FedConv: A Learning-on-Model Paradigm for Heterogeneous Federated Clients

Leming Shen¹, Qiang Yang¹,², Kaiyan Cui¹,³, Yuanqing Zheng¹, Xiao-Yong Wei⁴,¹, Jianwei Liu⁵, Jinsong Han⁶
¹The Hong Kong Polytechnic University, ²University of Cambridge, ³Nanjing University of Posts and Telecommunications, ⁴Sichuan University, ⁵Zhejiang University, ⁶The University of Hong Kong

ABSTRACT

Federated Learning (FL) facilitates collaborative training of a shared global model without exposing clients’ private data. In practical FL systems, clients (e.g., edge servers, smartphones, and wearables) typically have disparate system resources. Conventional FL, however, adopts a one-size-fits-all solution, where a homogeneous large global model is transmitted to and trained on each client, resulting in an overwhelming workload for less capable clients and starvation for other clients. To address this issue, we propose FedConv, a client-friendly FL framework, which minimizes the computation and memory burden on resource-constrained clients by providing heterogeneous customized sub-models. FedConv features a novel learning-on-model paradigm that learns the parameters of the heterogeneous sub-models via convolutional compression. Unlike traditional compression methods, the compressed models in FedConv can be directly trained on clients without decompression. To aggregate the heterogeneous sub-models, we propose transposed convolutional dilation to convert them back to large models with a unified size while retaining personalized information from clients. The compression and dilation processes, transparent to clients, are optimized on the server leveraging a small public dataset. Extensive experiments on six datasets demonstrate 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).

CCS CONCEPTS
• Human-centered computing → Ubiquitous and mobile computing; • Computing methodologies → Learning paradigms.

KEYWORDS
Federated learning, Model heterogeneity, Model compression

Permission to make this digital or hard copy of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

1 INTRODUCTION

Federated Learning (FL) allows mobile devices to collaboratively train a shared global model without exposing their private data [8–10, 80, 81]. 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 [53].

In real-world deployments, federated clients typically have diverse system resources, calling for heterogeneous models with different sizes [29, 38, 86]. As shown in Fig. 1, high-end PCs can support large models, while wearables cannot. Simply assigning the smallest affordable model to all clients results in resource underutilization and sub-optimal performance.

Previous solutions that generate heterogeneous models mainly include knowledge distillation (KD) [46, 48], parameter sharing [22], and parameter pruning [20, 42]. KD distills the knowledge from heterogeneous client models to a global model for aggregation. Nonetheless, it imposes additional compute overhead on clients [54] as they must first train on public data and then transfer knowledge via private data. 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. Parameter pruning methods utilize 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 (§2.2). 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.

Ideally, the heterogeneous sub-models should retain the information of the global model in a way that they can 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 via various receptive fields while preserving 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 the parameters of heterogeneous sub-models via convolution while retaining the global model’s prediction capability? To tackle this problem, 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 that of the global model. 2) How to preserve clients’ personalized information after converting their models to a unified size for aggregation? We apply separate TC operations on each client’s model parameters and learn a set of dilated models, which inherit their personalized information. We add a residual connection to further enhance the transfer of personalized information from client models to dilated models. 3) How to aggregate these dilated models with imbalanced contributions of heterogeneous federated clients? As client models are trained on the non-independent and identically distributed (non-IID) personalized data, directly averaging [53] these large models would lead to performance degradation. To tackle this issue, 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 FedConv1 based on a user-friendly FL framework (Flower [5]) 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 our knowledge, FedConv is the first model compression method based on convolution operations. This paradigm can not only compress the global model effectively, but also preserve its 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 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 based on Flower and conduct comprehensive evaluations with heterogeneous mobile devices. The results demonstrate the superior performance of FedConv in terms of both inference accuracy and resource efficiency.

## 2 MOTIVATION

In this section, we underscore the necessity of model heterogeneity-aware FL systems and analyze SOTA works (parameter sharing and model pruning) to motivate our work. Knowledge distillation-aware FL systems and analyze SOTA works (parameter sharing and model pruning) to motivate our work. Knowledge distillation-based methods incur heavy overhead on clients (§6.3), which is not suitable for resource-constrained mobile devices.

