Hierarchical and Heterogeneous Federated Learning via a Learning-on-Model Paradigm

Leming Shen, *Student Member, IEEE,* Qiang Yang, *Member, IEEE,* Kaiyan Cui, *Member, IEEE,* Yuanqing Zheng, *Senior Member, IEEE,* Xiao-Yong Wei, *Senior Member, IEEE,* Jianwei Liu, Jinsong Han, *Senior Member, IEEE*

***Abstract*—Federated Learning (FL) collaboratively trains a shared global model without exposing clients' private data. In practical FL systems, clients (e.g., smartphones and wearables) typically have disparate system resources. Traditional FL, however, adopts a one-size-fits-all solution, where a homogeneous large model is sent to and trained on each client. This method results in an overwhelming workload for less capable clients and starvation for others. To tackle this, we propose *FedConv*, a client-friendly FL framework, minimizing the system overhead on resource-constrained clients by providing heterogeneous customized sub-models. *FedConv* features a novel *learning-on-model* paradigm that learns the parameters of heterogeneous sub-models via *convolutional compression*. To aggregate heterogeneous sub-models, we propose *transposed convolutional dilation* to convert them back to large models with a unified size while retaining personalized information. The compression and dilation processes, transparent to clients, are tuned on the server using a small public dataset. We further propose a *hierarchical and clustering-based local training* strategy for enhanced performance. Extensive experiments on six datasets show that *FedConv* outperforms state-of-the-art FL systems in terms of model accuracy (by more than 35% on average), computation and communication overhead (with 33% and 25% reduction, respectively).**

***Index Terms*—Federated learning, Model heterogeneity, Model compression, Personalization**

I. Introduction

FEDERATED Learning (FL) allows mobile devices to collaboratively train a shared global model without exposing their private data [1], [2], [3], [4], [5]. In each communication round, clients keep their private data locally and only upload their model parameters or gradients to a server after local training. The server then orchestrates model aggregation and updates the global model for the next round [6], [7], [8], [9].

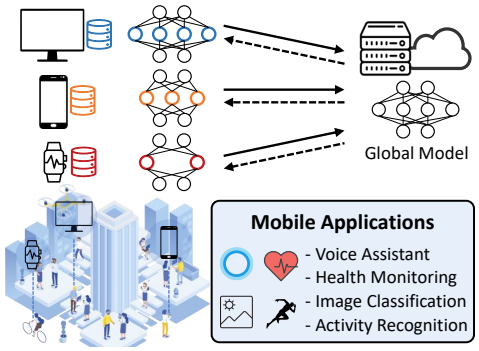


Fig. 1. Heterogeneous models in federated learning.

In conventional FL, all clients share the same model architecture. In practice, however, clients have diverse computation and system resources. For example, high-end PCs have more resources and computation powers than wearables (Fig. 1). Therefore, the size of a global model is upper-bounded by the clients with the least resources. Such a one-size-fits-all solution often leads to sub-optimal performance. Moreover, clients with more resources suffer from starvation when waiting for weaker clients in synchronized FL [10]. To make full use of more powerful clients while accommodating those with limited resources, it is necessary to develop an FL system that supports heterogeneous models with varied parameter sizes that best fit all clients with diverse resources [11], [12], [13], [14], [15].

Existing solutions that generate distinct models mainly include knowledge distillation (KD) [16], parameter sharing [17], and model pruning [18]. KD distills the knowledge from heterogeneous client models to a global model for aggregation. Yet, it imposes additional compute overhead on clients as they must first train on public data and then transfer knowledge via private data [16]. Parameter-sharing strategies distribute different regions of a global model as sub-models to different clients. However, some sub-models can only be trained on a small portion of the dataset. Model pruning methods adopt channel or filter level pruning to generate sparse sub-models. However, they suffer from information loss due to the removal of entire channels or filters (§ II-A). Moreover, to determine the pruning structure, clients need to receive the large global model from the server and then perform the pruning operation locally, increasing the overhead of clients.

This work is supported by Hong Kong GRF Grant No. 15211924 and 15206123, NSFC (Grant No. 62402238, 62372314, U21A20462, and 62372400), "Pioneer" and "Leading Goose" R&D Program of Zhejiang under grant No. 2024C03287, and the Postdoctoral Fellowship Program of CPSF under grant No. GZC20241488.

L. Shen and Y. Zheng are with the Department of Computing, The Hong Kong Polytechnic University, Hong Kong, China. E-mail: leming.shen@connect.polyu.hk, csyqzheng@comp.polyu.edu.hk.

Q. Yang is with the Department of Computer Science and Technology, University of Cambridge, UK, and also with the Department of Computing, The Hong Kong Polytechnic University, Hong Kong, China. E-mail: qiang.yang@cl.cam.ac.uk.

K. Cui is with the School of Computer Science, Nanjing University of Posts and Telecommunications, China, and also with the Department of Computing, The Hong Kong Polytechnic University, Hong Kong, China. E-mail: kaiyan.cui@njupt.edu.cn.

X. Wei is with the Department of Computer Science, Sichuan University, China, and also with the Department of Computing, The Hong Kong Polytechnic University, Hong Kong, China. E-mail: cswei@scu.edu.cn.

J. Liu is with the College of Information Science and Electronic Engineering, Zhejiang University, China, and also with the School of Information and Electrical Engineering, Hangzhou City University, China. E-mail: jianweiliu@zju.edu.cn.

J. Han is with the College of Computer Science and Technology, Zhejiang University, China. E-mail: hanjinsong@zju.edu.cn.

Corresponding author: Yuanqing Zheng.

Ideally, the heterogeneous sub-models should retain the global model information in a way that enables them to be efficiently sent to and trained on resource-constrained clients without any extra overhead. To this end, we propose *FedConv*, a client-friendly FL framework for heterogeneous models based on a new *learning-on-model* paradigm. *The key insight is that convolution, a technique to extract effective features from data, can also compress large models to preserve crucial information.* In *FedConv*, the server performs *convolutional compression* on the global model to learn parameters of diverse sub-models according to clients' resource budgets. Clients directly train on the compressed sub-models as in traditional FL without model decompression. In model aggregation, the server first uses transposed convolution (TC) to transform heterogeneous client models into large models that have the same size as the global model. Then, the server assigns different learned *weight vectors* to these dilated models and aggregates them. *FedConv* optimizes the model compression, dilation, and aggregation processes by leveraging a small dataset on the server that can be obtained via crowdsourcing, or voluntarily shared by users without compromising their privacy. Therefore, our system does not incur extra communication or computation overhead for resource-constrained clients.

To deliver a practical system, we address three key technical challenges: 1) How to learn heterogeneous sub-model parameters via convolution while retaining the global model's prediction capability? To tackle this, we formulate the compression process as a training task. By iteratively fine-tuning the convolution operations, heterogeneous sub-models can be learned effectively and achieve a performance comparable to the global model. 2) How to enhance the personalization performance during client-side local training and server-side aggregation? We propose a *hierarchical and clustering-based local training* strategy that enables parameter exchange among clients with similar data distributions. Additionally, during aggregation, we apply separate TC operations on each client's model parameters and learn a set of dilated models, which further inherit their personalized information into the global model. 3) How to aggregate the dilated models with imbalanced contributions of heterogeneous federated clients? Client models are trained on the non-independent and identically distributed (non-IID) data, directly averaging [19] these large models will lead to performance degradation. To solve it, we set different learnable *weight vectors* for the dilated models. Through a tuning process, the server can learn the relative importance of each model and orchestrate the final aggregation.

We implement *FedConv*[1] based on a user-friendly FL framework (Flower [20]) with two representative FL tasks (image classification and human activity recognition). We evaluate *FedConv* on six public datasets and compare its performance with eight baselines. The experiments show that *FedConv* outperforms the SOTA in terms of inference accuracy (by more than 35% on average), memory, and communication cost (with 21% and 25% reduction, respectively). Besides, *FedConv* substantially reduces the computation overhead for federated clients and saves the total training time. In summary, we make the following key contributions:

- To the best of our knowledge, *FedConv* is the first compression method based on convolution operations. This paradigm can not only compress the global model effectively but also preserve crucial information without imposing extra burden on resource-constrained mobile clients.
- *FedConv* handles heterogeneous models with new technologies. Specifically, we propose a *convolutional compression* module to compress the global model and generate heterogeneous sub-models via our *learning-on-model* paradigm. We create a *hierarchical and clustering-based local training* strategy to enhance the personalization performance of heterogeneous clients. We design a *transposed convolutional dilation* method to obtain models with uniform sizes and use *weighted average aggregation* to balance clients' contributions for final aggregation.
- We evaluate *FedConv* via comprehensive evaluations with heterogeneous mobile devices. The results demonstrate the superior performance of *FedConv* in terms of both inference accuracy and resource efficiency.

II. Motivation

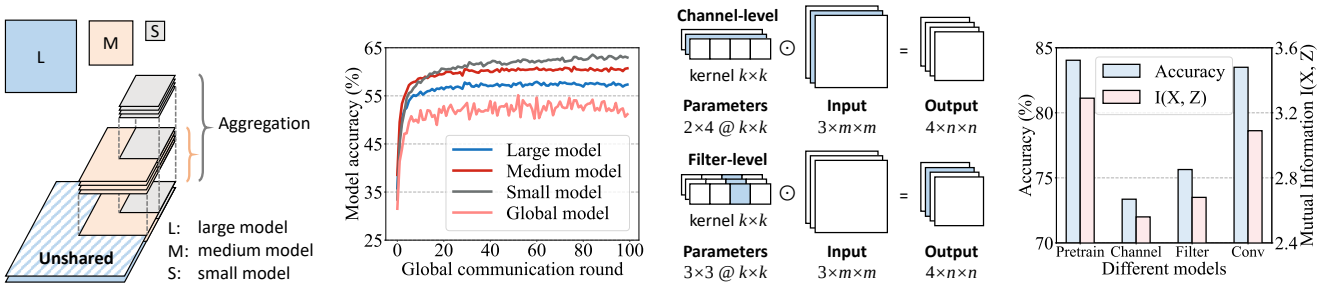
In this section, we analyze SOTA works (parameter sharing and model pruning) to motivate our work. KD-based methods incur heavy overhead on clients (§ VI-C), which is not suitable for resource-constrained mobile devices.

A. Limitations of Existing Solutions

Imbalanced Parameter Sharing. As a representative work, HeteroFL [17] makes clients share different regions of the global model. In Fig. 2(a), the shared portions (the overlapped parts) are fixed, and parameters are aggregated only from clients that hold them, missing information from other clients. To demonstrate its impact, we train a ResNet18 model [21] on the CIFAR10 dataset [22] and find that smaller models outperform larger models (Fig. 2(b)) due to their exposure to a larger volume of data held by more clients. Besides, the global model exhibits instability and even performs worse than the large model due to unbalanced aggregation. Thus, this scheme will lead to imbalanced performance among clients and unexpected performance degradation [23]. FedRolex [24] proposes a dynamic sharing scheme that enables sub-models to share different parts of the global model's parameters via multiple rolling windows, ensuring that the aggregated parameters are evenly trained. However, as different clients contribute distinct parts, the aggregated parameters comprise mixed windows from the diverse sub-models, leading to a distorted distribution of the global model's parameters with degraded performance and a longer convergence time [25].

Information Loss and Client Workload in Model Pruning. Model pruning can be categorized into two types (Fig. 2(c)). 1) Channel-level pruning removes some input channels from the model parameters, where the corresponding channels of the input data are also excluded from training. 2) Filter-level pruning discards some output channels (filters), generating fewer feature maps. Consequently, these two schemes suffer from information loss as they not only discard some input data channels or feature maps but also remove certain weights or connections [18]. To study the information loss, we apply

[1]The code is available at https://github.com/lemingshen/FedConv.



(a) The parameter sharing scheme (b) Performance difference (c) Two pruning schemes (d) Information loss

Fig. 2. The parameter sharing and pruning scheme with limitations (the pruned part is colored blue in (c)).

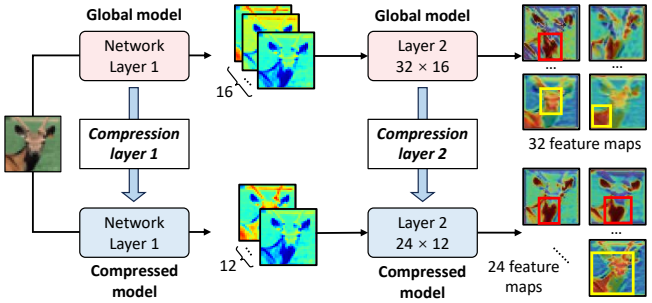


Fig. 3. Convolutional compression process.

these two pruning methods to the pre-trained ResNet18 model based on the parameter magnitude ranking and measure the mutual information (MI) $\mathrm{I}(X, Z)$, which quantifies the amount of information that can be inferred from X after observing Z [26]. We find that the MI between the parameters of the ResNet18 model and itself is 3.29, whereas the MI between the parameters of the pre-trained model and the channel-pruned model is reduced to 2.56, with an accuracy drop from 84.04% to 73.36%. Similarly, the accuracy of the filter-pruned model drops to 75.64% while MI is 2.68. This indicates information loss due to pruning. Moreover, existing pruning-based methods typically require the server to transmit all the parameters of a global model to clients, and perform model pruning at resource-constrained clients, which incurs high communication and computation overhead for clients.

