budget. BOSE consistently outperforms the baseline systems under different non-IID levels. Specifically, BOSE achieves 0.6%∼5.1% higher accuracy. These experimental results demonstrate the effectiveness of our system even on non-IID datasets.

E. Effect of System Scale

In this section, we evaluate the performance of BOSE and baseline systems with different number (*i.e.*, 30, 50 and 70) of participating devices. We train the CNN model on KWS and present the completion time to achieve 80% accuracy in Figure 15(a). With the increasing number of devices, all systems converge faster. For example, FedAvg with that with 70 devices achieves a speedup of 2.6× and 1.3×, compared with 30 and 50 devices, respectively. The reason is that when more devices are involved, a larger amount of training data becomes available. This increased data volume allows the model to learn more information in each training round, leading to faster convergence. With different numbers of participating devices, BOSE reaches the target accuracy 1.4×∼2.4× faster than the baseline systems.

In Figure 15(b), we present the learning curves of BOSE with 10, 30, 50 and 70 devices. The convergence speed is almost the same while the models trained on a small number of devices only achieve low accuracy. Specifically, the final accuracy of the models trained on 10, 30, 50 and 70 devices is 75.1%, 80.4%, 82.3% and 83.9%, respectively. The similar phenomena are also observed in other baselines. These results indicate that simply dropping low-capability devices may cause degradation of model accuracy [47]. Thus, assigning different training loads (*e.g.*, heterogeneous blocks in BOSE) based on devices' available resources is an effective approach.

Furthermore, Table IV presents the breakdown of the time spent on each running step of BOSE, which is averaged across devices and rounds. As the number of devices increases, the time spent on each running step increases accordingly. The longer local training time is attributed to the higher device heterogeneity that arises from the larger number of devices. The increasing devices lead to intensified competition for bandwidth resources, resulting in longer communication time. The server experiences a heavier workload in aggregating a larger number of local parameters and assigning blocks to a larger set of devices, which consequently prolong the time spent on the server. Nevertheless, we find that the time of block assignment is significantly shorter than other running steps of the system, which highlights the efficiency of BOSE. When even more devices (*e.g.*, 200, 1000) participate in the training, the bandwidth of the server for collecting local blocks may become the system bottleneck. In this case, a common practice is to select a part of devices (*e.g.*, 100 out of 1000) in each round to perform local training. For example, random sampling [3] or advanced fine-grained selection [36] can be adopted.

Moreover, we observe an interesting phenomenon: as the number of devices increases, the time taken for each round (as indicated in Table IV) is slightly extended, but the time required to achieve a target accuracy (as depicted in Figure 15) is actually reduced. This is attributed to the fact that with more devices, more data are available for training, leading to more knowledge being learned in each round and thus the reduced number of rounds to reach the target accuracy. As a result, the overall completion time is shortened.

TABLE V
TRAINING PERFORMANCE WITH DIFFERENT α WHEN TRAINING CNN ON KWS

Value of α	0.0	0.3	0.6	0.9
Model accuracy (%)	81.3	81.5	82.1	82.4
Time to reach target (s)	1002	931	905	863
Average estimation error	3.18	2.66	2.25	1.74

F. Sensitivity of the Parameter α

We conduct experiments to examine the sensitivity of the parameter α in BOSE, which is set to 0.9 by default. Table V reports the results of these experiments, including the final model accuracy, the time to reach 80% accuracy, and the average estimation error, with varying values of α used in Equations (13) and (15) for resource estimation. Suppose the block group $\mathcal{G}_m^k$ is assigned for device v_m at round k. The estimation error is defined as:

$$err_m^k = (\hat{t}(\mathcal{G}_m^k) - t(\mathcal{G}_m^k))^2, \tag{30}$$

where $\hat{t}(\mathcal{G}_m^k)$ is the estimated time using Equation (16) based on the estimated resources while $t(\mathcal{G}_m^k)$ is the actual measured time. When the whole training process is finished, the average estimation error is obtained by:

$$\overline{err} = \sqrt{\frac{1}{KM}\sum_{k=1}^{M}\sum_{k=1}^{K} err_m^k}. \tag{31}$$

From the results in Table V, we observe that the experiment with $\alpha = 0.9$ achieves the best training performance. However, the value of α is not expected to have a significant impact on the model accuracy and the training time since the resources on each device do not change drastically in our experiments. Hence, when dealing with values that exhibit slight variations around a fixed value, such as those following a uniform or Gaussian distribution, the method of moving average with a parameter of $\alpha = 0.9$ is a feasible approach for estimating the value.