### 2.1 Necessity of Heterogeneous Models

In a conventional FL system, all clients typically share the same model architecture. In practice, however, different clients have diverse computation and communication resources. For example, high-end edge PCs usually have more resources, while low-cost embedded systems have much constrained resources. Therefore, the size of a global model is typically upper-bounded by the clients with the least system resources in conventional FL. 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 [45, 62, 67]. 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.

### 2.2 Limitations of Existing Solutions

**Imbalance problem in parameter sharing.** HeteroFL [22] is a representative parameter sharing scheme where clients share different regions of the global model. As shown in Fig. 2(a), the shared portions (the overlapped part across different sizes of models) are fixed, and the parameters are aggregated only from clients that hold them, missing the information from other clients. To showcase its impact, we train a ResNet18 model [31] on the CIFAR10 dataset [40] with 100 global rounds. We 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 [4, 82]. FedRolex [4] proposes a dynamic sharing scheme to tackle the imbalance issue. It 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 on all client-side datasets. However, since different clients contribute distinct parts, the aggregated parameters comprise mixed windows.
from the diverse sub-models. As a result, the distribution of the global model’s parameters is distorted and thus cannot effectively extract useful features from the input data [70], leading to degraded performance and a longer convergence time.

**Information loss and client workload in model pruning.** As shown in Fig. 2(c), model pruning can be categorized into channel-level and filter-level pruning. Channel-level pruning removes some input channels from the model parameters, where the corresponding channels of the input data are also excluded from the training process. Filter-level pruning prunes out some output channels (filters), resulting in fewer output 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 in model parameters [7]. To study the information loss, we apply channel-level and filter-level pruning to the pre-trained ResNet18 model based on the parameter magnitude ranking. We measure the mutual information (MI) $I(X, Z)$, which quantifies the amount of information that can be inferred from $X$ after observing $Z$ [13]. 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.

**2.3 Model Compression via Convolution**

Ideally, a compression method should minimize the information loss of model parameters to retain the performance after compression, without posing extra computation or communication burden 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 information of the global model (e.g., parameter distributions and patterns). Our preliminary studies find that by applying refined convolution operations via various receptive fields [84], 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 data, similar to the features extracted by the large global model.

Fig. 3 shows the convolution-based compression process. To showcase that a compressed model generated by convolutional compression (convolution layers) can effectively extract features from the input data, we compress the pre-trained model described in § 2.2 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 [68]) 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 optimize the compression process, same as existing knowledge distillation-based FL systems [46, 48] that need server-side data during FL training, we also maintain a small publicly available dataset on the server to fine-tune the compression process (§ 4.1). The server-side data can be collected from public datasets, or crowdsourced by volunteers who are willing to share their data. We note that same as conventional FL schemes, clients do not need to send their data to the server. By leveraging such server-side data with iterative refining of the compression process, the server can gradually gain a comprehensive global view [20] of the entire FL process and thus transfer more general information from the server to heterogeneous clients [3].
3 FRAMEWORK OVERVIEW

Fig. 4 illustrates the architecture of FedConv, consisting of three main modules: convolutional compression (§ 4.1), transposed convolutional dilation (§ 4.2), and weighted average aggregation (§ 4.3).

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 (§ 3). 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 [20]. Then, based on the SRs, a set of fine-tuned convolution parameters are used to compress the global model with the convolutional compression module, and generate heterogeneous sub-models (②). Afterwards, the server sends the heterogeneous sub-models to federated clients (③). Clients then perform several epochs of local training with their private training dataset to fine-tune the received sub-models (④), and then upload the updated parameters to the server (⑤). 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.

4 FRAMEWORK DESIGN

4.1 Convolutional Compression

The convolutional compression module leverages a set of convolutional layers (termed as compression layers) to compress the global model and generate heterogeneous sub-models. As shown in Fig. 3, after feeding the global model parameters to the compression layers, the compressed parameters of the sub-model become smaller and output fewer feature maps. We use the server-side data to iteratively optimize the convolution parameters (i.e., the parameters of compression layers) until the sub-models can achieve comparable performance to the global model, as they inherit the parameter information from the global model 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.