B. Model Compression via Convolution

Ideally, a compression method should minimize the information loss of model parameters to retain the performance without posing extra system overhead on resource-constrained clients. To this end, we propose a novel *convolutional compression* technique that applies convolution operations on the global model parameters to generate the parameters of heterogeneous sub-models while preserving crucial global model information (*e.g.*, parameter distributions and patterns). Our studies find that by applying refined convolution operations via various receptive fields [27], the sub-model can inherit spatial and hierarchical parameter patterns from the global model. These receptive fields selectively determine which parameter information should be retained after convolution. Hence, the generated sub-models can also extract valuable features from the input, similar to the features extracted by the global model.

To demonstrate that a compressed model generated by *convolutional compression* can effectively extract features from the input data, we compress the pre-trained model described in § II-A at a shrinkage ratio of 0.75. We then select the top-4 and top-3 feature maps with the highest importance outputted by a convolutional layer (measured by IG [28]) from the large model and the sub-model, respectively. As shown in Fig. 3, both the large model and the sub-model can learn and focus on the key features (*e.g.*, the deer's body, head, and horn). Moreover, compared with the large model, the first two feature maps from the sub-model pay more attention (deeper color) to the deer's body and ears. The third feature map can be regarded as a fusion of the last two feature maps from the large model, as it focuses on both the body and head of the deer. This observation indicates that the feature extraction capability of the large model can be effectively preserved and transferred to the sub-model via *convolutional compression*. Besides, the accuracy of the sub-model only decreases by 0.19% and the mutual information between the parameters of the large model and the sub-model is 3.09 (Fig. 2(d)), which is much higher than that of the pruned model. This indicates that our proposed *convolutional compression* method can effectively minimize information loss after model compression.

To refine the compression process, same as existing knowledge distillation-based FL works [16], [29] that use server-side data during FL training, we also maintain a small publicly available dataset on the server to fine-tune the compression process (§ IV-A). The server-side data can be crowdsourced by clients and volunteers who are willing to share their data, or collected from public datasets. By iteratively refining the compression process using the server-side data that has the same task (*e.g.*, image classification) to the clients, the server can gradually gain a comprehensive global view [30] of the entire FL process and thus transfer more general information from the server to heterogeneous clients [31]. Note that same as conventional FL schemes, clients do not need to share their data and only focus on local training.

III. Framework Overview

FedConv comprises four main modules (Fig. 4): *convolutional compression* (§ IV-A), *hierarchical and clustering-based Local training* (§ IV-B), *transposed convolutional dilation* (§ IV-C), and *weighted average aggregation* (§ IV-D).

The server first initializes a global model with an estimated memory requirement and records a set of shrinkage ratios (SR) reported by each client based on their resource profiles (①). In the first communication round, the server pre-trains the global model for several epochs with a server-side dataset to gain a better global view of the data distribution. Based on the SRs, a set of fine-tuned *convolution parameters* are then used to compress the global model with the *convolutional compression* module, and generate heterogeneous sub-models (②). Next, the server sends the heterogeneous sub-models to corresponding federated edge servers (③), each managing a group of clients with the same SR value. The edge server will distribute the parameters to their corresponding clients.

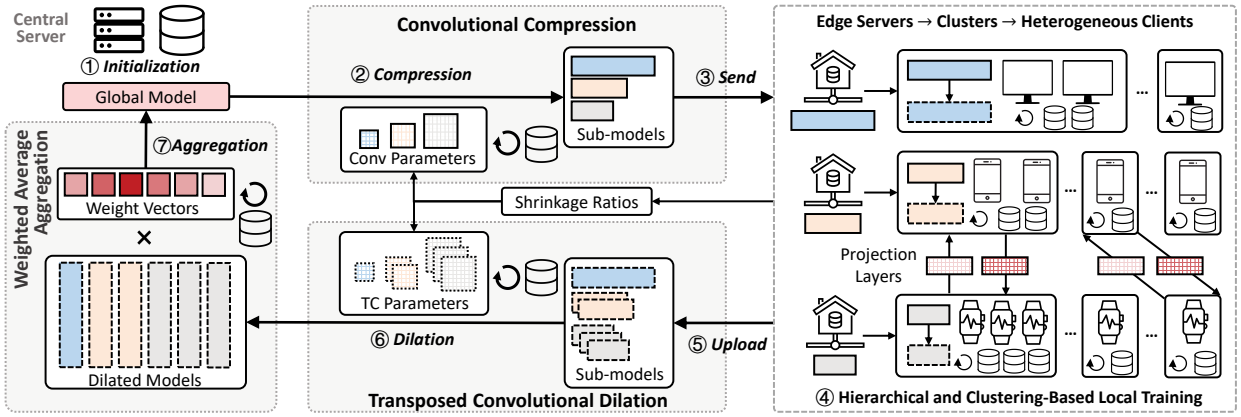


Fig. 4. Framework architecture of *FedConv*.

Then, with the *hierarchical and clustering-based local training* module, each edge server further groups its clients into several clusters based on the similarity of their local data distributions. Clients will perform several epochs of local training and exchange intermediate outputs with clients from other clusters to enhance their personalization performance (④). The updated parameters will then be uploaded to the server (⑤) through the edge servers. After that, the server performs the *transposed convolutional dilation*, where different *transposed convolution parameters* are used to dilate the sub-models to a set of large models that have the same size as the global model (⑥). Finally, the server applies the *weighted average aggregation* to aggregate the dilated models with the learned weights(⑦).

In *FedConv*, the compression and dilation operations are transparent to clients and performed by the powerful server, which can be seamlessly integrated into conventional FL systems where clients only need to perform local training.

IV. Framework Design

A. Convolutional Compression

This module utilizes a set of convolutional layers (termed as *compression layers*) to compress the global model and generate heterogeneous sub-models. After feeding the global model parameters to the *compression layers*, the generated sub-model parameters are smaller and output fewer feature maps (Fig. 3). We use the server-side data to iteratively tune *convolution parameters* (the parameters of *compression layers*) until sub-models achieve comparable performance to the global model. As such, the parameter information from the global model is inherited with a comprehensive perspective. Thus, they are able to extract general features and can be further updated by clients to fit their local data for personalization enhancement.

Convolution Configurations. To determine sub-model sizes, clients first specify their shrinkage ratios (SRs). Specifically, the server first broadcasts the global model size to all clients. Next, each client determines an appropriate SR for its corresponding sub-model to meet the resource budget[2] (*e.g.*, memory, bandwidth). The SRs are then sent back to the server, which accordingly determines corresponding configurations of the *compression layers* (input channel in, output channel out, kernel size (k_1, k_2), stride s, and padding p) so that the sizes of sub-models match with expected SRs. Taking convolution layers as an example, in the upper part of Fig. 5, a convolutional layer in the global model has 16 input and 32 output channels with a 3×3 kernel. Regarding each element in the kernel as a single unit, we can reshape the weight from (32,16,3,3) to 9×(1,32,16). Suppose the SR is 0.75, the shape of the weight should be 9×(1,24,12) after compression. Thus, we use nine separate 2D convolutional layers (*i.e.*, *compression layers*) to compress the reshaped weight. The configuration[3] of each *compression layer* is $\text{Conv}\langle in{=}1, out{=}1, k{=}(9,5), s{=}1, p{=}0\rangle$. This convolution-based process can also be applied to compress other types of layers by properly adjusting the configurations. Note that the input channels of the first layer are not compressed, ensuring that all channels of the raw data can be fed into the sub-model. Similarly, the output channels of the last layer are also uncompressed, ensuring that the sub-models and the global model have the same prediction task.

Convolution Parameter Fine-tuning. Next, we fine-tune the *convolution parameters* to generate sub-models that inherit parameter information from the global model with comparable performance. We use the server-side data to iteratively adapt the *convolution parameters* by minimizing the loss between the ground truth and the prediction result of the sub-model:

$$\begin{aligned} &\min_{\boldsymbol{w}^i_{Conv,l}} \textstyle\sum_x \mathcal{L}(f(x; \boldsymbol{W}^i_{G,l} \odot \boldsymbol{w}^i_{Conv,l}), y), \\ &s.t.\ \forall\, l \in \{1, 2, \cdots, L\}, \forall\, (x, y) \in \mathcal{D}, \forall\, i \in \{1, 2, \cdots, I_l\}. \end{aligned} \tag{1}$$

where $\mathcal{L}$ is the Cross-entropy loss [33], $f(\cdot)$ is the forward function of the compressed model, (x, y) is the data and the corresponding label from server-side data $\mathcal{D}$, $\boldsymbol{W}^i_{G,l}$ is the i-th weight matrix of the l-th layer in the global model, I_l is the number of parameter matrices of the l-th layer, $\boldsymbol{w}^i_{Conv,l}$ is the *convolution parameters* for compression, and $\odot$ is the convolution operation. To fine-tune *convolution parameters*, the compressed sub-model ($\boldsymbol{W}^i_{G,l} \odot \boldsymbol{w}^i_{Conv,l}$) is first evaluated on the server-side data. Specifically, taking the l-th layer from the global model as an example, the forward process is:

$$\begin{aligned} w_{c,l} &= \boldsymbol{W}_{G,l} \odot w_{Conv,l} \\ z_l &= w_{c,l} \cdot \boldsymbol{X}_l \\ L &= \mathcal{L}(\hat{y}, y) \end{aligned} \tag{2}$$

where $w_{Conv,l}$ is the *convolution parameters* to compress the weight matrix $\boldsymbol{W}_{G,l}$ of the l-th layer from the global model, $w_{c,l}$ is the compressed weight, $\boldsymbol{X}_l$ is the input data for the l-th layer, and z_l is the output generated by the compressed weight. Cascaded with multiple such compressed layers, the compressed model generates an output, denoted as $\hat{y}$. We then compute the loss, L, based on $\hat{y}$ and the ground truth label y. Next, gradients of the *convolution parameters* are calculated

[2]Resource profiling can be performed by existing tools, *e.g.*, nn-Meter [32].

[3]By default, we set the stride and padding as one and zero, respectively.

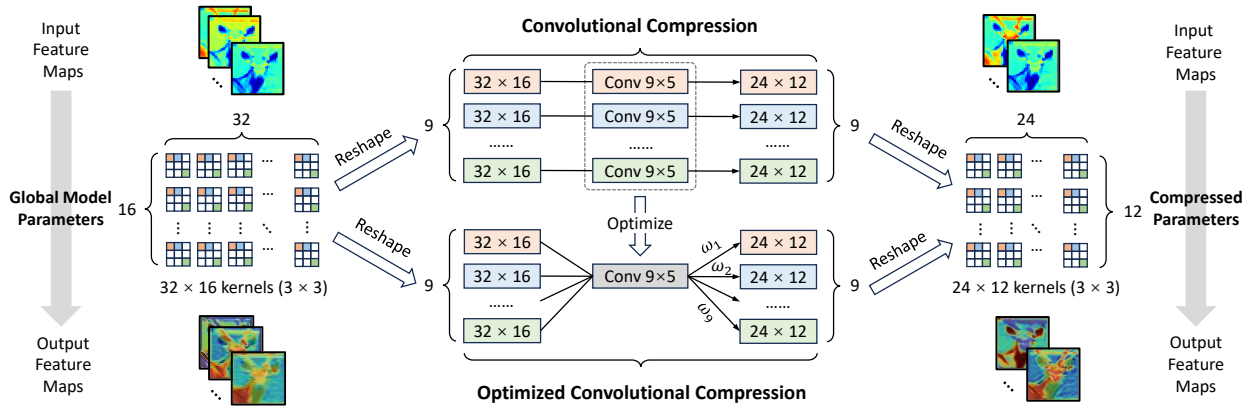
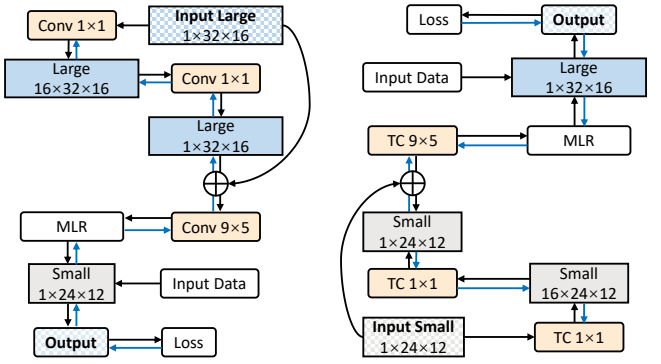


Fig. 5. The convolutional compression process and the optimization mechanism.