Nevertheless, it is possible that on-device resources may experience significant changes at times. In such cases, we can obtain real-time values of resources through measurement

tools like iperf3 [48] or estimate them using advanced prediction techniques such as link forecasting [49] and short-term resource prediction [50]. As dynamic resource measurement and prediction are not the main focus of this work, readers are referred to related studies [48], [49], [50] for more information.

VI. Related Work

In edge computing, which is a popular application scenario for federated learning (FL), the available resources for model training are typically limited and heterogeneous. In this regard, we provide a review of existing literature on FL, focusing on two key aspects: communication efficiency and computation efficiency.

A. Communication-Efficient Federated Learning

At the network edge, the communication cost often becomes a critical bottleneck for FL systems. This is because training over-parameterized deep learning (DL) models in a distributed manner requires the transmission of a large number of model parameters.

To reduce the amount of transmitted data and speed up distributed model training, many prior researches propose various compression techniques, which can be divided into two categories. (1) *Quantization* compresses the transmitted data by representing each model parameter with a fewer number of bits (*e.g.*, from float32 to float16). For example, Dettmers [51] proposed to use one bit for denoting the sign and 7 bits for representing the exponent and mantissa of each element in model parameters. More extremely, Seide et al. [52] implemented the 1-bit quantization algorithm, where only the elements that are larger than the predefined threshold are quantized as 1 and others are set to 0. To accommodate the unique characteristics in FL, FedPAQ [9] combined parameter quantization with periodic model averaging and partial client participation to further accelerate the training process. (2) *Sparsification* only transmits a subset of the original model parameters. Equipped with error compensation techniques, this method can achieve the same convergence rate as the methods without compression [8]. Apart from random selection of the transmitted parameters [53], some works proposed to select the elements whose absolute values are above a fixed [54] or adaptive [10] threshold. In addition, the combination of sparsification and quantization will further reduce communication cost [55], [56]. For example, CE-FedAvg [55] demonstrated a significant reduction in the total uploaded data by up to 3 times.

Beyond model compression that reduces the volume of transferred data at each round, another widely used method, called local SGD [57], is to reduce communication frequency. In local SGD, global model aggregation is performed after multiple updates rather than each update in traditional distributed SGD algorithms. Several methods [58], [59] propose to adaptively determine optimal aggregation intervals under the constraint of available resources while guaranteeing model performance. Besides, hybrid local SGD (HL-SGD) [60] partitioned edge devices into disjoint clusters and utilized both fast device-to-device communication (within a cluster) and slow device-to-server communication (between clusters and the server) to speed up the training process. Furthermore, cooperative edge-based FedAvg [61] adopted edge servers to achieve accurate and fast training. Specifically, edge servers orchestrate model training among the devices within their coverage and collaborate with other edge servers to learn a shared global model.

These methods only concentrate on reducing communication cost but overlook computation constraints in edge computing. Although BOSE has the ability to save both communication and computation cost, the above methods can be used in conjunction with ours for more aggressive reduction of communication cost.

B. Computation-Efficient Federated Learning

As the computation capabilities of edge devices are constrained and heterogeneous, there are a number of works that aim to improve computation efficiency in FL. We classify these works into two categories.

Local models are parts of the global model [11], [12], [62], [63]. In other words, each device only train a part of the global model, and the aggregation can be performed in the corresponding parameter positions. Wang et al. [11] first divided the original model into multiple partitions and then progressively expanded the model from the first partition to the full model during the training process. Thus, at the early stages, devices only need to train the first several layers. Based on devices' capabilities, some other works [12], [62], [63] generated heterogeneous local submodels from the full global model. Our work belongs to this category. Different from above mentioned works, the proposed BOSE is more flexible in both training stage and inference stage to deal with resource dynamics by utilizing auxiliary classifiers.