Convolution configurations. To determine the sizes of the submodels, clients first specify their shrinkage ratios (SRs). Specifically, the server first broadcasts the size of the global model to all the clients. Then, each client will determine an appropriate SR for the corresponding sub-model to meet its own computing resource budget (e.g., GPU memory, network bandwidth). Subsequently, the SRs are transmitted back to the server. We note that same as conventional FL schemes, no client-side sensor data needs to be transferred to the server. Accordingly, the server determines the corresponding configurations (i.e., input channel in, output channel out, kernel size (k1, k2), stride s, and padding p) of the compression layers so that the sizes of generated sub-models match with the expected SRs. Let’s take convolution layers as an example. As shown in Fig. 3, a convolutional layer in the global model has 16 input and 32 output channels with a kernel size of (3, 3). Regarding each element in the kernel as a single unit, we can reshape its parameter matrix from (32, 16, 3, 3) to 9 × (1, 32, 16). Suppose the SR is 0.75, the shape of the parameter matrix becomes 9 × (1, 24, 12) after compression. Therefore, we use nine separate 2D convolutional layers (i.e., compression layers) to compress the reshaped matrix. The configuration of each compression layer is Conv(in=1, out=1, k=(9, 5), s=1, p=0). 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 will not be 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 need to fine-tune the convolution parameters so that the generated sub-models inherit the parameter information from the global model and achieve comparable performance. We use the server-side data to iterate the convolution parameters by minimizing the loss between the ground truth and the prediction result of the compressed model:

\[
\min_{w_{\text{Conv},l}} \sum_{x} L(f(x; W_{G,l} \circ w_{\text{Conv},l}), y),
\]

s.t. \( \forall l \in \{1, 2, \ldots, L\}, \forall (x, y) \in D \).

where \( L \) is the Cross-entropy loss [75] and \( f(\cdot) \) is the forward function of the compressed model. \((x, y)\) is the data and the corresponding label in the server-side data \( D \). \( W_{G,l} \) is the parameters of the \( l \)-th layer in the global model. \( w_{\text{Conv},l} \) is the convolution parameters. \( \circ \) is the convolution operation. To fine-tune the convolution

---

2The resource profiling process can be performed by existing tools, e.g., nn-Meter [87].

3By default, we set the stride and padding as one and zero, respectively. We will vary their values and the kernel size to explore the impact in § 6.4.4.
parameters, the compressed sub-model \( W_{GL} \otimes w_{Conv1} \) is first evaluated on the server-side data. By back-propagating the calculated loss, the convolution parameters \( w_{Conv1} \) is updated while others (i.e., the parameters of the global model and the sub-model) are frozen.

**Remarks.** In the model compression process, the server applies compression layers on the global model parameters and iteratively fine-tunes the convolution parameters to learn heterogeneous sub-models, aiming to preserve crucial parameter information and prediction capability from the global model. Such a learning-on-model method fundamentally differs from the traditional learning-on-data paradigm. Specifically, the learning-on-data method takes raw data as input and trains 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 fine-tuned by minimizing the loss between the sub-model outputs and the ground-truth labels.

**Challenges.** Nonetheless, several practical challenges emerge during this compression process. We use a pre-trained model on the MNIST [19] dataset (with an accuracy of 99.04%) as an example to show how we address these challenges and the progress we make.

1. **Information loss.** After fine-tuning the convolution parameters as aforementioned, we find that the amount of the parameter information in the sub-model inferred from the global model is still low (the mutual information between the parameters of the global model and the sub-model is only 0.84). This can be attributed to the limited capability of simple compression layers to capture more fine-grained parameter information effectively, leading to a lower accuracy of the sub-model (90.2%). To address this issue, we add two \( 1 \times 1 \) convolutional layers with biases before compression (Fig. 5(a)). Intuitively, the first \( 1 \times 1 \) increases the number of output channels to 16, capturing more diverse and complex parameter information in the global model. The second \( 1 \times 1 \) decreases the number back to one, fusing information from different channels and producing a comprehensive parameter representation. In addition, we add a residual connection between the global model parameters and the output of the second \( 1 \times 1 \), facilitating the transfer of parameter information from the global model to sub-models through convolution operations. With these designs, the accuracy of the sub-models increases to 93.15%, indicating that the information loss is effectively mitigated.