(a) The convolution process (b) The TC process

Fig. 6. Convolution/TC process (black arrow: forward, blue arrow: backward, blue box: large parameters, grey box: small parameters, orange: Conv/TC).

via back-propagation using the chain rule:

$$\begin{aligned}\delta^{(L)} &= \frac{\partial L}{\partial \hat{y}} \\ \delta^{(w_{c,l})} &= \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_{c,l}} \\ \delta^{(w_{Conv,l})} &= \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_{c,l}} \cdot \frac{\partial w_{c,l}}{\partial w_{Conv,l}}\end{aligned} \tag{3}$$

where $\delta^{(L)}$ is the gradient of loss with respect to output, $\delta^{(w_{c,l})}$ is the gradient of compressed weight, and $\delta^{(w_{Conv,l})}$ is the gradient of *convolution parameters*, which are further updated:

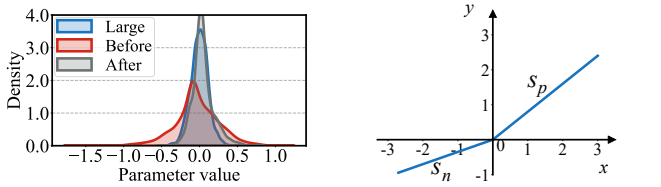
$$w_{Conv,l} = w_{Conv,l} - \alpha \cdot \delta^{(w_{Conv,l})} \tag{4}$$

where α is the learning rate. During iterative fine-tuning, the *convolution parameters* $\boldsymbol{w}^i_{Conv,l}$ is updated while others (*i.e.*, the parameters of the global model and the sub-model) are frozen. Through such a back-propagation process, the tuned *convolution parameters* effectively compress the global model without discarding important parameter information.

Remarks. In *convolutional compression*, the server fine-tunes *convolution parameters* to learn sub-model parameters, thus preserving the global model's crucial parameter information and prediction capability. Such a *learning-on-model* paradigm fundamentally differs from traditional *learning-on-data* methods. Specifically, *learning-on-data* methods take raw data as input and train a model to extract features, while our *learning-on-model* paradigm takes model parameters as input and uses *compression layers* to generate sub-model parameters. The parameters of *compression layers* are tuned by minimizing the loss between sub-model outputs and ground-truth labels.

Challenges. Several practical challenges emerge during this compression process. We use a trained model on the MNIST [34] dataset (with 99.04% accuracy) as an example to show how we address these challenges and the progress we make.

(1) Information Loss. After tuning *convolution parameters*, we find that the amount of parameter information in the sub-model



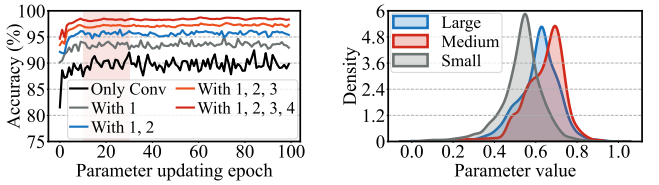
(a) Parameter distribution (b) Modified activation function

Fig. 7. (a) After using the MLR, the sub-model's parameter distribution becomes similar to the large global model (blue curve); (b) The MLR function.

inferred from the global model is still low (the MI between the parameters of the global model and the sub-model is only 0.84). This is because a single *compression layer* may fail to capture fine-grained parameter information, leading to a lower accuracy (90.2%). To address this issue, we add two 1x1 convolutional layers before compression (Fig. 6(a)). The first layer increases the number of output channels to 16, capturing diverse and complex parameter information in the global model. The second layer decreases the channel number back to one, fusing information from different channels and producing a comprehensive parameter representation. We also add a skip connection between the input large parameters and the output of the second Conv1x1, facilitating parameter information transfer from the global model to sub-models. With these designs, the sub-models' accuracy increases to 93.15%, effectively mitigating the information loss.

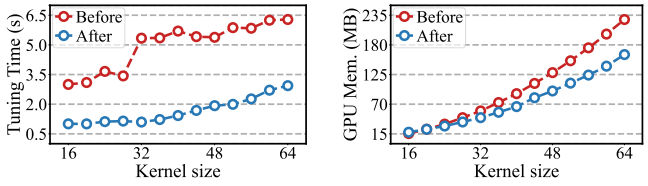
(2) Imbalanced Parameter Distribution. Though the parameter distributions of sub-models and the global model are similar, the sub-model parameters skew towards negative values (Fig. 7(a)), leading to numerical instability, slow convergence, and performance degradation. We adopt a modified Leaky ReLU (MLR) activation function (Fig. 7(b)) to rectify negative parts, where s_n and s_p are slopes for negative and positive values, respectively. Smaller s_n can suppress the negative parameters but not entirely eliminate them, thereby preserving potential information embedded in negative parameters. After applying the MLR, the sub-model parameters exhibit a similar distribution pattern and value range to the global model (Fig. 7(a)), with the sub-model accuracy further increasing to 95.06%.

(3) Performance Fluctuation. During tuning, we observe significant performance fluctuation of the sub-model. This is because in *learning-on-data* methods, model parameters are directly updated during training. However, in our *learning-on-model* method, only *convolution parameters* are updated, which subsequently generate sub-model parameters via the convolution process. As a result, the sub-model exhibits much higher sensitivity to *convolution parameters*. To tackle this, we



(a) Compressed model's accuracy (b) Different parameter distribution

Fig. 8. (a) Accuracy of the compressed model with: 1) two Conv 1×1, 2) the MLR function, 3) weight normalization, 4) learning rate scheduler; (b) The parameter distribution of the dilated models from different client models.



(a) Fine-tuning time (b) Fine-tuning GPU memory

Fig. 9. Server overhead before and after adopting the optimization method.

apply weight normalization on *convolution parameters* to decouple their magnitude and direction during tuning, stabilizing convergence in a fine-grained way [35]. Moreover, we apply a learning rate scheduler that dynamically varies the learning rate to avoid local optima and enables faster convergence [36]. The learning rate undergoes a cosine function decay:

$$lr = lr_{min} + 0.5(lr_{max} - lr_{min})(1 + \cos(e/T_{max} \cdot \pi)) \quad (5)$$

where e is the current epoch index, lr_{min} and lr_{max} are the lower and upper bound of the learning rate, and T_{max} is the maximum number of iterations before the lr restarts to lr_{min}. Specifically, the learning rate begins with a large value (lr_{max}) to explore the loss landscape quickly. As training progresses, it decreases smoothly following the cosine curve to help the model settle into a minimum without overshooting. By the last few epochs, the learning rate decreases slowly, focusing on tuning *convolution parameters* via a smaller value (lr_{min}). With weight normalization and the scheduler, the accuracy of the sub-model improves to 96.8% and 98.59%, respectively. Meanwhile, the performance becomes more stable, and the sub-model converges after around 20 epochs (Fig. 8(a)).

(4) Heavy Serve Overhead. During compression, we find that fine-tuning *convolution parameters* poses a heavy overhead on the server, especially when the global model has a vast number of parameters. Specifically, we vary the kernel size of a single network layer in the global model to control the parameter amount. With a constant SR, we create several sets of *convolution parameters* and record the tuning time and GPU memory usage when updating the *convolution parameters*. From Fig. 9 we see that as the size of the network layer increases, the required tuning time and the GPU memory increase dramatically. The main reason is that we create separate *convolution parameters* for each weight matrix in all the network layers of the global model. As a result, when the global model has larger parameter sizes, the *convolution parameters* also become extremely large, consuming a significant amount of GPU memory and time during fine-tuning.

To tackle this, we propose an optimization mechanism that allows the parameter matrices within one network layer to use the same *convolution parameters*. The lower part in Fig. 5 shows the difference after applying our optimization method. For example, with the weight matrix (32, 16, 3, 3) from the global model, we only create one set of *convolution parameters* for all of the nine extracted (1, 32, 16) matrices, instead of nine separate sets. We also assign different weights for the nine matrices to balance their contribution during tuning. With our optimization method, the tuning process is:

$$\begin{aligned} &\min_{\boldsymbol{w}_{Conv,l}} \textstyle\sum_x \mathcal{L}(f(x; \omega_l^j \cdot \boldsymbol{W}_{G,l}^j \odot \boldsymbol{w}_{Conv,l}), y), \\ &s.t.\ \forall\, l \in \{1, 2, \cdots, L\}, \forall\, (x, y) \in \mathcal{D}, \forall\, j \in \{1, 2, \cdots, J_l\}. \end{aligned} \quad (6)$$

where for all the parameter matrices $\boldsymbol{W}_{G,l}^j$ of the l-th network layer, we use the same *convolution parameters* $\boldsymbol{w}_{Conv,l}$ for compression and tuning. ω_l^j is the weight assigned for the j-th weight matrix of the l-th network layer. By compressing the matrices using one shared set of *convolution parameters* and collaboratively fine-tuning the parameter set, the server overhead (*i.e.*, GPU memory usage and tuning time) can be significantly reduced, as shown in Fig. 9. Further evaluations also demonstrate that our proposed optimization approach only results in a slight performance degradation (§ VI-E2).

Algorithm 1: Convolutional compression

Input : Global round r, pre-training epochs e_p, *convolution parameters* updating epochs e_c, device type number n, SR list $\{SR_1, SR_2, \cdots, SR_n\}$, T_{max}, lr_{max}, lr_{min}
Output: Compressed parameters $\{P_1, P_2, \cdots, P_n\}$

1 /* Configuration initialization */;
2 **if** $r == 1$ **then**
3 Initialize the global model as $\boldsymbol{W}(r)$;
4 Pre-train $\boldsymbol{W}(r)$ for e_p epochs on $\mathcal{D}$;
5 **for** *device_type* $i \in \{1, 2, \cdots, n\}$ ***parallel*** **do**
6 $current\_shrinkage\_ratio \leftarrow SR_i$;
7 $Conv_i(r) \leftarrow$ **Initialize_Conv**$(\boldsymbol{W}(r), SR_i)$;
8 **end**
9 **else**
10 $Conv_i(r) \leftarrow Conv_i(r-1)$;
11 **end**
12 /* Convolution parameters fine-tuning */;
13 **for** *device_type* $i \in \{1, 2, \cdots, n\}$ ***parallel*** **do**
14 **for** $e \in \{1, 2, \cdots, e_c\}$ **do**
15 $lr = lr_{min} + 0.5(lr_{max} - lr_{min})(1 + \cos(e/T_{max} \cdot \pi))$;
16 **for** ***each*** (x, y) ***in*** $\mathcal{D}$ **do**
17 **for** $l \in \{1, 2, \cdots, L\}; j \in \{1, 2, \cdots, J_l\}$ **do**
18 $P_i \leftarrow$ **MLR**$(\omega_l^i \cdot \boldsymbol{W}(r)_l^j \odot Conv_i(r)_l)$;
19 $output \leftarrow f(x; P_i)$;
20 $loss \leftarrow$ **Loss_fn**$(output, y)$;
21 Back-propagate gradient g_i to $Conv_i(r)_l$;
22 $Conv_i(r)_l \leftarrow Conv_i(r)_l - lr \cdot g_i$;
23 **end**
24 **end**
25 **end**
26 **end**
27 Send P_i to the corresponding client;

After *convolutional compression*, the server sends the compressed parameters to the corresponding clients for local training. The tuned *convolution parameters* are kept on the server and updated in the next global round. This process is performed entirely on the server without imposing any extra computation or communication burden on clients. Algo. 1 shows the detailed process of *convolutional compression*. The server first initializes the global model and *convolution parameters* based on SR values reported from heterogeneous clients (Line 1–11). Next, for each SR, the server performs several epochs of *convolution parameter* tuning so that the compressed model achieves comparable performance to the global model (Line 12–26). The *convolutional compression* process fully exploits our *learning-on-model* paradigm with our MLR function, cosine learning rate scheduler, and tuning optimization method.

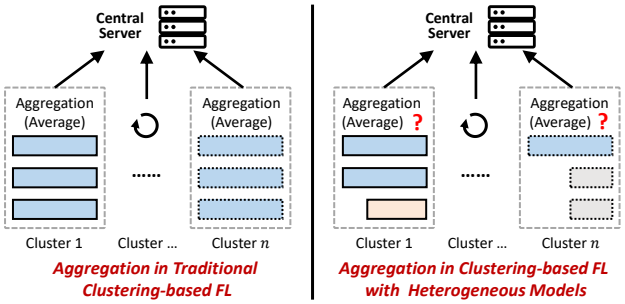


Fig. 10. Traditional clustering-based FL vs. with heterogeneous models.

B. Hierarchical and Clustering-Based Local Training

A practical issue arises during local training: different clients may have diverse data distributions, *i.e.*, data heterogeneity [37]. Existing works typically adopt clustering-based methods, where clients with similar data distributions are clustered together. The server first aggregates client models within the same cluster, *i.e.*, intra-aggregation. With such aggregated models from different clusters, the server further aggregates them via weighted average to mitigate data heterogeneity, *i.e.*, inter-aggregation. However, when clients within the same cluster have heterogeneous model sizes, they cannot be directly aggregated due to model size mismatch (Fig. 10). As a result, during intra-aggregation, clients may miss model information from other client models trained on local data from different domains. For instance, smartphone models may miss information from PC models that capture features of data from a different domain. This leads to domain misalignment among clients and causes unexpected performance degradation [38]. To mitigate such misalignment among clients under both data and model heterogeneity, we propose a *hierarchical and clustering-based local training* module to enhance the client model performance. We first create several edge servers and organize clients into a hierarchical structure: edge servers $\rightarrow$ clusters $\rightarrow$ clients. By grouping clients with similar data distributions into clusters, the personalized performance after local training can be enhanced. Besides, we create a knowledge-sharing strategy by creating several projection layers to enable cross-cluster parameter exchange. Through such a hierarchical design, the personalization performance of heterogeneous clients can be improved with reduced domain gaps.