Local models are in different architectures [64], [65]. These methods go a step further to allow different model structures and sizes on various devices. To fuse these heterogeneous local models, Lin et al. [64] used unlabeled data or artificially generated examples by GAN [66] to perform ensemble distillation. Afonin et al. [65] formulated the federated distillation problem as a kernel regression on combined local datasets. Based on the theoretical analysis and empirical observation, they argued that the distillation under data heterogeneity significantly degrades model performance in FL and thus it requires an entirely new framework to achieve effective and efficient federated distillation.

VII. Conclusion

In this paper, we have designed and implemented a novel FL system, called BOSE, to accelerate model training on edge devices under resource limitation, heterogeneity and dynamics. Inspired by multi-exit models, we divide a conventional DL model into several blocks, which can be trained independently on devices. Considering on-device resources and blocks' potential for enhancing model performance, BOSE adaptively assigns different blocks for heterogeneous devices, alleviating the impact of stragglers. The experimental results show that BOSE significantly outperforms the baseline FL systems, providing a speedup of $1.4\times\sim3.8\times$ on multiple tasks and DL models without sacrificing model accuracy.

Appendix A
Proof of Lemma 1

With $\Delta_{m,b}^{k} = w_{m,b}^{k} - w_{m,b}^{k-1}, k \geq 2$, we define $\overline{\Delta_{b}^{k}} \triangleq \frac{1}{M}\sum_{m=1}^{M}\Delta_{m,b}^{k}$, which is the average updates of the block b at round k.

By L-smoothness, we obtain:

$$\begin{aligned}
&\mathbb{E}[F(w_b^{k+1})] \\
&\le F(w_b^k) + \left\langle \nabla F(w_b^k), \mathbb{E}[w_b^{k+1} - w_b^k] \right\rangle + \frac{L}{2}\mathbb{E}[\|w_b^{k+1} - w_b^k\|^2] \\
&= F(w_b^k) + \left\langle \nabla F(w_b^k), \mathbb{E}[\eta\overline{\Delta_b^k}] \right\rangle + \frac{L}{2}\mathbb{E}[\|\eta\overline{\Delta_b^k}\|^2] \\
&= F(w_b^k) + \eta \left\langle \nabla F(w_b^k), \mathbb{E}[\overline{\Delta_b^k} + \nabla F(w_b^k) - \nabla F(w_b^k)] \right\rangle \\
&\quad + \frac{L}{2}\eta^2\mathbb{E}[\|\overline{\Delta_b^k}\|^2] \\
&= F(w_b^k) - \eta(\|\nabla F(w_b^k)\|^2 \\
&\quad - \underbrace{\left\langle \nabla F(w_b^k), \mathbb{E}[\overline{\Delta_b^k} + \nabla F(w_b^k)] \right\rangle}_{A_1}) + \frac{L}{2}\eta^2 \underbrace{\mathbb{E}[\|\overline{\Delta_b^k}\|^2]}_{A_2}.
\end{aligned} \tag{32}$$

To bound A_1, we have:

$$\begin{aligned}
&\left\langle \nabla F(w_b^k), \mathbb{E}[\overline{\Delta_b^k} + \nabla F(w_b^k)] \right\rangle \\
&= \left\langle \nabla F(w_b^k), \mathbb{E}\left[-\frac{1}{M}\sum_{m=1}^{M} \nabla F_m(w_{m,b}^k) + \nabla F(w_b^k) \right] \right\rangle \\
&= \left\langle \nabla F(w_b^k), \frac{1}{M}\mathbb{E}\left[\sum_{m=1}^{M} \left(\nabla F(w_b^k) - \nabla F_m(w_{m,b}^k) \right) \right] \right\rangle \\
&\le \frac{1}{2}\|\nabla F(w_b^k)\|^2 + \frac{1}{2M} \underbrace{\mathbb{E}\left[\|\nabla F(w_b^k) - \nabla F_m(w_{m,b}^k)\|^2 \right]}_{A_3},
\end{aligned} \tag{33}$$

where the last inequality follows from that $2\langle \mathbf{a}, \mathbf{b} \rangle \le \|\mathbf{a}\|^2 + \|\mathbf{b}\|^2, \forall \mathbf{a}, \mathbf{b}$ and $\|\sum_{i=1}^n \mathbf{a}_i\|^2 \le n\sum_{i=1}^n \|\mathbf{a}_i\|^2, \forall \mathbf{a}_i$. Furthermore, $\sqrt{A_3}$ can be bounded as follows:

$$\begin{aligned}
&\|\mathbb{E}_{p_{m,b}^k} \nabla F_m(w_b^k) - \nabla F(w_b^k)\| \\
&= \| \int \nabla F_m(w_{m,b}^k) p_{m,b-1}^k(z) dz \\
&\quad - \int \nabla F_m(w_{m,b}^k) p_{b-1}^*(z) dz \| \\
&\le \int \|\nabla F_m(w_{m,b}^k)\| \cdot |p_{m,b-1}^k(z) - p_{b-1}^*(z)| dz \\
&= \int (\|\nabla F_m(w_{m,b}^k)\| \sqrt{|p_{m,b-1}^k(z) - p_{b-1}^*(z)|}) \cdot \\
&\qquad \sqrt{|p_{m,b-1}^k(z) - p_{b-1}^*(z)|} dz \\
&\le \sqrt{\int \|\nabla F_m(w_{m,b}^k)\|^2 |p_{m,b-1}^k(z) - p_{b-1}^*(z)| dz} \cdot \\
&\quad \sqrt{\int |p_{m,b-1}^k(z) - p_{b-1}^*(z)| dz} \\
&= \underbrace{\sqrt{\int \|\nabla F_m(w_{m,b}^k)\|^2 |p_{m,b-1}^k(z) - p_{b-1}^*(z)| dz}}_{A_4} \sqrt{c_{m,b-1}^k},
\end{aligned} \tag{34}$$

where the last inequality comes from Cauchy-Swchartz inequality.

$(A_4)^2$ can be bounded:

$$\begin{aligned}
&\int \|\nabla F_m(w_{m,b}^k)\|^2 |p_{m,b-1}^k(z) - p_{b-1}^*(z)| dz \\
&\le \int \|\nabla F_m(w_{m,b}^k)\|^2 (p_{m,b-1}^k(z) + p_{b-1}^*(z)) dz \\
&= \mathbb{E}_{p_{m,b}^k}[\|F_m(w_{m,b}^k)\|^2] + \mathbb{E}_{p_{b-1}^*}[\|F_m(w_{m,b}^k)\|^2] \\
&\le 2G.
\end{aligned} \tag{35}$$

To bound A_2, we have:

$$\begin{aligned}
\mathbb{E}\left[\|\overline{\Delta_b^k}\|^2\right] &= \mathbb{E}\left[\|\frac{1}{M}\sum_{m=1}^{M} \nabla F_m(w_{m,b}^k)\|^2 \right] \\
&\le \mathbb{E}\left[\frac{1}{M^2}\sum_{m=1}^{M} \|\nabla F_m(w_{m,b}^k)\|^2 \right] \\
&\le \frac{G}{M},
\end{aligned} \tag{36}$$

where the first inequality is due to the fact that $\|\sum_{i=1}^n \mathbf{a}_i\|^2 \le n\sum_{i=1}^n \|\mathbf{a}_i\|^2, \forall \mathbf{a}_i$.

Substituting the inequalities (33)-(36) into inequality (32), we complete the proof:

$$\mathbb{E}[F(w_b^{k+1})] \le F(w_b^k) - \eta(\frac{1}{2}\|\nabla F(w_b^k)\|^2 - \frac{G}{M}c_{b-1}^k) + \frac{G\eta^2 L}{2M}.$$

Appendix B
Proof of Theorem 1

Rearranging Lemma 1, we have:

$$\frac{\eta}{2}\|\nabla F(w_b^k)\|^2 \le F(w_b^k) - F(w_b^{k+1}) + \frac{\eta G}{M}c_{b-1}^k + \frac{G\eta^2 L}{2M}. \tag{37}$$

Furthermore, summing $k = 0, \ldots, K-1$, we obtain:

$$\begin{aligned}
\sum_{k=0}^{K-1} \frac{\eta}{2}\|\nabla F(w_b^k)\|^2 &\le F(w_b^0) - F(w_b^K) \\
&\quad + \frac{\eta G}{M}\sum_{k=0}^{K-1} c_{b-1}^k + \frac{KLG}{2M}\eta^2,
\end{aligned} \tag{38}$$

which implies

$$\frac{1}{K}\sum_{k=0}^{K-1} \|\nabla F(w_b^k)\|^2 \le \frac{2F(w_b^0)}{K\eta} + \frac{2G}{MK}\sum_{k=0}^{K-1} c_{b-1}^k + \frac{LG\eta}{M}.$$