2. **Imbalanced parameter distribution.** As shown in Fig. 6(a), although the distributions of the parameters in sub-models and the global model are similar, the parameters in sub-model skew towards negative values, leading to numerical instability, slow convergence, and unexpected performance degradation [27]. Therefore, we adopt a modified Leaky ReLU (MLR) activation function (Fig. 6(b)) to rectify the negative value, where \( s_n \) and \( s_p \) are slopes for negative and positive values, respectively. With a small \( s_n \), the MLR 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. 6(a)). The accuracy of the sub-model further increases to 95.06%.

3. **Performance fluctuation.** During the fine-tuning process, 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 the convolution parameters are updated, which subsequently generate sub-model parameters via the convolution process. As a result, the performance of the sub-model exhibits much higher sensitivity to the changes in convolution parameters. To address this issue, we apply weight normalization [60] on the convolution parameters to decouple their magnitude and direction during updating, which stabilizes the convergence in a fine-grained way. Moreover, we apply a cosine annealing learning rate scheduler [35] that dynamically varies the learning rate to avoid local optima and enables faster convergence [41]. The learning rate undergoes a cosine function decay as the epoch progresses:

\[
    lr = lr_{\text{min}} + 0.5(lr_{\text{max}} - lr_{\text{min}})(1 + \cos(\pi e / T_{\text{max}}))
\]

where \( e \) is the current epoch index, \( lr_{\text{min}} \) and \( lr_{\text{max}} \) are the lower and upper bound of the learning rate, and \( T_{\text{max}} \) is the maximum number of iterations before \( lr \) restarts to \( lr_{\text{min}} \). As shown in Fig. 7(a), after applying weight normalization and the learning rate 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.

After fine-tuning the convolution parameters, the compressed models will have comparable performance to the global model. The server then sends the compressed parameters to the corresponding clients for local training. The fine-tuned convolution parameters will be kept on the server and updated in the next communication round. Note that this process is performed completely on the server without imposing any extra computation or communication burden on clients. The detailed process is shown in Algorithm 1.

4.2 Transposed Convolutional Dilation

Upon receiving the updated sub-models from clients, we need to rescale the heterogeneous client models to a unified size for further aggregation. Although knowledge distillation-based methods [46, 48] are promising, they impose significant computational and communication overhead on clients (§ 1). Instead, we use transposed convolution (TC) layers on the server side, a reverse operation to the dilation process (detailed in Fig. 5(b)).

TC configurations. To transform the heterogeneous models from different clients to a unified size, it is important to ensure that the configurations of each TC layer for dilation are identical to the corresponding compression layer. For instance, as illustrated in Fig. 5(b), a convolutional layer in a client model has 12 input and 24 output channels with a kernel size of (3, 3). With the SR as 0.75, the configurations of the TC layer should be TC(in=1, out=1, k=(9, 5), s=1, p=0). Similarly, this process can also be employed to dilate other kinds of network layers. Note that the input channel number of the first layer and the output channel number of the last layer in all client models are also unchanged.

TC parameter fine-tuning. To fine-tune the TC parameters, we also set them as learnable variables and minimize the loss between the ground truth and the prediction result of the dilated large model:

\[
\min_{w_{TC,j}} \sum_{x,y} L(F(x; W_{C,j} \odot w_{TC,j}, y)),
\]

s.t. \( \forall i \in \{1, 2, \cdots, L\}, \forall (x, y) \in D \).

where \( F(\cdot) \) is the forward function of the dilated large model, \( W_{C,j} \) is the parameters of the client model, \( w_{TC,j} \) denotes the TC parameters, and \( \odot \) represents the TC operation. To further enhance the integration of personalized information into the dilated models, we also add two TC 1 x 1 layers with a residual connection before the dilation process (detailed in Fig. 5(b)).

4.3 Weighted Average Aggregation

After generating a set of dilated large models, the server aggregates them to obtain the global model. However, we find that directly averaging [53] the parameters of all the dilated models leads to severe performance degradation (the accuracy of the aggregated model is only 47.6% on the MNIST dataset). The reasons are two-folded: 1) the magnitude of the dilated models’ parameters varies with their sizes [20]; 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. 7(b)). Simply aggregating these dilated models overlooks the diverse contributions that heterogeneous clients can make in the aggregation process.