Clustering. Before FL starts, the edge servers first group their corresponding clients into multiple clusters based on their data distributions. Specifically, each edge server first creates a model and distributes it to its clients. Clients perform one epoch of local training to tune the received model. Then, each client sends the tuned model back to the edge server. The edge server performs K-means Clustering [39] based on the Kullback-Leibler Divergence (KLD) [40], which serves as a distance measuring metric between different model parameters. A higher KLD value indicates greater similarity between the two models with a more similar data distribution [41]. Note that the number of clusters, K, can be dynamically determined during runtime following previous works [42], [41].

Local Training & Cross-cluster Parameter Exchange. After receiving the model parameters from the edge server, clients perform several epochs of local training same as the conventional FL process. Next, clients transmit their models to the edge server, which will perform parameter aggregation for each cluster. By extracting the last linear layer that best represents clients' data distributions [43] into a pool, the edge server will construct a triangular matrix by calculating the KLD value between the parameters of each layer pair in the pool. The KLD matrix $\boldsymbol{K}_{(n,n)}$ is expressed as:

$$\boldsymbol{K}_{(n,n)} = \text{KLD}(\boldsymbol{w}_i, \boldsymbol{w}_j),\ \forall\ i \in \{1, 2, \cdots, n\}, j \in \{i+1, \cdots, n\} \quad (7)$$

where n is the number of clusters from all the edge servers, $\boldsymbol{w}_i$ is the parameter of the i-th linear layer in the pool, and $\text{KLD}(\cdot)$ represents the KLD function. With the triangular matrix, we select a subset of layer pairs whose KLD values are below a pre-defined threshold. Two projection layers are then created between the two selected linear layers. Furthering this, all clients continue to perform several epochs of local training with the projection layers. Specifically, the input x_i to client i's layer will be fed into a projection layer $\mathcal{P}_{i\rightarrow j}(\cdot)$. The output of the project layer is combined with client j's layer to generate the final prediction output. This process can be expressed as:

$$\begin{aligned} o_i &= \mathcal{F}_i(x_i) + \lambda \cdot \mathcal{P}_j(x_j) \\ o_j &= \mathcal{F}_j(x_j) + (1-\lambda) \cdot \mathcal{P}_i(x_i) \end{aligned} \quad (8)$$

where o_i and o_j are the output of the two clients' last linear layers. $\mathcal{F}_i(\cdot)$ and $\mathcal{P}_i(\cdot)$ are the linear layer and the projection layer of client i, respectively. λ is a hyperparameter to balance the knowledge sharing between the two clients. During local training, we only update the linear layer and the projection layer via back-propagation. Such a two-stage training method enables cross-cluster information sharing among clients with similar data distributions but different model sizes, significantly enhancing personalization performance(§ VI-E3). The intermediate outputs and the back-propagated gradients are transmitted via such a "client $\leftrightarrow$ edge server $\leftrightarrow$ client" path.

Privacy. Though transmitting intermediate outputs may pose privacy risks via model inversion attacks [44], [45], SplitFed [46] has theoretically proven that such methods are privacy-preserving. Each client only accesses the smashed data (*i.e.*, intermediate output before linear layers) from another client. Reconstructing the original data requires inverting entire model parameters, which is impractical. This risk can be reduced by using larger fully connected layers [47] or modifying loss functions [48]. Privacy can be further enhanced with techniques like differential privacy (DP), which adds calibrated noise while maintaining utility [49]. Our experiments confirm that *FedConv* still achieves comparable performance on the MNIST dataset [34] after applying DP.

Remarks. We do not pose overhead on resource-constrained clients as the projection layers are kept on edge servers. Besides, the overhead of edge servers is also moderate as they only perform aggregation and parameter transmission. Following FedAvg, we adopt the same client selection strategy (random selection), ensuring that each client has an equal probability of being selected. To further enhance fairness and model performance, we can orthogonally integrate advanced client selection methods [50] or dropping strategies [41].

C. Transposed Convolutional Dilation

Upon receiving the updated sub-models from clients, we rescale the heterogeneous client models to a unified size for further aggregation. Although KD-based methods are promising, they impose significant system overhead on clients (§ I). Instead, we adopt transposed convolution (TC), a reverse operation to the *convolution compression*. We apply different

TABLE I
THE HARDWARE CONFIGURATION OF HETEROGENEOUS DEVICES IN A REAL-WORLD EXPERIMENT.

Type	Device Name	Number	CPU	RAM	GPU	GDDR	Network	SR
Server	ASUS W790-ACE Server	1	Intel Xeon Gold 6248R, 3.0GHz	640GB	NVIDIA A100	40GB	Ethernet	-
Edge Server	Supermicro X11SCA-F	3	Intel Xeon E-2236, 3.4GHz	32GB	NVIDIA RTX 4090	24GB	Ethernet	-
Router	Mi Router AX3000	1	Qualcomm IPQ5000 A53, 1.0GHz	256MB	-	-	Ethernet	-
PC	Supermicro X11SCA-F	2	Intel Xeon E-2236, 3.4GHz	32GB	NVIDIA RTX A4000	16GB	Ethernet	1.0
	Supermicro SYS-5038A-I	2	Intel Xeon E5-2620 v4, 2.10GHz	64GB	NVIDIA GeForce GTX 1080 Ti	12GB * 2	Wi-Fi	1.0
	ThinkPad P52s Laptop	4	Intel i5-8350U , 1.70GHz	32GB	NVIDIA Quadro P500	2GB	Wi-Fi	0.75
Board	NVIDIA Jetson TX2	4	Dual-Core NVIDIA Denver 2, 2GHz	8GB	256-core NVIDIA Pascal GPU	4GB	Wi-Fi	0.75
	NVIDIA Jetson Nano	4	ARM Cortex-A57 MPCore, 1.5 GHz	4GB	NVIDIA Maxwell architecture GPU	2GB	Wi-Fi	0.5
	Raspberry Pi 4	4	Quad core Cortex-A72, 1.8GHz	8GB	-	-	Wi-Fi	0.25

TC layers to each of the received client models, as they are trained on non-IID data with different sensing heterogeneity and thereby inherently carry diverse personalized information. Then, by meticulously fine-tuning the *TC parameters* (*i.e.*, parameters of the *TC layers*), the personalized information embedded in each client model's parameters will be preserved and transferred to the dilated models for subsequent aggregation.

TC Configurations. To transform heterogeneous client models to unified sizes, the configurations of each *TC layer* for dilation are identical to the corresponding *compression layer*. For instance, a convolutional layer in a client model has 12 input and 24 output channels with a (3,3) kernel. With the SR of 0.75, the *TC layer* configuration should be TC$\langle in$=1, out=1, k=$(9,5)$, s=1, p=0$\rangle$, as shown in Fig. 6(b). This process can also be employed to dilate other kinds of layers. Note that the input channel of the first layer and the output channel of the last layer are also unchanged.

TC Parameter Fine-tuning. To fine-tune *TC parameters*, we also set them learnable and minimize the loss between the ground truth and the prediction result of the dilated model. Similarly, we also apply the optimization strategy to the TC dilation process to reduce the server's system overhead during tuning. The fine-tuning process can be expressed as

$$\begin{aligned} &\min_{\boldsymbol{w}_{TC,l}} \textstyle\sum_x \mathcal{L}(F(x; \Omega_l^j \cdot \boldsymbol{W}_{C,l}^j \otimes \boldsymbol{w}_{TC,l}), y), \\ &s.t. \; \forall\, l \in \{1, 2, \cdots, L\}, \forall\, (x, y) \in \mathcal{D}, \forall\, j \in \{1, 2, \cdots, J_l\}. \end{aligned} \tag{9}$$

where $F(\cdot)$ is the dilated model's forward function, $\boldsymbol{W}_{C,l}$ is client model parameters, $\boldsymbol{w}_{TC,l}^j$ is the *TC parameters* assigned for the j-th weight matrix in the l-th layer of the client model, $\otimes$ is the TC operation, and Ω_l^j is the weight assigned for the *TC parameters*. To further enhance the transfer of personalized information into dilated models, we also add two 1x1 TC layers with a residual connection before dilation (Fig. 6(b)).

D. Weighted Average Aggregation

Given a set of dilated models, the server aggregates them to obtain the global model. However, we find that directly averaging dilated models' parameters leads to severe performance degradation (the accuracy of the aggregated model is only 47.6%). The reasons are twofold: 1) the magnitude of the dilated models' parameters varies with their sizes [30]; 2) the parameters of the dilated models through TC operations also carry personalized information from different clients, thus exhibiting distinct patterns and varying skewness toward client-side data distribution (Fig. 8(b)). Simply aggregating these dilated models overlooks the diverse contributions that heterogeneous clients can make in the aggregation process.

Contribution Balancing. To balance the diverse personalized information of dilated models, we first normalize their parameters to $[0, 1]$ and then assign different learnable *weight vectors* to every network layer in each dilated model for *weighted aggregation*. The parameters of the l-th aggregated layer are:

$$\boldsymbol{W}_l = (\sum_{j=1}^{n} \boldsymbol{v}_{j,l} \cdot s_j \cdot \boldsymbol{w}_{j,l}) / \sum_{j=1}^{n} s_j \tag{10}$$

where $\boldsymbol{W}_l$ is the parameters of the l-th layer in the aggregated model, n is the number of large models. $\boldsymbol{w}_{j,l}$ and $\boldsymbol{v}_{j,l}$ are the parameters and the corresponding *weight vector* of the l-th layer in the j-th large model, s_j is the number of data samples that are used for training the j-th client model. We iteratively optimize $\boldsymbol{v}_{j,l}$ to gradually balance the distinct contributions.

Aggregation Enhancement. To further quantify the distinct contributions of heterogeneous clients and enhance the aggregation process, we use KLD to measure the similarity between the parameters of the global model and the dilated models. The higher the similarity, the more contribution the large model will make to the aggregation. Thus, the aggregated global model can attain higher generalizability and a more comprehensive global perspective. The KLD for the j-th dilated model is denoted as $KLD_j = \sum_{l=1}^{L} \sum_x \boldsymbol{W}_{G,l}(x) \log \frac{\boldsymbol{W}_{G,l}(x)}{\boldsymbol{w}_{j,l}(x)}$, where $\boldsymbol{W}_G$ is the global model parameters from the previous communication round. The optimization of the *weight vectors* is:

$$\mathcal{L}(\boldsymbol{v}) = \mathcal{L}_{\mathcal{D}}(\boldsymbol{W}) + \lambda \sum_{j=1}^{n} KLD_j \tag{11}$$

where $\mathcal{L}$ is the Cross-Entropy loss for the model output, and λ is a coefficient for balance. After fine-tuning the *weight vectors*, the aggregated model will be used for the next round.

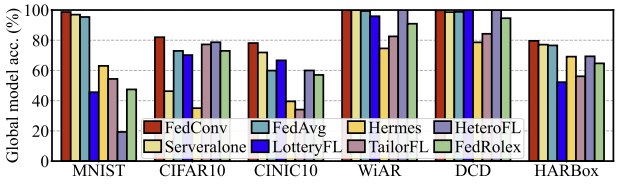
V. EXPERIMENT SETUP

A. Implementation

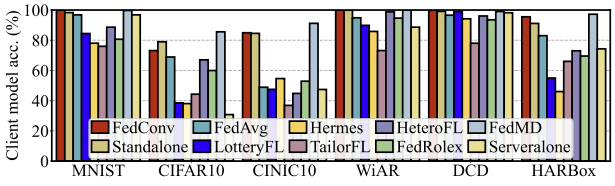
We develop *FedConv* with Flower [20]. We override the `load_state_dict()` function to enable gradients to back-propagate to *convolution/TC parameters*. We evaluate *FedConv* with a cloud server, four edge servers, a router, and 20 heterogeneous mobile devices. Detailed configurations of the devices are described in Table I. We deploy these edge devices in our offices and laboratories under real-world network conditions.

B. Datasets and Models

We select two representative mobile applications and use different model architectures and sizes on various datasets.



(a) Global model accuracy comparison



(b) Client model accuracy comparison

Fig. 11. Accuracy comparison under model heterogeneity ($\alpha = 0.1$).

Image Classification is a popular computer vision application. We choose three datasets: 1) **MNIST** [34] consists of 60K 28×28 gray-scale images of ten handwritten digits. We use a convolutional neural network (CNN) with two Conv layers and one linear layer for evaluation; 2) **CIFAR10** [22] consists of 60K 32×32 color images in ten classes. We use ResNet18 [21] for evaluation; 3) **CINIC10** [51] contains 180K 32×32 color images in ten classes. We use GoogLeNet [52] for evaluation. **Human Activity Recognition (HAR)** [53], [54] analyzes different types of sensor data (*e.g.*, Depth camera, IMU [55], [56], [57], and WiFi CSI [58], [59], [60], [61]). We select three datasets: 1) **WiAR** [62] includes 480 90×250 CSI data of 16 activities. We augment it to 64K samples following [63]; 2) **Depth camera dataset** (DCD) [41] contains 5K 36×36 gray-scale depth images of five gestures; 3) **HARBox** [41] includes 30K 1×900 features extracted from IMU data of five activities, collected from 121 users with 77 smartphones, exhibiting sensing heterogeneity. We use a CNN model with three Conv layers and one linear layer for evaluation.