Appendix C
Proof of Corollary 1

By applying the definitions in Equations (27) and (28), we can prove the convergence of the full model:

$$\begin{aligned}
&\frac{1}{K}\sum_{k=0}^{K-1} \|\nabla F(w^k)\|^2 \\
&= \frac{1}{K}\sum_{k=0}^{K-1} \frac{\|\nabla F(w^k)\|^2}{\|\nabla F(w_b^k)\|^2} \|\nabla f(w_b^k)\|^2 \\
&\le q' \frac{1}{K}\sum_{k=0}^{K-1} \|\nabla F(w_b^k)\|^2 \\
&\le \frac{2q'F'}{K\eta} + \frac{2q'Gc'}{M} + \frac{q'LG\eta}{M}.
\end{aligned} \tag{39}$$

REFERENCES

[1] L. Chen, S. Zhou, and J. Xu, "Computation peer offloading for energy-constrained mobile edge computing in small-cell networks," *IEEE/ACM Trans. Netw.*, vol. 26, no. 4, pp. 1619–1632, Aug. 2018.

[2] Y. Liao, Y. Xu, H. Xu, Z. Yao, L. Wang, and C. Qiao, "Accelerating federated learning with data and model parallelism in edge computing," *IEEE/ACM Trans. Netw.*, early access, Aug. 24, 2023, doi: 10.1109/TNET.2023.3299851.

[3] 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. Statist.*, 2017, pp. 1273–1282.

[4] I. Dayan et al., "Federated learning for predicting clinical outcomes in patients with COVID-19," *Nature Med.*, vol. 27, no. 10, pp. 1735–1743, 2021.

[5] D. Leroy, A. Coucke, T. Lavril, T. Gisselbrecht, and J. Dureau, "Federated learning for keyword spotting," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, May 2019, pp. 6341–6345.

[6] P. Goyal et al., "Accurate, large minibatch SGD: Training ImageNet in 1 hour," 2017, *arXiv:1706.02677*.

[7] C. Yang et al., "Characterizing impacts of heterogeneity in federated learning upon large-scale smartphone data," in *Proc. Web Conf.*, Apr. 2021, pp. 935–946.

[8] F. Haddadpour, M. M. Kamani, A. Mokhtari, and M. Mahdavi, "Federated learning with compression: Unified analysis and sharp guarantees," in *Proc. Int. Conf. Artif. Intell. Statist.*, 2021, pp. 2350–2358.

[9] L. Qu, S. Song, and C.-Y. Tsui, "FedDQ: Communication-efficient federated learning with descending quantization," in *Proc. IEEE Global Commun. Conf.*, Dec. 2022, pp. 2021–2031.

[10] P. Han, S. Wang, and K. K. Leung, "Adaptive gradient sparsification for efficient federated learning: An online learning approach," in *Proc. IEEE 40th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Nov. 2020, pp. 300–310.

[11] H.-P. Wang, S. Stich, Y. He, and M. Fritz, "ProgFed: Effective, communication, and computation efficient federated learning by progressive training," in *Proc. Int. Conf. Mach. Learn.*, 2022, pp. 23034–23054.

[12] E. Diao, J. Ding, and V. Tarokh, "HeteroFL: Computation and communication efficient federated learning for heterogeneous clients," in *Proc. Int. Conf. Learn. Represent.*, 2021, pp. 1–24.

[13] A. Li, J. Sun, P. Li, Y. Pu, H. Li, and Y. Chen, "Hermes: An efficient federated learning framework for heterogeneous mobile clients," in *Proc. 27th Annu. Int. Conf. Mobile Comput. Netw.*, Oct. 2021, pp. 420–437.

[14] S. Scardapane, M. Scarpiniti, E. Baccarelli, and A. Uncini, "Why should we add early exits to neural networks?" *Cognit. Comput.*, vol. 12, no. 5, pp. 954–966, Sep. 2020.

[15] T. Bolukbasi, J. Wang, O. Dekel, and V. Saligrama, "Adaptive neural networks for efficient inference," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 527–536.