Contribution coordination. To fuse and balance the diverse personalized information from the dilated models, we first normalize the parameters of all the dilated models to [0, 1] and then assign different learnable weight vectors to every network layer in each dilated model for the weighted aggregation. The parameters of the \( l \)-th aggregated network layer are then expressed as:

\[
W_l = (\sum_{j=1}^n p_{j,l} \cdot s_j \cdot w_{j,l})//s_j
\]

where \( W_l \) is the parameters of the \( l \)-th layer in the aggregated model, \( n \) is the number of large models, \( w_{j,l} \) and \( p_{j,l} \) 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 \( p_{j,l} \) via gradient descent to gradually balance the distinct contributions.

Aggregation enhancement. To further quantify the different contributions of heterogeneous clients and enhance the aggregation process, we use Kullback-Leibler Divergence (KLD) [47] as a practical criterion to measure the similarity between the parameters of the global model and the dilated models. The higher

---

**Algorithm 1: Convolutional compression**

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>/* Configuration initialization */;</td>
</tr>
<tr>
<td>2</td>
<td>if ( r = t ) then</td>
</tr>
<tr>
<td>3</td>
<td>Initialize the global model as ( w(r) );</td>
</tr>
<tr>
<td>4</td>
<td>Pre-train ( w(r) ) for ( e_p ) epochs on ( D );</td>
</tr>
<tr>
<td>5</td>
<td>for device_type ( i \in {1, 2, \cdots, n} ) parallel do</td>
</tr>
<tr>
<td>6</td>
<td>current_shrinkage_ratio ( \leftarrow SR_i );</td>
</tr>
<tr>
<td>7</td>
<td>( Con_{0}(r) \leftarrow \text{Initialize_Conv}(w(r), SR_i) );</td>
</tr>
<tr>
<td>8</td>
<td>end</td>
</tr>
<tr>
<td>9</td>
<td>else</td>
</tr>
<tr>
<td>10</td>
<td>( Con_{0}(r) \leftarrow Con_{0}(r-1) );</td>
</tr>
<tr>
<td>11</td>
<td>end</td>
</tr>
<tr>
<td>12</td>
<td>/* Convolution parameters fine-tuning */;</td>
</tr>
<tr>
<td>13</td>
<td>for device_type ( i \in {1, 2, \cdots, n} ) parallel do</td>
</tr>
<tr>
<td>14</td>
<td>for ( e \in {1, 2, \cdots, e_p} ) do</td>
</tr>
<tr>
<td>15</td>
<td>( IR = IR_{\text{min}} + 0.5(\max(IR_{\text{max}} - IR_{\text{min}}) + \cos(\pi/2)) );</td>
</tr>
<tr>
<td>16</td>
<td>for each ( (x, y) ) in ( D ) do</td>
</tr>
<tr>
<td>17</td>
<td>( P_l \leftarrow \text{MLR}(w(r) \odot Con_{0}(r)) );</td>
</tr>
<tr>
<td>18</td>
<td>output ( \leftarrow f(P_l; x) );</td>
</tr>
<tr>
<td>19</td>
<td>loss ( \leftarrow \text{Loss_fn}(\text{output}, y) );</td>
</tr>
<tr>
<td>20</td>
<td>Back-propagate gradient ( g_i ) to ( Con_{0}(r) );</td>
</tr>
<tr>
<td>21</td>
<td>( Con_{0}(r) \leftarrow Con_{0}(r) - IR \cdot g_i );</td>
</tr>
<tr>
<td>22</td>
<td>end</td>
</tr>
<tr>
<td>23</td>
<td>end</td>
</tr>
<tr>
<td>24</td>
<td>end</td>
</tr>
<tr>
<td>25</td>
<td>Send ( P_l ) to the corresponding client;</td>
</tr>
</tbody>
</table>
We select two representative mobile applications and use different methods to evaluate. We evaluate FedConv. We implement FedConv; the model parameters from the previous communication round. The model parameters from the previous communication round. The aggregated global model can at least achieve 2 seconds is applied to generate 900-dimensional features for each type of sensing heterogeneity. As there is no standard model for these datasets, we use a CNN model with three convolution layers and one FC layer that can achieve high accuracy.