We divide these datasets into four parts: 1) the IID data for *convolution/TC parameters* and *weight vectors* tuning, 2) IID test data for global model evaluation, 3) training and 4) testing data (IID or non-IID) for client model. Each part counts for 5%, 20%, 70%, and 5% of the total dataset, respectively. The first and second sub-sets are kept on the server, whereas the third and fourth parts are distributed among heterogeneous clients. To emulate real-world heterogeneity, we also employ different datasets on the server and clients (§ VI-G).

C. Baselines

1) **Serveralone** trains one model with only the server-side data. We evaluate the model using the server-side IID test data and non-IID client-side test data. 2) **Standalone** allows each client to train an affordable local model using private data without parameter exchange. 3) **FedAvg** [19] is a classic FL paradigm where clients upload the updated local model to a central server for averaging aggregation. We assign the smallest affordable models to all clients to meet the constrained resources of some devices. 4) **FedMD** [16] utilizes KD to reach a consensus among heterogeneous client models by training on a public dataset. 5) **LotteryFL** [64] exploits the Lottery Ticket hypothesis to generate heterogeneous sub-models. 6) **Hermes** [18] finds a sparse sub-model for each client by performing channel-wise pruning to reduce client overhead. 7) **TailorFL** [30] produces sub-models by filter-level pruning based on the learned importance value of each filter. 8) **HeteroFL** [17] is a parameter-sharing method that allows each client to share a subset of the parameters from the global model. 9) **FedRolex** [24] adopts dynamic rolling windows when extracting sub-models for heterogeneous clients.

D. Heterogeneity Consideration

For model heterogeneity, we consider four SRs: [0.25, 0.5, 0.75, 1.0], according to the resource profiles of the heterogeneous clients, as detailed in Table I. We adopt larger SRs for powerful clients (*e.g.*, 0.75 for laptops) and smaller SRs for resource-constrained clients (*e.g.*, 0.25 for Raspberry Pis). For data heterogeneity, we sample the disjoint non-IID client-side data using the Dirichlet distribution $\mathbf{Dir}(\alpha)$. A smaller α generates a more heterogeneous distribution [65].

E. Hyper-parameter Settings

We set the number of global rounds to 100, the local training epochs to 5, and the learning rate to 0.001. In model compression and dilation, the stride and padding of all the *convolution parameters* are 1 and 0. The server-side pre-training epoch number is 5. The epoch number for updating *convolution/TC parameters* are both 20, and the T_{max}, lr_{min}, lr_{max} in the learning rate scheduler are 4, 0.00001, and 0.001, respectively. s_p and s_n (Fig. 7(b)) of the activation function are 0.85 and 0.001, respectively. In model aggregation, the number of epochs, the learning rate for updating weight vectors, and λ in Eq. (10) are 10, 0.001, and 0.2, respectively.

VI. Evaluation

A. Metrics

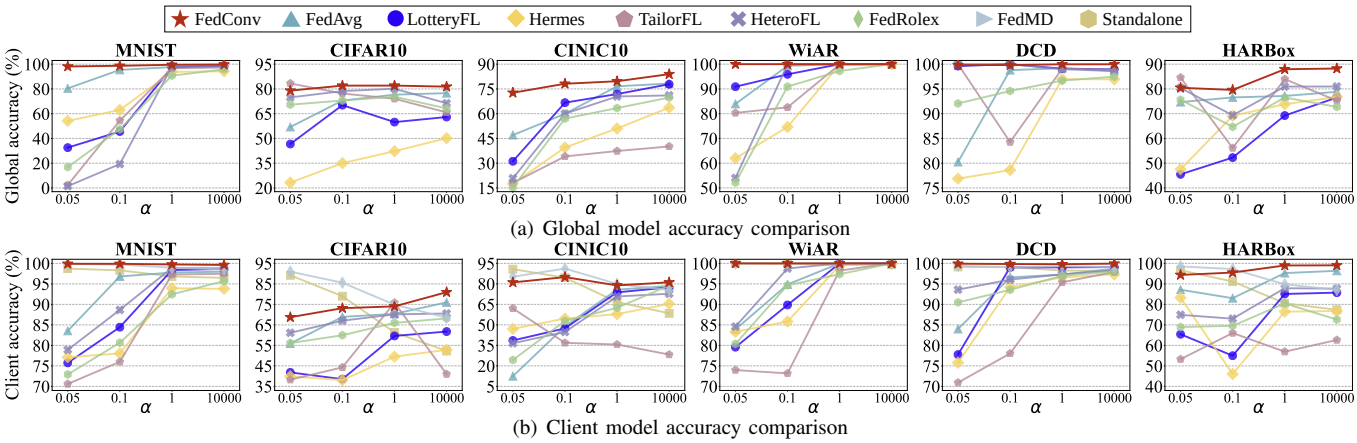
Training Performance: 1) *Model accuracy*: we measure the global model accuracy on the server-side test data to evaluate the generalizability of the global model. We report the average client model accuracy on client-side test data to evaluate personalization performance. 2) *Communication cost*: we monitor the network traffic of all the clients over 100 global rounds.

Runtime Performance: 1) *Memory footprint*: we monitor real-time GPU memory usage via PyTorch CUDA Toolkit. We track each client's process ID over 100 communication rounds to monitor their CPU usage and report the average value. 2) *Wall-clock time*: we measure the execution time of each client from receiving model parameters to finishing local training, and report the average wall-clock time in each round.

B. Overall Performance

1) Global Model Performance: We first evaluate the global model's accuracy to investigate its generalizability. Standalone and FedMD are excluded because they do not create global models. Fig. 11(a) shows the global model accuracy under the same data heterogeneity level (α = 0.1). Serveralone achieves higher accuracy than baselines in most cases, as the server-side training and testing data are both IID. *FedConv* achieves average improvements of 20.5%, 13.8%, and 10.5% compared with pruning-based methods (Hermes and TailorFL), parameter-sharing-based method (HeteroFL and FedRolex) and the rests, respectively. Since we assign the smallest affordable model to all clients in FedAvg, the client models have an insufficient number of parameters for training. Therefore, *FedConv* can outperform FedAvg even with IID data. This shows the superior generalization performance of *FedConv*.

Moreover, Fig. 12(a) shows the global model accuracy of *FedConv* and baselines across different non-IID data on all



(a) Global model accuracy comparison

(b) Client model accuracy comparison

Fig. 12. The inference accuracy of aggregated global models and client models on different datasets.

TABLE II
SYSTEM RESOURCE OVERHEAD.

Metric	System	Heterogeneous Data ($\alpha = 0.05$)						Homogeneous Data ($\alpha = 10000$)					
		MNIST	CIFAR10	CINIC10	WiAR	DCD	HARBox	MNIST	CIFAR10	CINIC10	WiAR	DCD	HARBox
Memory Footprint CPU + GPU (GB)	Standalone	2.14	3.51	4.07	3.95	2.24	2.19	2.13	3.47	4.47	4.03	2.21	2.17
	FedAvg	1.90	2.40	3.31	2.39	1.98	2.01	1.90	2.51	2.79	2.36	1.88	2.08
	FedMD	2.71	3.65	7.51	4.71	2.99	2.79	2.71	3.65	7.93	4.58	2.99	2.81
	LotteryFL	2.62	3.51	4.30	3.23	2.69	2.67	2.63	3.49	4.36	3.27	2.70	2.66
	Hermes	2.64	3.45	6.07	3.28	2.73	2.69	2.64	3.35	6.13	3.32	2.72	2.68
	TailorFL	2.75	3.61	5.09	3.41	2.79	2.71	2.75	3.47	7.52	3.16	2.77	2.70
	HeteroFL	2.63	3.31	4.15	3.25	2.73	2.67	2.63	3.45	4.10	3.08	2.73	2.67
	FedRolex	2.63	3.21	4.15	3.25	2.72	2.67	2.60	3.54	4.16	3.16	2.68	2.69
	FedConv	**2.52**	**3.21**	**4.15**	**3.02**	**2.60**	**2.67**	**2.52**	**3.35**	**4.10**	**3.14**	**2.62**	**2.67**
Wall-clock Time (s)	Standalone	3.87	24.65	279.62	8.05	5.91	3.54	9.38	52.38	273.52	7.60	6.14	3.56
	FedAvg	7.05	39.19	285.30	10.62	10.19	10.09	13.75	97.95	1711.34	20.79	43.67	26.98
	FedMD	44.34	437.14	5370.83	55.03	75.25	32.92	45.17	475.42	6700.17	64.43	79.10	34.53
	LotteryFL	9.18	147.98	699.35	8.89	8.61	5.69	17.59	235.89	1829.33	19.77	22.06	10.92
	Hermes	43.22	714.00	5580.71	103.90	169.97	104.53	43.84	937.82	7621.38	117.85	217.97	115.31
	TailorFL	6.98	62.89	393.46	14.44	12.72	10.11	13.61	99.60	813.94	25.53	13.96	13.27
	HeteroFL	6.96	42.56	641.21	10.78	10.03	5.10	13.56	82.07	1310.81	22.26	23.90	10.98
	FedRolex	6.92	45.98	602.48	11.57	12.34	4.87	12.46	84.25	1389.41	23.64	20.14	11.26
	FedConv	**5.96**	**40.68**	**264.30**	**12.96**	**10.15**	**4.40**	**10.33**	**71.26**	**1406.87**	**21.79**	**17.22**	**9.89**

datasets. We can see that the performance enhancement of *FedConv* becomes more significant as α decreases, meaning that *FedConv* can better cope with the increased data heterogeneity. Although *FedConv* does not obviously outperform FedAvg with homogeneous data, it exhibits better generalizability and robustness in the global model under heterogeneous data. *FedConv* also provides better personalization performance for clients (§ VI-B2). The performance improvements of the global model stem from our *convolutional compression* and *TC dilation* methods. They facilitate the information embedded in the global model being preserved and transferred from the server to clients through our *learning-on-model* approach.

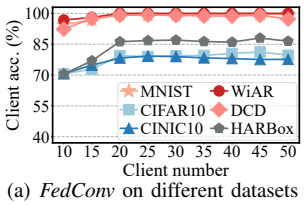
2) Client Model Performance: To evaluate personalization performance, we measure the average accuracy of the client models. Fig. 11(b) shows that with the same non-IID data settings, *FedConv* outperforms baselines (FedAvg, LotteryFL, Hermes, TailorFL, HeteroFL, and FedRolex) with accuracy improvements ranging from 8.4% to 50.6%. In Serveralone, when evaluating the global model on the client-side non-IID data, the accuracy drops below that of most baseline systems. This is because, in *FedConv*, the server-side data occupies a small portion (5%) of the entire dataset. Therefore, Serveralone's global model hasn't seen sufficient data, leading to degraded performance on the client-side non-IID data.

Additionally, Fig. 12(b) shows the client model accuracy under different data heterogeneity levels. We see that the performance disparities become more substantial as α decreases, implying that *FedConv* is more robust and can achieve consistently high accuracy across diverse data distribution. This performance gain stems from the *TC dilation* process, where distinct *TC parameters* are assigned to each uploaded client model on the server. The dilated models will thereby preserve clients' personalization information, which is then aggregated into the global model. Besides, in Fig. 11(b), with sensing heterogeneity in the HARBox dataset, *FedConv* achieves a better and more stable performance. However, when α is small (*e.g.*, 0.05 and 0.1 on CIFAR10), the client model accuracy of FedMD is higher than *FedConv*. The better performance stems from the distilled knowledge shared by all clients. Nonetheless, the downside is that it imposes excessive communication and computational overhead on clients (Table II & Table III). By contrast, *FedConv* can achieve comparable personalization performance without an extra burden on clients. In practice, we can further improve the personalization performance by adding task-specific layers [66] (detailed in§ VI-F).

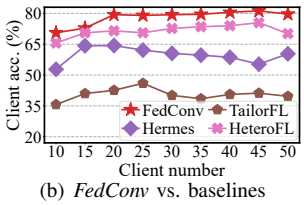
TABLE III
COMMUNICATION OVERHEAD COMPARISON (GB).

System	MNIST	CIFAR10	CINIC10	WiAR	DCD	HARBox
FedAvg	14.80	4815.84	2697.85	28.24	13.45	8.87
FedMD	19.99	5126.46	2859.79	40.91	19.94	16.24
LotteryFL	11.11	4713.91	2623.93	23.01	10.05	8.55
Hermes	16.34	7099.66	2848.83	36.63	15.02	12.95
TailorFL	11.40	4787.18	2686.15	24.30	10.32	8.82
HeteroFL	11.11	4713.91	2623.93	23.01	10.05	8.55
FedRolex	11.11	4713.91	2623.93	23.01	10.05	8.55
FedConv	**11.11**	**4713.91**	**2623.93**	**23.01**	**10.05**	**8.55**

Remarks. *FedConv* exhibits significant performance gains in both global and client models across various settings. The parameter information of the global model can be preserved via our *convolutional compression* module. We suspect that the performance instability of some baselines might be attributed to the information loss in pruning and parameter sharing.

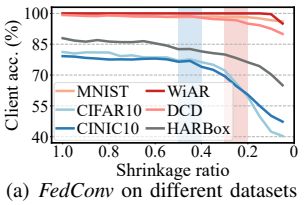


(a) *FedConv* on different datasets

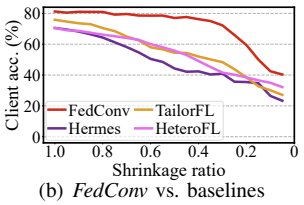


(b) *FedConv* vs. baselines

Fig. 13. Varying number of clients.