[16] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*.

[17] Y. Kang et al., "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," *ACM SIGPLAN Notices*, vol. 52, no. 4, pp. 615–629, May 2017.

[18] X. Xiao, T. B. Mudiyanselage, C. Ji, J. Hu, and Y. Pan, "Fast deep learning training through intelligently freezing layers," in *Proc. Int. Conf. Internet Things (iThings), IEEE Green Comput. Commun. (GreenCom), IEEE Cyber, Phys. Social Comput. (CPSCom), IEEE Smart Data (SmartData)*, Jul. 2019, pp. 1225–1232.

[19] C. Chen et al., "Communication-efficient federated learning with adaptive parameter freezing," in *Proc. IEEE 41st Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jul. 2021, pp. 1–11.

[20] S. Lee, T. Zhang, C. He, and S. Avestimehr, "Layer-wise adaptive model aggregation for scalable federated learning," 2021, *arXiv:2110.10302*.

[21] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.

[22] Y. Jiang et al., "Model pruning enables efficient federated learning on edge devices," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, Apr. 25, 2022, doi: 10.1109/TNNLS.2022.3166101.

[23] J. Liu, Y. Xu, H. Xu, Y. Liao, Z. Wang, and H. Huang, "Enhancing federated learning with intelligent model migration in heterogeneous edge computing," in *Proc. IEEE 38th Int. Conf. Data Eng. (ICDE)*, May 2022, pp. 1586–1597.

[24] Y. Liao, Y. Xu, H. Xu, L. Wang, and C. Qian, "Adaptive configuration for heterogeneous participants in decentralized federated learning," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, May 2023, pp. 1–10.

[25] Cisco. (2020). *Cisco Annual Internet Report (2018–2023), White Paper*. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.pdf

[26] J. Liu et al., "Adaptive asynchronous federated learning in resource-constrained edge computing," *IEEE Trans. Mobile Comput.*, vol. 22, no. 2, pp. 674–690, Feb. 2023.

[27] T. Ouyang, Z. Zhou, and X. Chen, "Follow me at the edge: Mobility-aware dynamic service placement for mobile edge computing," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 10, pp. 2333–2345, Oct. 2018.

[28] E. Belilovsky, M. Eickenberg, and E. Oyallon, "Decoupled greedy learning of CNNs," in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 736–745.

[29] P. Warden, "Speech commands: A dataset for limited-vocabulary speech recognition," 2018, *arXiv:1804.03209*.

[30] Y. Guo, H. Shi, A. Kumar, K. Grauman, T. Rosing, and R. Feris, "SpotTune: Transfer learning through adaptive fine-tuning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 4800–4809.

[31] K. Berestizshevsky and G. Even, "Dynamically sacrificing accuracy for reduced computation: Cascaded inference based on softmax confidence," in *Proc. Int. Conf. Artif. Neural Netw.* Cham, Switzerland: Springer, 2019, pp. 306–320.

[32] S. Teerapittayanon, B. McDanel, and H. T. Kung, "BranchyNet: Fast inference via early exiting from deep neural networks," in *Proc. 23rd Int. Conf. Pattern Recognit. (ICPR)*, Dec. 2016, pp. 2464–2469.

[33] M. Phuong and C. Lampert, "Distillation-based training for multi-exit architectures," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 1355–1364.

[34] F. Lai, X. Zhu, H. V. Madhyastha, and M. Chowdhury, "Oort: Efficient federated learning via guided participant selection," in *Proc. 15th USENIX Symp. Operating Syst. Design Implement.*, 2021, pp. 19–35.

[35] J. Shin, Y. Li, Y. Liu, and S.-J. Lee, "FedBalancer: Data and pace control for efficient federated learning on heterogeneous clients," in *Proc. 20th Annu. Int. Conf. Mobile Syst., Appl. Services*, Jun. 2022, pp. 436–449.

[36] C. Li, X. Zeng, M. Zhang, and Z. Cao, "PyramidFL: A fine-grained client selection framework for efficient federated learning," in *Proc. 28th Annu. Int. Conf. Mobile Comput. Netw.*, Oct. 2022, pp. 158–171.