We divide these datasets into four parts: 1) the IID server-side global data for convolution/TC parameters and weight vectors tuning, 2) IID test data for evaluating the aggregated global model, 3) client-side training and 4) testing data (IID or non-IID). Each part counts for 5%, 20%, 70%, and 5% of the total dataset, respectively. The first and second parts of the dataset are kept on the server, whereas the third and fourth parts are distributed among heterogeneous clients.

Besides, to emulate real-world heterogeneity, we employ different datasets on the server and clients ($\S$ 6.7).

### 5.3 Baselines

We compare FedConv with the following baselines: 1) Serveralone trains one model with only the server-side global 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 model locally using their private data without parameter exchange. 3) FedAvg [53] is a classic FL paradigm where clients collaboratively train a shared global model and upload the updated model parameters to a central server for averaging aggregation. Due to the constrained resources of some devices, we assign the smallest affordable models to all clients. 4) FedMD [46] utilizes knowledge distillation to reach a consensus among heterogeneous client models through training on a public dataset. 5) LotteryFL [43] generates sub-models by exploiting the Lottery Ticket hypothesis on heterogeneous clients for personalization. 6) Hermes [42] finds a sparse sub-model for each client by using a channel-wise pruning scheme to reduce the communication overhead. 7) TailorFL [20] produces sub-models by filter-level pruning based on the learned importance of each filter. 8) HeteroFL [22] is a parameter sharing method that allows each client to select a subset of the parameters from the global model. 9) FedExConv [4] adopts dynamic rolling windows when extracting sub-models for heterogeneous clients.

### 5.4 Heterogeneity Consideration

For model heterogeneity, we consider four SRs: 0.25, 0.5, 0.75, and 1.0, according to the resource profiles of the heterogeneous clients.
clients. The SR for each client is detailed in Table 1. For powerful clients, we assign larger SRs (e.g., 0.75 for laptops), and for resource-constrained clients, we assign smaller SRs (e.g., 0.25 for Raspberry Pis). For data heterogeneity, we sample the disjoint non-IID client-side data using the Dirichlet distribution Dir(α). A larger α (e.g., 10000) indicates a more homogeneous distribution and a smaller α (e.g., 0.05) generates a more heterogeneous distribution [33], The sample distribution among different classes is illustrated in Fig. 8.

5.5 Hyper-parameter Settings
For baselines and FedConv, we set the number of communication rounds to 100. Each client performs 5 local training epochs with a learning rate of 0.001. In the model compression and dilation process, 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 $T_{max}$, $T_{min}$, $T_{max}$ in the cosine annealing scheduler are 4, 0.00001, and 0.001, respectively. $s_p$ and $s_t$ (Fig. 6(b)) in 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 $\lambda$ in Eq. (4) are 10, 0.001, and 0.2, respectively.

6 EVALUATION

6.1 Metrics
Training performance: 1) Inference accuracy: we measure the global model accuracy with the server-side test dataset to evaluate the generalizability of the global model. We also report the average client model accuracy with client-side private test datasets to evaluate the effectiveness of personalization. 2) Communication cost: we use the Pympler library to monitor the network traffic of all the clients over 100 communication rounds.

Runtime performance: 1) Memory footprint: the real-time GPU memory usage is monitored using the PyTorch CUDA Toolkit. We track each client’s process ID over 100 communication rounds to monitor its 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 the training task, and report the average wall-clock time in each round.

6.2 Overall Performance
We evaluate the overall performance of FedConv with heterogeneous models and data distribution.

6.2.1 Global model performance. We first evaluate the accuracy of the aggregated global model to demonstrate its generalizability. Standalone and FedMD are excluded because they do not create global models. Fig. 9(a) shows the global model accuracy under the same degree of heterogeneous data ($\alpha = 0.1$). Serveralone achieves a higher global model accuracy than the baselines in most cases, as the server-side data for training and testing 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 other baselines (FedAvg and LotteryFL), 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. 10(a) shows the global model accuracy of FedConv and all baselines across different data heterogeneity on all datasets. We can see that the performance enhancement of FedConv becomes more significant as $\alpha$ 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 (§6.2.2). 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.