(a) *FedConv* on different datasets



(b) *FedConv* vs. baselines

Fig. 14. Varying shrinkage ratios.

C. Overhead Assessment

We evaluate the memory footprint, wall-clock time, and communication overhead of each client in *FedConv* and baselines with both homogeneous (α = 10000) and heterogeneous (α = 0.05) data across clients. Table II provides an overview of the average memory usage and the average wall-clock time of each client. With the same set of SRs, *FedConv* achieves an average saving of 40.6% in memory cost and 54.6% in computation overhead compared with the baselines, respectively. Furthermore, when the client model is complex (ResNet18 and GoogLeNet), *FedConv* only needs approximately half of the memory and training time compared to the pruning-based methods. For example, in the homogeneous data condition, *FedConv* needs 2GB less memory and saves around 90 minutes of wall-clock time than Hermes in one single round. This is because the computation-intensive pruning operations are executed on the resource-constrained clients. In contrast, clients in *FedConv* only need to perform local training in each round, resulting in significant savings in terms of memory, computation, and communication resources. Note that FedAvg consumes less memory and wall-clock time because we assign the smallest affordable models to all clients.

Table III lists the total size of data packets transmitted through the network by all clients. We observe that the communication cost of *FedConv*, LotteryFL, and HeteroFL are comparable, as they exclusively transmit sub-model parameters without extra contents. In contrast, Hermes and TailorFL have to transmit the pruning structure, and FedMD needs to transmit logits. Thus, *FedConv*, LotteryFL, and HeteroFL are more friendly to resource-constrained clients. Moreover, it holds significant potential that exiting quantization techniques [67], [68] and masking method [69] can be extended to *FedConv*, to further diminish the communication overhead.

Remarks. In summary, benefiting from the lighter communication and computation burden imposed on resource-constrained clients, *FedConv* saves more system resources and performs inference tasks faster than the baselines.

D. Sensitivity Analysis

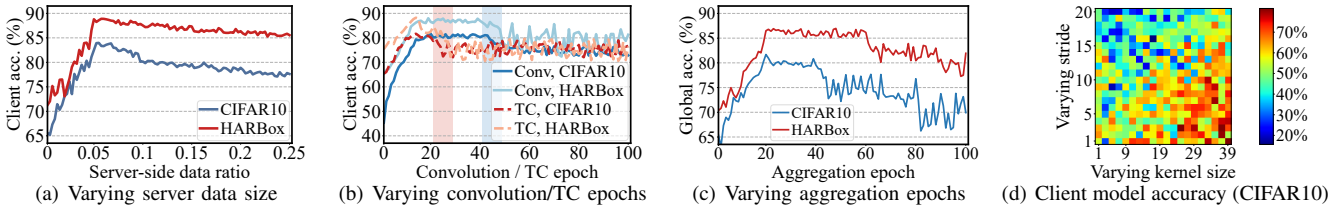
1) Varying Client Number: We simulate 100 clients and vary the number of selected participating clients from 10 to 50 (α = 10000) to compare the client model performance with the baselines. As shown in Fig. 13(a), the client model accuracy in *FedConv* exhibits an upward trend as the number of clients increases. For example, the client model accuracy on HARBox increases by 17.54% when the number of clients increases from 10 to 50. We then select CIFAR10 to compare the client model performance of *FedConv* with pruning-based and parameter-sharing-based methods. From Fig. 13(b), we see that *FedConv* attains an average client model accuracy that is at least 32.5% higher than that of the baselines. The results demonstrate the scalability and superiority of *FedConv* with varying client numbers.

2) Varying Shrinkage Ratios: To investigate the trade-off between the SR and model performance, we set the SR for 10 clients as 1.0 and set the SR for the remaining 10 clients as r. We then vary r from 1.0 to 0.05 and record the average client model accuracy (α = 10000). From Fig. 14(a), we can see that as the SR decreases below a certain threshold, there is a notable accuracy drop in client models, as expected. For MNIST, WiAR, and DCD, the SR threshold is about 0.25 (the red shadow), and for CIFAR10, CINIC10, and HARBox, the threshold is about 0.4 (the blue shadow). Fortunately, we find that even a lightweight device (*e.g.*, Raspberry Pis) can afford the GoogLeNet model on CINIC10 when the SR is 0.4. Consequently, as long as the SR remains above the corresponding threshold, it can be reduced to conserve system resources effectively.

We then use CIFAR10 to compare *FedConv* with the baselines, and the client model accuracy is shown in Fig. 14(b). We can see that though the accuracy of *FedConv* also decreases with limited resources, it can retain much higher accuracy than the baselines. The reason is that with a lower SR (higher pruning rate), the baselines discard a larger amount of parameter information. In contrast, with *convolutional compression*, *FedConv* can effectively preserve the parameter information of the global model as much as possible to the sub-models bounded by their sizes and resource budgets.

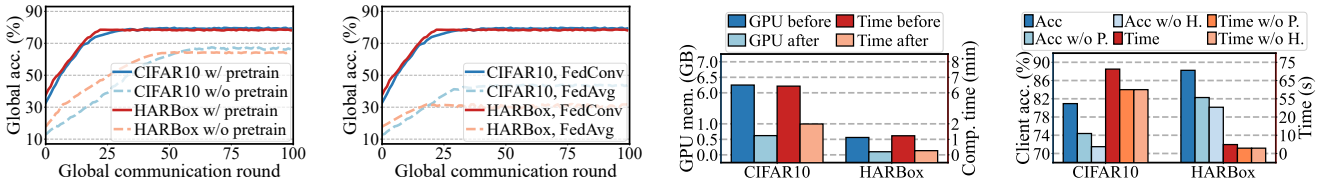
3) Varying Server-side Data Sizes: To investigate the impact of server-side data, we vary the sample number ratio of the server-side data from 1% to 25%, with a step of 0.005. As shown in Fig. 15(a), we obtain two key observations: 1) When the ratio of the server-side data varies from 1% to 5%, the client models will have better performance, due to the richer information obtained from the server-side data; 2) After the turning point (5%) the global model tends to overfit to the server-side data, leading to less personalization and degraded client model performance. In our evaluation, we set the default sample ratio of the server-side data to 5%. Note that the actual turning point may differ in practice. In addition, continuous learning [70] or incremental learning [71] techniques can be further applied as more server-side data become available.

4) Varying Hyper-parameters: We vary the number of tuning epochs in the model compression, dilation, and aggregation to evaluate the impact on the personalization performance of client models. We select CIFAR10 and HARBox for demonstration. We first vary the number of epochs for updating the *convolution/TC parameters* in each global round. Fig. 15(b) shows that when the number of *convolution/TC parameters* updating epochs is around 20, the client models achieve better and more stable performance. After the 20-th and the 40-th epoch, the client model accuracy gradually drops due to the *convolution/TC parameters* being over-fitted to the server-side data. Similarly, from Fig. 15(c), we can observe that when the number of tuning epochs for updating *weight vectors* exceeds



(a) Varying server data size (b) Varying convolution/TC epochs (c) Varying aggregation epochs (d) Client model accuracy (CIFAR10)

Fig. 15. Sensitivity analysis of varying hyper-parameters in *FedConv*.



(a) Server-side pre-training (b) Weight vectors (c) Tuning optimization (d) Local training

Fig. 16. Ablation study of *FedConv* (P. for the cross-cluster parameter exchange and H. for the *hierarchical and clustering-based local training*).

40 and 80, the accuracy of the aggregated global model also decreases. Therefore, we set the number of epochs for model compression, dilation, and aggregation to 20.

We also vary the kernel size and stride length of the *compression layers* and report the mean client model accuracy to explore the impact on client performance. We select a convolutional layer from the large model as an example, whose parameter matrix has a shape of $9 \times (1, 64, 64)$. With the SR being 0.75, the compressed parameter matrix will have a shape of $9 \times (1, 48, 48)$. Since the kernel size k and the stride s should satisfy $(64 + 2p - k + 1)/s = 48$, the padding p can then be determined accordingly. In general, a larger kernel captures more comprehensive parameter information, and a smaller stride captures more fine-grained information. As shown in Fig. 15(d), client models tend to have better performance as the kernel size increases and the stride decreases. However, a larger kernel incurs high computational complexity and imposes a heavy workload on the server. Therefore, we set the kernel size and stride as 23 and 1 by default, respectively.

E. Ablation Study

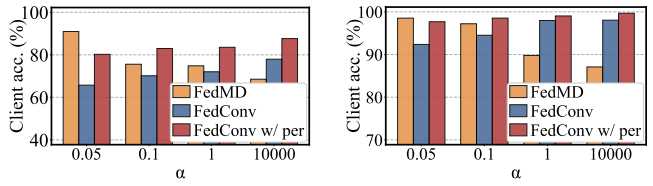
Next, we conduct ablation studies to investigate the importance of different technical modules in *FedConv*.

1) Server-side Pre-training: Fig. 16(a) shows the impact on global model accuracy with and without server-side pre-training with $\alpha = 0.05$. It can be observed that with the integration of pre-training, the global model achieves higher average accuracy (about 15.69%) and reaches faster convergence (about 40 communication rounds earlier), which helps the FL server and clients save communication, computation, and energy costs involved in the training process.

2) Tuning Optimization Mechanism: To investigate the influence of our tuning optimization mechanism on the server-side overhead, we monitor its average memory footprint and wall-clock time before and after adopting the optimization strategy, respectively. Specifically, we select two datasets (CIFAR10 and HARBox) and record the maximum GPU memory usage during the *convolutional compression* process. Additionally, we also report the time cost from when the server begins to compress the global model to when it completes model compression for all SRs. As shown in Fig. 16(c), after adopting our optimization mechanism for the server, both the GPU requirement and the time cost for *convolutional compression* decrease dramatically. Take HARBox as an example, the tuning optimization method saves around 80% GPU memory and 83% tuning time, respectively. The ratios will be further enlarged when the model has a more complex architecture, *i.e.*, nearly 90% ResNet18 for CIFAR10. Besides, we observe that the global model and the client model only experience slight performance degradation (only 2.3%). Such a huge system resource reduction at the cost of slight performance degradation stems from our tuning optimization mechanism, where we assign only one set of *convolution parameters* for all the parameter matrices from the network layer.

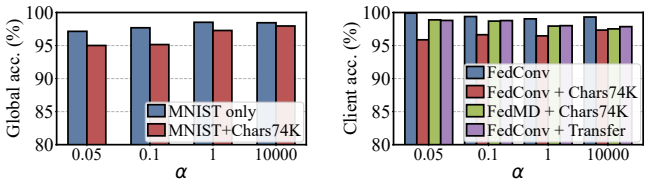
3) Hierarchical and Clustering-based Local Training: To explore the effect of our proposed *hierarchical and clustering-based local training* strategy on client model performance, we conduct two ablation studies. First, we remove the cross-cluster parameter exchange module, making the local training and aggregation similar to hierarchical FL [72], *i.e.*, clients perform conventional local training while each edge server aggregates the model parameters from its corresponding clients and then sends the aggregated model to the central server for further aggregation. Next, we remove the hierarchical architecture of the local clients, *i.e.*, each client separately performs traditional local training and directly transmits the tuned model to the central server for aggregation. We then record the average client model accuracy and wall-clock local training time on two datasets with the original *FedConv*, *FedConv* without parameter exchange, and *FedConv* without the entire *hierarchical and clustering-based local training* module, respectively. As shown in Fig. 16(d), the performance degradation of client models is more obvious when removing the parameter exchange strategy than when removing the hierarchical local training structure. The main reason is that the cross-cluster parameter exchange methodology allows knowledge sharing among clients from different clusters with disparate model sizes that have similar data distributions, which can further enhance the personalization performance. Since training the projection layers during parameter exchange requires extra tuning epochs, the wall-clock time will be reduced to some extent if we remove the *hierarchical and clustering-based local training* module. The results demonstrate the effectiveness of our proposed parameter exchange strategy to enable knowledge sharing between different clients for enhanced personalization performance.

4) Weighted Average Aggregation: To demonstrate the impact of the *weight vectors* for model aggregation, we assign weights with respect to sample number as in FedAvg to all clients and measure the global model accuracy. Fig. 16(b)



(a) CIFAR10 (b) HARBox

Fig. 17. *FedConv* with personalized FL.



(a) Global model accuracy (b) Client model accuracy

Fig. 18. Case study with real-world heterogeneity.

shows the effect on global model accuracy when performing model aggregation with learned weights and equal weights separately. Significant performance degradation can be observed when employing the averaging aggregation method. This is because the parameters from heterogeneous client models usually exhibit varying skewness toward their local data distribution. Merely averaging all the model parameters overlooks the different contributions made by clients in the aggregation process. On the contrary, with the learned *weight vectors*, clients can contribute different parameter information to the aggregated model and improve its generalization ability.

F. Personalization Enhancement

We extend *FedConv* by adding task-specific layers [66] on each client for enhanced personalization, and evaluate the client model accuracy. Specifically, each client appends it personal layers to the sub-model received from the server. By doing so, the personalization performance can be enhanced during local training. We record the average accuracy of client models after 100 global rounds. Fig. 17 shows the performance improvement on five datasets. Compared with FedMD, which achieves the highest client model accuracy (§ VI-B2), we can see that *FedConv* with personalization enhancement is able to surpass FedMD in most cases. This result indicates that *FedConv* can be enhanced with existing personalized federated learning methods to achieve better performance.