[37] S. Krithivasan, S. Sen, S. Venkataramani, and A. Raghunathan, "Accelerating DNN training through selective localized learning," *Frontiers Neurosci.*, vol. 15, p. 759807, 2022.

[38] A. Brock, T. Lim, J. M. Ritchie, and N. Weston, "FreezeOut: Accelerate training by progressively freezing layers," 2017, *arXiv:1706.04983*.

[39] B. Singh, S. De, Y. Zhang, T. Goldstein, and G. Taylor, "Layer-specific adaptive learning rates for deep networks," in *Proc. IEEE 14th Int. Conf. Mach. Learn. Appl. (ICMLA)*, Dec. 2015, pp. 364–368.

[40] A. Mohtashami, M. Jaggi, and S. Stich, "Masked training of neural networks with partial gradients," in *Proc. Int. Conf. Artif. Intell. Statist.*, 2022, pp. 5876–5890.

[41] A. Paszke et al., "PyTorch: An imperative style, high-performance deep learning library," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 32, 2019, pp. 1–12.

[42] A. Krizhevsky, "Learning multiple layers of features from tiny images," M.S. thesis, Dept. Comput. Sci., Citeseer, Univ. Toronto, Toronto, ON, Canada. [Online]. Available: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf

[43] Oldpan. (2023). *PyTorch-Memory-Utils*. [Online]. Available: https://github.com/Oldpan/Pytorch-Memory-Utils

[44] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*.

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

[46] H. Wang, Z. Kaplan, D. Niu, and B. Li, "Optimizing federated learning on non-IID data with reinforcement learning," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, Jul. 2020, pp. 1698–1707.

[47] K. Bonawitz et al., "Towards federated learning at scale: System design," in *Proc. Mach. Learn. Syst.*, vol. 1, 2019, pp. 374–388.

[48] B. A. Mah et al. (2023). *IPERF3: A TCP, UDP, and SCTP Network Bandwidth Measurement Tool*. [Online]. Available: https://github.com/esnet/iperf

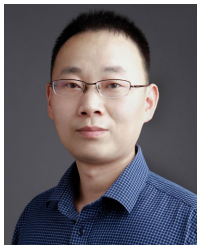
[49] C. Yue, R. Jin, K. Suh, Y. Qin, B. Wang, and W. Wei, "LinkForecast: Cellular link bandwidth prediction in LTE networks," *IEEE Trans. Mobile Comput.*, vol. 17, no. 7, pp. 1582–1594, Jul. 2018.

[50] T. Panapongpakorn and D. Banjerdpongchai, "Short-term load forecast for energy management systems using time series analysis and neural network method with average true range," in *Proc. 1st Int. Symp. Instrum., Control, Artif. Intell., Robot. (ICA-SYMP)*, Jan. 2019, pp. 86–89.