### 6.2.2 Client model performance.
To evaluate the personalization performance, we measure the accuracy of each client model with client-side test datasets and report the average value. Fig. 9(b) shows that with the same heterogeneous 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 using the client-side non-IID data, the accuracy of the client model 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. 10(b) shows the client model accuracy of FedConv and all baselines with different data heterogeneity. We can see that the performance disparities become more substantial as \( \alpha \) 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 rescaled large models will thereby preserve the personalization information from clients, which is then aggregated into the global model. Besides, Fig. 9(b) shows that, with sensing heterogeneity in the HARBox dataset, FedConv achieves a better and more stable performance. However, when \( \alpha \) is small (e.g., \( \alpha \in \{0.05, 0.1\} \) on CIFAR10), the client model accuracy of FedMD is higher than FedConv. The better performance stems from the distilled knowledge edge shared by all clients. Nonetheless, the downside is that it imposes excessive communication and computational overhead on clients (Table 2 & Table 3). 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 [39] (detailed in §6.6).

#### 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 model pruning and the imbalance issue in parameter sharing.

### 6.3 Overhead Assessment
We evaluate the memory footprint, wall-clock time, and communication overhead of each client in FedConv and baselines with both homogeneous (\( \alpha=10000 \)) and heterogeneous (\( \alpha=0.05 \)) data across clients. Table 2 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 3 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.

#### Table 2: System resource overhead.

<table>
<thead>
<tr>
<th>Metric</th>
<th>System</th>
<th>MNIST</th>
<th>CIFAR10</th>
<th>CLINIC10</th>
<th>WiAR</th>
<th>DCD</th>
<th>HARBox</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standalone</td>
<td>2.14</td>
<td>3.51</td>
<td>4.07</td>
<td>3.95</td>
<td>2.24</td>
<td>2.19</td>
<td>2.13</td>
</tr>
<tr>
<td>FedAvg</td>
<td>1.90</td>
<td>2.40</td>
<td>3.31</td>
<td>2.98</td>
<td>2.01</td>
<td>2.10</td>
<td>2.01</td>
</tr>
<tr>
<td>FedMD</td>
<td>2.71</td>
<td>3.65</td>
<td>7.51</td>
<td>4.71</td>
<td>2.99</td>
<td>2.71</td>
<td>2.65</td>
</tr>
<tr>
<td>Footprint CPU+GPU</td>
<td>2.62</td>
<td>3.51</td>
<td>4.30</td>
<td>3.23</td>
<td>2.69</td>
<td>2.63</td>
<td>2.93</td>
</tr>
<tr>
<td>Memory</td>
<td>2.64</td>
<td>3.45</td>
<td>6.07</td>
<td>3.28</td>
<td>2.73</td>
<td>2.64</td>
<td>3.35</td>
</tr>
<tr>
<td>Footprint MNIST</td>
<td>2.75</td>
<td>3.61</td>
<td>5.09</td>
<td>3.41</td>
<td>2.79</td>
<td>2.71</td>
<td>3.75</td>
</tr>
<tr>
<td>Footprint DCD</td>
<td>2.63</td>
<td>3.31</td>
<td>4.15</td>
<td>3.25</td>
<td>2.73</td>
<td>2.63</td>
<td>4.35</td>
</tr>
<tr>
<td>Footprint HARBox</td>
<td>2.63</td>
<td>3.21</td>
<td>4.15</td>
<td>3.25</td>
<td>2.72</td>
<td>2.60</td>
<td>4.16</td>
</tr>
</tbody>
</table>