G. Case Study with Real-World Heterogeneity

In our default configuration, both the server-side and client-side data are from the same domain. To assess the impact of real-world heterogeneity, we conduct a case study where the Chars74K dataset [73] resides on the server, while the MNIST dataset is kept on heterogeneous clients. The Chars74K dataset contains images of digits from computer fonts with variations (italic, bold, and normal). In this case, the global model learns and extracts general features (*e.g.*, different digit shapes), while heterogeneous clients further fine-tune the sub-model to extract personalized features (*e.g.*, various writing styles of the digits from the MNIST dataset). The *convolutional compression* process and the *TC dilation* process can be regarded as a transformation from one data domain to another. The generated sub-models via *convolutional compression* contain parameter information from the large global model and can thereby extract general features. Similarly, the server applies TC to the locally trained heterogeneous client models to rescale them. This facilitates the aggregation process to form a new global model, retaining the personalization information of the client-side data. As shown in Fig. 18(a) and Fig. 18(b), due to the domain gap between the server-side and the client-side data, there is a decrease in both the global model and the client model accuracy. FedMD still achieves comparable performance to *FedConv* with only the MNIST dataset, benefiting from the knowledge distillation method. However, after applying transfer learning methods [74], [75] tailored to FL, the domain gap between the server-side and the client-side data is significantly mitigated, with *FedConv* achieving much higher accuracy than the baselines. This observation indicates that *FedConv* can be integrated with existing federated transfer learning approaches orthogonally to achieve better performance.

VII. Discussion

Privacy Concerns. In addition to model parameter transmission between the server and clients, *FedConv* requires all clients to report their SRs before the FL training starts. To determine appropriate SRs, clients perform resource profiling locally and report to the server. Additionally, though in the *hierarchical and clustering-based local training* module, there are some extra transmissions between edge servers and clients, only model parameters or intermediate outputs are sent through the network. Note that same as conventional FL schemes, no client-side sensor data is transferred to the server during this process. Thus, we believe the privacy protection of conventional FL schemes can be effectively retained.

Practicality of *FedConv*. In *FedConv*, we use the Flower [20] framework to orchestrate the entire FL process. While Flower offers a stable and robust simulated environment for FL, deploying it in a mixed Linux-Android environment encounters significant obstacles. These include technical challenges in training neural network models on Android devices and issues related to the compatibility of the Flower framework with Android systems. Fortunately, recent advancements [76] in Flower support federated learning setup with Android clients using TensorFlow Lite [77]. Additionally, considering real-world scenarios where clients are typically geographically isolated, we can create multiple edge servers for each SR. For instance, if there are 10 clients with the same SR value, with 5 located in one position and the rest located in another, two edge servers can be deployed to manage them separately.

Server-side Data. Same as KD-based works that share server-side data among clients throughout the FL process, *FedConv* also keeps a small publicly available dataset on the server to refine the *convolution parameters*. Unlike KD-based methods that require clients to first train on the public data and then on the local data for knowledge transfer, *FedConv* keeps the public data only on the server during compression and dilation, eliminating any extra burden on clients. From clients' perspective, they do not need to participate in the pre-training and fine-tuning process and can join the FL process at any time, which is the same as the conventional FL systems. When server data is unavailable due to strict privacy constraints, we can skip the *convolutional compression* process in the first round and later iteratively minimize the logits generated by the global model and heterogeneous sub-models.

Generalize to Other Architectures. In *FedConv*, our *convolutional compression* module focuses on compressing convolu-

tional layers and linear layers, since these two types of network layers dominate most existing popular AI models [78], [79]. For other network architectures (*e.g.*, Transformers [80]), merely compressing the model weights is insufficient. This is because Transformer models include internal hyperparameters (*e.g.*, the number of attention heads) that control the model size and should also be considered for compression. In future work, we plan to comprehensively generalize our *learning-on-model* paradigm to other model architectures.

VIII. Related Work

Data Heterogeneity. Recent works optimize FL performance under non-IID data. *Clustering-based* methods group clients according to the distribution of their data or model parameters. For example, ClusterFL [41] captures the intrinsic clustering patterns among clients by measuring the similarity of client models. Shu *et al*. [81] propose a clustered multi-task federated learning on non-IID data. *Personalized FL* adopts local fine-tuning or add task-specific layers on client side. For example, pFedMe [82] uses Moreau envelopes as a regularized loss function to decouple the task of optimizing a personalized model from the global model learning. Yosinski *et al*. [82] enable the upper layers of the global model to learn task-specific features, while the lower layers capture more general features which are further shared across clients. Our work is orthogonal to these works and requires minimal modification to clients for integration into existing FL systems.

Model Heterogeneity and Compression. To accommodate heterogeneous clients [83], [84], recent works mainly compress the global model to reduce communication and computation costs: 1) *Knowledge distillation* (KD) FL [29] shares a publicly available dataset among the server and heterogeneous clients to perform knowledge transfer. In FCCL [85], clients first compute a distillation loss by feeding their local data into the global model from the previous round, and then calculate an additional distillation loss by inputting the same data into a pre-trained local model. However, extra overhead on clients is unavoidable in KD-based FL, as the public dataset plays a crucial role in knowledge transfer between the server and heterogeneous clients; 2) *Parameter sharing strategies* [86] allow sub-models to share a part of the global model parameters to reduce computation overhead. HeteroFL [17] enables heterogeneous clients to select fixed subsets of global parameters with minimal modification to the existing FL framework. Yet, the sharing strategy suffers from the imbalance issue; 3) *Pruning-based methods* have gained popularity in heterogeneous FL. Hermes [18] applies a channel-level pruning method to selectively prune out less important channels. TailorFL [30] proposes an importance value-based filter-level pruning scheme to enable a dual-personalized FL system. Removing entire channels or filters results in information loss and performance degradation [87]. Unlike these works, we compress the global model with *convolutional compression* to generate sub-models. Orthogonal to our work, traditional compression techniques (*e.g.*, quantization [88]) can be applied to compress model parameters and reduce network traffic. However, as the compressed parameters should be decompressed back to their original size before training, these works cannot reduce the system overhead of clients.

IX. Conclusion

We propose *FedConv*, a client-friendly federated learning framework for heterogeneous clients, aiming to minimize the system overhead on resource-constrained mobile devices. *FedConv* contributes four key technical modules: 1) a novel model compression scheme that generates heterogeneous sub-models with *convolutional compression* on the global model; 2) a *hierarchical and clustering-based local training* strategy that enhances client performance by enabling cross-cluster knowledge sharing; 3) a *transposed convolutional dilation* module that converts heterogeneous client models back to large models with a unified size; and 4) a *weighted average aggregation* scheme that fully leverages personalization information of client models to update the global model. Extensive experiments demonstrate that *FedConv* outperforms SOTA baselines in terms of model accuracy with much lower computation and communication overhead for FL clients. We believe the proposed *learning-on-model* paradigm is worthy of further exploration and can potentially benefit other FL tasks where heterogeneous sub-models can be generated to retain the information of a global model.

References

[1] T. Guo, S. Guo, J. Wang, X. Tang, and W. Xu, "Promptfl: Let federated participants cooperatively learn prompts instead of models-federated learning in age of foundation model," *IEEE TMC*, 2023.

[2] W. Sun, Y. Zhao, W. Ma, B. Guo, L. Xu, and T. Q. Duong, "Accelerating convergence of federated learning in mec with dynamic community," *IEEE TMC*, 2023.

[3] R. A. Sater and A. B. Hamza, "A federated learning approach to anomaly detection in smart buildings," *ACM TIOT*, vol. 2, no. 4, pp. 1–23, 2021.

[4] A. K. Bhuyan, H. Dutta, and S. Biswas, "Towards federated multi-armed bandit learning for content dissemination using swarm of uavs," *ACM TIOT*, 2025.

[5] H. Wang, D. Eklund, A. Oprea, and S. Raza, "Fl4iot: Iot device fingerprinting and identification using federated learning," *ACM TIOT*, vol. 4, no. 3, pp. 1–24, 2023.

[6] L. Shen and Y. Zheng, "Poster: Towards federated embodied ai with feai," in *ACM MobiSys*, 2025, pp. 1–2.

[7] K. Lee, Y. Shin, J. Yun, S. Kim, J. Han, and J. Ko, "Detrigger: A gradient-centric approach to backdoor attack mitigation in federated learning," *arXiv preprint arXiv:2411.12220*, 2024.

[8] K. Wang, Z. Zhou, and Z. Li, "Latte: Layer algorithm-aware training time estimation for heterogeneous federated learning," in *ACM MobiCom*, 2024, pp. 1470–1484.

[9] L. Shen, Q. Yang, K. Cui, Y. Zheng, X.-Y. Wei, J. Liu, and J. Han, "Fedconv: A learning-on-model paradigm for heterogeneous federated clients," in *ACM MobiSys*, 2024, pp. 398–411.

[10] J. Liu, J. H. Wang, C. Rong, Y. Xu, T. Yu, and J. Wang, "Fedpa: An adaptively partial model aggregation strategy in federated learning," *Computer Networks*, vol. 199, p. 108468, 2021.

[11] J. Liu, J. Yan, J. Qi, H. Xu, S. Wang, C. Qiao, and L. Huang, "Adaptive local update and neural composition for accelerating federated learning in heterogeneous edge networks," *IEEE TON*, 2025.

[12] J. Liu, J. Liu, H. Xu, Y. Liao, Z. Yao, M. Chen, and C. Qian, "Enhancing semi-supervised federated learning with progressive training in heterogeneous edge computing," *IEEE TMC*, 2024.

[13] Y. Shin, K. Lee, S. Lee, Y. R. Choi, H.-S. Kim, and J. Ko, "Effective heterogeneous federated learning via efficient hypernetwork-based weight generation," in *ACM SenSys*, 2024, pp. 112–125.

[14] C. Lin, K. Wang, Z. Li, and Y. Pu, "A workload-aware dvfs robust to concurrent tasks for mobile devices," in *ACM MobiCom*, 2023, pp. 1–16.

[15] L. Shen and Y. Zheng, "Iotcoder: A copilot for iot application development," in *ACM MobiCom*, 2024, pp. 1647–1649.

[16] D. Li and J. Wang, "Fedmd: Heterogenous federated learning via model distillation," *CoRR*, 2019.

[17] E. Diao, J. Ding, and V. Tarokh, "Heterofl: Computation and communication efficient federated learning for heterogeneous clients," in *ICLR*. OpenReview.net, 2021.

[18] A. Li, J. Sun, P. Li, Y. Pu, H. Li, and Y. Chen, "Hermes: an efficient federated learning framework for heterogeneous mobile clients," in *ACM MobiCom*, 2021, pp. 420–437.

[19] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, AISTATS*, 2017.

[20] D. J. Beutel, T. Topal, A. Mathur, X. Qiu, T. Parcollet, and N. D. Lane, "Flower: A friendly federated learning research framework," *CoRR*, 2020.

[21] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE CVPR*, 2016, pp. 770–778.

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

[23] D. Yao, W. Pan, Y. Wan, H. Jin, and L. Sun, "Fedhm: Efficient federated learning for heterogeneous models via low-rank factorization," *CoRR*, vol. abs/2111.14655, 2021.

[24] S. Alam, L. Liu, M. Yan, and M. Zhang, "Fedrolex: Model-heterogeneous federated learning with rolling sub-model extraction," *Advances in neural information processing systems*, 2022.

[25] E. Thibeau-Sutre, S. Collin, N. Burgos, and O. Colliot, "Interpretability of machine learning methods applied to neuroimaging," *Machine Learning for Brain Disorders*, pp. 655–704, 2023.

[26] T. M. Cover and J. A. Thomas, *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)*, 2006.

[27] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *IEEE ECCV*, 2014.

[28] M. Sundararajan, A. Taly, and Q. Yan, "Axiomatic attribution for deep networks," in *ICML*, vol. 70, 2017, pp. 3319–3328.

[29] X. Fang and M. Ye, "Robust federated learning with noisy and heterogeneous clients," in *IEEE/CVF CVPR*, 2022, pp. 10 072–10 081.

[30] Y. Deng, W. Chen, J. Ren, F. Lyu, Y. Liu, Y. Liu, and Y. Zhang, "Tailorfl: Dual-personalized federated learning under system and data heterogeneity," in *ACM SenSys*, 2022.

[31] A. Afonin and S. P. Karimireddy, "Towards model agnostic federated learning using knowledge distillation," *ICLR*, 2021.

[32] L. L. Zhang, S. Han, J. Wei, N. Zheng, T. Cao, Y. Yang, and Y. Liu, "nn-meter: towards accurate latency prediction of deep-learning model inference on diverse edge devices," in *ACM MobiSys*, 2021, pp. 81–93.

[33] A. Mao, M. Mohri, and Y. Zhong, "Cross-entropy loss functions: Theoretical analysis and applications," in *International conference on Machine learning*. PMLR, 2023, pp. 23 803–23 828.

[34] L. Deng, "The MNIST database of handwritten digit images for machine learning research," *IEEE Signal Process. Mag.*, 2012.