[51] T. Dettmers, "8-bit approximations for parallelism in deep learning," 2015, *arXiv:1511.04561*.
[52] F. Seide, H. Fu, J. Droppo, G. Li, and D. Yu, "1-bit stochastic gradient descent and its application to data-parallel distributed training of speech DNNs," in *Proc. Interspeech*, Sep. 2014, pp. 1–5.
[53] J. Wangni, J. Wang, J. Liu, and T. Zhang, "Gradient sparsification for communication-efficient distributed optimization," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 31, 2018, pp. 1299–1309.
[54] N. Ström, "Scalable distributed DNN training using commodity GPU cloud computing," in *Proc. Interspeech*, 2015. [Online]. Available: https://www.amazon.science/publications/scalable-distributed-dnn-training-using-commodity-gpu-cloud-computing
[55] J. Mills, J. Hu, and G. Min, "Communication-efficient federated learning for wireless edge intelligence in IoT," *IEEE Internet Things J.*, vol. 7, no. 7, pp. 5986–5994, Jul. 2020.
[56] F. Sattler, S. Wiedemann, K.-R. Müller, and W. Samek, "Robust and communication-efficient federated learning from non-i.i.d data," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 9, pp. 3400–3413, Sep. 2020.
[57] S. U. Stich, "Local SGD converges fast and communicates little," 2018, *arXiv:1805.09767*.
[58] 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.
[59] Y. Xu, Y. Liao, H. Xu, Z. Ma, L. Wang, and J. Liu, "Adaptive control of local updating and model compression for efficient federated learning," *IEEE Trans. Mobile Comput.*, vol. 22, no. 10, pp. 5675–5689, Oct. 2023.
[60] Y. Guo, Y. Sun, R. Hu, and Y. Gong, "Hybrid local SGD for federated learning with heterogeneous communications," in *Proc. Int. Conf. Learn. Represent.*, 2021, pp. 1–42.
[61] Z. Zhang, Z. Gao, Y. Guo, and Y. Gong, "Scalable and low-latency federated learning with cooperative mobile edge networking," *IEEE Trans. Mobile Comput.*, early access, Oct. 25, 2022, doi: 10.1109/TMC.2022.3216837.
[62] A. Shamsian, A. Navon, E. Fetaya, and G. Chechik, "Personalized federated learning using hypernetworks," in *Proc. Int. Conf. Mach. Learn.*, 2021, pp. 9489–9502.
[63] J. Hong, H. Wang, Z. Wang, and J. Zhou, "Efficient split-mix federated learning for on-demand and *in-situ* customization," 2022, *arXiv:2203.09747*.
[64] T. Lin, L. Kong, S. U. Stich, and M. Jaggi, "Ensemble distillation for robust model fusion in federated learning," in *Proc. NIPS*, vol. 33, 2020, pp. 2351–2363.
[65] A. Afonin and S. P. Karimireddy, "Towards model agnostic federated learning using knowledge distillation," 2021, *arXiv:2110.15210*.
[66] I. Goodfellow et al., "Generative adversarial networks," *Commun. ACM*, vol. 63, no. 11, pp. 139–144, 2020.

Lun Wang received the B.S. degree from the University of Electronic Science and Technology of China in 2019. He is currently pursuing the Ph.D. degree with the School of Computer Science and Technology, University of Science and Technology of China. His research interests include mobile edge computing, federated learning, and distributed machine learning.

Yang Xu (Member, IEEE) received the B.S. degree from the Wuhan University of Technology in 2014 and the Ph.D. degree in computer science and technology from the University of Science and Technology of China in 2019. He is currently an Associate Researcher with the School of Computer Science and Technology, University of Science and Technology of China. His research interests include ubiquitous computing, deep learning, and mobile edge computing.



Hongli Xu (Member, IEEE) received the B.S. degree in computer science and the Ph.D. degree in computer software and theory from the University of Science and Technology of China, China, in 2002 and 2007, respectively. He is currently a Professor with the School of Computer Science and Technology, University of Science and Technology of China. He has published more than 100 papers in famous journals and conferences, including the IEEE/ACM TRANSACTIONS ON NETWORKING, IEEE TRANSACTIONS ON MOBILE COMPUTING, IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, Infocom, and ICNP. He holds more than 30 patents. His main research interests include software-defined networks, edge computing, and the Internet of Things. He was awarded the Outstanding Youth Science Foundation of NSFC in 2018. He received the best paper award or the best paper candidate in several famous conferences.

Zhida Jiang received the B.S. degree from the Hefei University of Technology in 2019. He is currently pursuing the Ph.D. degree with the School of Computer Science and Technology, University of Science and Technology of China (USTC). His research interests include mobile edge computing and federated learning.

Min Chen (Student Member, IEEE) received the Ph.D. degree from the School of Computer Science and Technology, University of Science and Technology of China, in 2023. He is currently a Senior Engineer with Huawei Cloud Computing Technology Company Ltd. His research interests include cloud-native systems, edge computing, and distributed machine learning.

Wuyang Zhang (Member, IEEE) received the Ph.D. degree from WINLAB, Rutgers University. He is currently a Senior Research Scientist with Meta Inc. His research interests include distributed machine learning systems and edge computing. He was a recipient of the IEEE Communications Society Phoenix ISS Award.

Chen Qian (Senior Member, IEEE) received the B.Sc. degree in computer science from Nanjing University in 2006, the M.Phil. degree in computer science from The Hong Kong University of Science and Technology in 2008, and the Ph.D. degree in computer science from The University of Texas at Austin in 2013. He is currently an Assistant Professor with the Department of Computer Science and Engineering, University of California at Santa Cruz. He has published more than 60 research papers in highly competitive conferences and journals. His research interests include computer networking, network security, and the Internet of Things. He is a member of ACM.