[35] T. Salimans and D. P. Kingma, "Weight normalization: A simple reparameterization to accelerate training of deep neural networks," *Advances in neural information processing systems, NeurIPS*, vol. 29, 2016.

[36] G. Huang, Y. Li, G. Pleiss, Z. Liu, J. E. Hopcroft, and K. Q. Weinberger, "Snapshot ensembles: Train 1, get M for free," in *ICLR*, 2017.

[37] C. Yang, M. Xu, Q. Wang, Z. Chen, K. Huang, Y. Ma, K. Bian, G. Huang, Y. Liu, X. Jin *et al.*, "Flash: Heterogeneity-aware federated learning at scale," *IEEE TMC*, 2022.

[38] G. Zhu, X. Liu, S. Tang, and J. Niu, "Aligning before aggregating: Enabling communication efficient cross-domain federated learning via consistent feature extraction," *IEEE TMC*, 2023.

[39] J. A. Hartigan, M. A. Wong *et al.*, "A k-means clustering algorithm," *Applied statistics*, vol. 28, no. 1, pp. 100–108, 1979.

[40] J. Lin, "Divergence measures based on the shannon entropy," *IEEE Transactions on Information theory*, vol. 37, no. 1, pp. 145–151, 1991.

[41] X. Ouyang, Z. Xie, J. Zhou, G. Xing, and J. Huang, "Clusterfl: A clustering-based federated learning system for human activity recognition," *ACM TOSN*, pp. 1–32, 2022.

[42] L. Tu, X. Ouyang, J. Zhou, Y. He, and G. Xing, "Feddl: Federated learning via dynamic layer sharing for human activity recognition," in *ACM SenSys*, 2021, pp. 15–28.

[43] P. Kirichenko, P. Izmailov, and A. G. Wilson, "Last layer re-training is sufficient for robustness to spurious correlations," *arXiv preprint arXiv:2204.02937*, 2022.

[44] J. Zhang, C. Chen, and L. Lyu, "Ideal: Query-efficient data-free learning from black-box models," *arXiv preprint arXiv:2205.11158*, 2022.

[45] Y. Zhang, R. Jia, H. Pei, W. Wang, B. Li, and D. Song, "The secret revealer: Generative model-inversion attacks against deep neural networks," in *IEEE/CVF CVPR*, 2020, pp. 253–261.

[46] C. Thapa, P. C. M. Arachchige, S. Camtepe, and L. Sun, "Splitfed: When federated learning meets split learning," in *AAAI*, vol. 36, no. 8, 2022.

[47] O. Gupta and R. Raskar, "Distributed learning of deep neural network over multiple agents," *Journal of Network and Computer Applications*, vol. 116, pp. 1–8, 2018.

[48] P. Vepakomma, O. Gupta, A. Dubey, and R. Raskar, "Reducing leakage in distributed deep learning for sensitive health data," *arXiv preprint arXiv:1812.00564*, vol. 2, p. 8, 2019.

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

[50] C. Li, X. Zeng, M. Zhang, and Z. Cao, "Pyramidfl: a fine-grained client selection framework for efficient federated learning," in *ACM MobiCom*, 2022, pp. 158–171.

[51] L. N. Darlow, E. J. Crowley, A. Antoniou, and A. J. Storkey, "Cinic-10 is not imagenet or cifar-10," *arXiv preprint arXiv:1810.03505*, 2018.

[52] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *IEEE CVPR*, 2015, pp. 1–9.

[53] K. Cui, Y. Wang, Y. Zheng, and J. Han, "Shakereader:'read'uhf rfid using smartphone," *IEEE TMC*, 2021.

[54] K. Cui, Q. Yang, L. Shen, Y. Zheng, F. Xiao, and J. Han, "Towards isac-empowered mmwave radars by capturing modulated vibrations," *IEEE TMC*, 2024.

[55] J. Liu, W. Song, L. Shen, J. Han, X. Xu, and K. Ren, "Mandipass: Secure and usable user authentication via earphone imu," in *IEEE ICDCS*. IEEE, 2021, pp. 674–684.

[56] H. Xu, P. Zhou, R. Tan, M. Li, and G. Shen, "Limu-bert: Unleashing the potential of unlabeled data for imu sensing applications," in *ACM SenSys*, 2021, pp. 220–233.

[57] L. Shen, Q. Yang, Y. Zheng, and M. Li, "Autoiot: Llm-driven automated natural language programming for aiot applications," in *ACM MobiCom*, 2025.

[58] X. Zheng, K. Yang, J. Xiong, L. Liu, and H. Ma, "Pushing the limits of wifi sensing with low transmission rates," *IEEE TMC*, 2024.

[59] X. Leiyang, Z. Xiaolong, and L. Liang, "Vortex em wave-based rotation speed monitoring on commodity wifi," *Chinese Journal of Electronics*, 2024.

[60] L. Shen, Q. Yang, X. Huang, Z. Ma, and Y. Zheng, "Gpiot: Tailoring small language models for iot program synthesis and development," in *ACM SenSys*, 2025.

[61] S. Ji, X. Zhang, Y. Zheng, and M. Li, "Construct 3d hand skeleton with commercial wifi," in *ACM SenSys*, 2023, pp. 322–334.

[62] L. Guo, S. Guo, L. Wang, C. Lin, J. Liu, B. Lu, J. Fang, Z. Liu, Z. Shan, and J. Yang, "Wiar: A public dataset for wifi-based activity recognition," *IEEE Access*, vol. 7, pp. 154 935–154 945, 2019.

[63] R. Xiao, J. Liu, J. Han, and K. Ren, "Onefi: One-shot recognition for unseen gesture via cots wifi," in *ACM SenSys*, 2021.

[64] A. Li, J. Sun, B. Wang, L. Duan, S. Li, Y. Chen, and H. Li, "Lotteryfl: Empower edge intelligence with personalized and communication-efficient federated learning," in *IEEE/ACM SEC*, 2021, pp. 68–79.

[65] T. H. Hsu, H. Qi, and M. Brown, "Measuring the effects of non-identical data distribution for federated visual classification," *CoRR*, vol. abs/1909.06335, 2019.

[66] C. Keçeci, M. Shaqfeh, H. Mbayed, and E. Serpedin, "Multi-task and transfer learning for federated learning applications," *CoRR*, 2022.

[67] A. M. Abdelmoniem and M. Canini, "Towards mitigating device heterogeneity in federated learning via adaptive model quantization," in *Proceedings of the 1st Workshop on Machine Learning and Systems*, 2021, pp. 96–103.

[68] T. Dettmers, "8-bit approximations for parallelism in deep learning," *arXiv preprint arXiv:1511.04561*, 2015.

[69] A. Li, J. Sun, X. Zeng, M. Zhang, H. Li, and Y. Chen, "Fedmask: Joint computation and communication-efficient personalized federated learning via heterogeneous masking," in *ACM SenSys*, 2021.

[70] X. Ma, S. Jeong, M. Zhang, D. Wang, J. Choi, and M. Jeon, "Cost-effective on-device continual learning over memory hierarchy with miro," in *ACM MobiCom*, 2023, pp. 1–15.

[71] H. He, S. Chen, K. Li, and X. Xu, "Incremental learning from stream data," *IEEE Transactions on Neural Networks*, vol. 22, no. 12, pp. 1901–1914, 2011.

[72] B. Gong, T. Xing, Z. Liu, W. Xi, and X. Chen, "Towards hierarchical clustered federated learning with model stability on mobile devices," *IEEE TMC*, 2023.

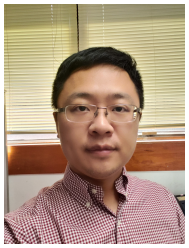
[73] T. E. de Campos, B. R. Babu, and M. Varma, "Character recognition in natural images," in *International conference on computer vision theory and applications*, vol. 1. SCITEPRESS, 2009, pp. 273–280.

[74] S. Saha and T. Ahmad, "Federated transfer learning: concept and applications," *Intelligenza Artificiale*, vol. 15, no. 1, pp. 35–44, 2021.
[75] Y. Chen, X. Qin, J. Wang, C. Yu, and W. Gao, "Fedhealth: A federated transfer learning framework for wearable healthcare," *IEEE Intelligent Systems*, vol. 35, no. 4, pp. 83–93, 2020.
[76] D. J. Beutel *et al.*, "Flower android example (tensorflowlite)," https://github.com/adap/flower/tree/main/examples/android, 2023.
[77] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous systems," 2015.
[78] A. Khan, A. Sohail, U. Zahoora, and A. S. Qureshi, "A survey of the recent architectures of deep convolutional neural networks," *Artificial intelligence review*, vol. 53, pp. 5455–5516, 2020.
[79] L. Alzubaidi, J. Zhang, A. J. Humaidi, A. Al-Dujaili, Y. Duan, O. Al-Shamma, J. Santamaría, M. A. Fadhel, M. Al-Amidie, and L. Farhan, "Review of deep learning: concepts, cnn architectures, challenges, applications, future directions," *Journal of big Data*, 2021.
[80] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *NeurIPS*, 2017, pp. 5998–6008.
[81] J. Shu, T. Yang, X. Liao, F. Chen, Y. Xiao, K. Yang, and X. Jia, "Clustered federated multitask learning on non-iid data with enhanced privacy," *IEEE Internet Things J.*, 2023.
[82] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?" in *Advances in Neural Information Processing Systems 27 2014, NeurIPS*, 2014.
[83] J. Liu, J. Yan, H. Xu, Z. Wang, J. Huang, and Y. Xu, "Finch: Enhancing federated learning with hierarchical neural architecture search," *IEEE TMC*, vol. 23, no. 5, pp. 6012–6026, 2023.
[84] J. Liu, S. Wang, H. Xu, Y. Xu, Y. Liao, J. Huang, and H. Huang, "Federated learning with experience-driven model migration in heterogeneous edge networks," *IEEE/ACM TON*, 2024.
[85] W. Huang, M. Ye, and B. Du, "Learn from others and be yourself in heterogeneous federated learning," in *IEEE/CVF CVPR*, 2022, pp. 10 143–10 153.
[86] L. Shen and Y. Zheng, "Feddm: Data and model heterogeneity-aware federated learning via dynamic weight sharing," in *2023 IEEE ICDCS*. IEEE, 2023, pp. 975–976.
[87] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, "Rethinking the value of network pruning," in *ICLR*, 2019.
[88] N. Tonellotto, A. Gotta, F. M. Nardini, D. Gadler, and F. Silvestri, "Neural network quantization in federated learning at the edge," *Information Sciences*, vol. 575, pp. 417–436, 2021.

Leming Shen is a PhD student in the Department of Computing, The Hong Kong Polytechnic University. Prior to that, he received his B.E. degree in Software Engineering from Zhejiang University. His research interest mainly lies in Large Language Models, Mobile/Edge Computing, and IoT Applications. He is a student member of IEEE and ACM.

Qiang Yang is a postdoctoral research associate in the Department of Computer Science and Technology, University of Cambridge, UK. He obtained his Ph.D. degree from The Hong Kong Polytechnic University in 2023. Prior to that, he received M.E. and B.E. degrees in Computer Science from Shenzhen University, China, and Henan University, China, respectively. His research interests include mobile health, mobile and ubiquitous computing, and IoT. He is a member of IEEE.

Kaiyan Cui is currently an assistant professor at the School of Computer Science, Nanjing University of Posts and Telecommunications, Nanjing, China. She received the B.S. degree from Taiyuan University of Technology, Taiyuan, China, in 2016, and the Joint Ph.D. degree from Hong Kong Polytechnic University, Hong Kong, China, and Xi'an Jiaotong University, Xi'an, China, in 2023. Her research interests include smart sensing, mobile computing, and IoT. She is a member of the IEEE.



Yuanqing Zheng is an associate professor in the Department of Computing, the Hong Kong Polytechnic University. He received the Ph.D. degree in Computer Science from Nanyang Technological University, Singapore. He received the B.S. degree in Electrical Engineering and the M.E. degree in Communication and Information System both from Beijing Normal University, Beijing, China. His research interest includes wireless networking and mobile computing, acoustic and wireless sensing, and IoT. He is a senior member of IEEE and ACM.

Xiao-Yong Wei is a professor and the head of the Department of Computer Science, Sichuan University of China since 2010, and is a visiting professor of the Department of Computing, The Hong Kong Polytechnic University. His research interests include Multimedia Computing, Health Computing, Machine Learning, and Large-Scale Data Mining. He served as an associate editor of Interdisciplinary Sciences: Computational Life Sciences since 2020, the program chair of ICMR 2019, ICIMCS 2012, and the technical committee member of over 20 conferences such as ICCV, CVPR, SIGKDD, ACM MM, ICME, and ICIP. He is a senior member of IEEE.

Jianwei Liu received the Ph.D. degree from Zhejiang University. He is now a postdoctoral researcher at Zhejiang University and Hangzhou City University. His research interests include wireless sensing, the Internet of Things, and mobile computing.

Jinsong Han is a professor at the College of Computer Science and Technology, Zhejiang University, Hangzhou, China. He received his Ph.D. degree in Computer Science from Hong Kong University of Science and Technology in 2007. His research interests focus on IoT security, smart sensing, wireless and mobile computing. He is a senior member of IEEE and ACM.