#### Table 3: Communication overhead comparison (GB).

<table>
<thead>
<tr>
<th>System</th>
<th>MNIST</th>
<th>CIFAR10</th>
<th>CLINIC10</th>
<th>WiAR</th>
<th>DCD</th>
<th>HARBox</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standalone</td>
<td>14.80</td>
<td>4815.84</td>
<td>2697.85</td>
<td>28.24</td>
<td>13.75</td>
<td>7.60</td>
</tr>
<tr>
<td>FedAvg</td>
<td>19.99</td>
<td>5126.46</td>
<td>2859.79</td>
<td>40.91</td>
<td>19.94</td>
<td>11.26</td>
</tr>
<tr>
<td>FedMD</td>
<td>44.34</td>
<td>437.14</td>
<td>5370.83</td>
<td>55.03</td>
<td>75.25</td>
<td>43.67</td>
</tr>
<tr>
<td>LotteryFL</td>
<td>9.18</td>
<td>147.89</td>
<td>699.35</td>
<td>8.89</td>
<td>8.61</td>
<td>34.53</td>
</tr>
<tr>
<td>Hermes</td>
<td>43.22</td>
<td>714.00</td>
<td>5580.71</td>
<td>193.90</td>
<td>169.97</td>
<td>43.84</td>
</tr>
<tr>
<td>TailorFL</td>
<td>6.98</td>
<td>62.89</td>
<td>393.46</td>
<td>14.44</td>
<td>12.72</td>
<td>11.57</td>
</tr>
<tr>
<td>HeteroFL</td>
<td>6.96</td>
<td>42.56</td>
<td>641.21</td>
<td>10.78</td>
<td>10.03</td>
<td>10.98</td>
</tr>
<tr>
<td>FedRolex</td>
<td>6.92</td>
<td>45.98</td>
<td>602.48</td>
<td>11.57</td>
<td>12.34</td>
<td>11.26</td>
</tr>
<tr>
<td>FedConv</td>
<td>5.96</td>
<td>40.68</td>
<td>264.30</td>
<td>12.96</td>
<td>10.15</td>
<td>9.89</td>
</tr>
</tbody>
</table>

In conclusion, FedConv provides a learning-on-model paradigm that effectively addresses the challenges of heterogeneous federated learning. It preserves and transfers the personalization information from clients, leading to better generalization and robustness. Additionally, it reduces communication and computational overhead, making it a compelling choice for resource-constrained environments.
FedConv the GoogLeNet model on CINIC10 when the SR is 0.4. Consequently, between the SR and model performance, we set the SR for 10 clients

32.5% higher than that of the baselines. The results demonstrate the number of clients increases from 10 to 50. We then select CIFAR10

the client model accuracy on HARBox increases by 17.54% when the number of selected participating clients from 10 to 50 (\(\alpha\) = 10000).

6.4 Sensitivity Analysis

6.4.1 Varying client number. We simulate 100 clients and vary the number of selected participating clients from 10 to 50 (\(\alpha = 10000\)) to compare the client model performance with the baselines. As shown in Fig. 11(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. 11(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.

6.4.2 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 (\(\alpha = 10000\)). From Fig. 12(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 Pi) 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. 12(b). We can see that 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.

6.4.3 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 0.5%. As shown in Fig. 13(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 [52] or incremental learning [30] techniques can be further applied as more server-side data become available.

6.4.4 Varying hyper-parameters. We vary the number of epochs for fine-tuning the model compression, dilation, and aggregation to evaluate their impact on the personalization performance of client models. We select two datasets (CIFAR10 and HARBox) for demonstration. We first vary the number of epochs for updating the convolution/TC parameters in each global round. Fig. 13(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. 13(c), we can observe that when the number of tuning epochs for updating weight vectors exceeds 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 can capture more comprehensive parameter information, and a smaller stride can capture more fine-grained information. As shown in Fig. 13(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, in our default settings, the kernel size and stride are set as 23 and 1, respectively.

### 6.5 Ablation Study

Next, we conduct ablation studies to investigate the importance of the server-side pre-training process and the weighted average aggregation module, respectively.

#### 6.5.1 Server-side pre-training

Fig. 14(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.

#### 6.5.2 Weighted average aggregation

To demonstrate the impact of our learned 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. 14(b) 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 global model and improve its generalization performance.

### 6.6 Personalization Enhancement

To evaluate the potential in personalization, we extend FedConv by adding task-specific layers [39] on each client, and evaluate the client model accuracy. Specifically, after receiving the parameters from the server, each client appends its own personal layers to the sub-model. By doing so, the personalization performance of each client can be enhanced during local training. We record the average accuracy of client models after 100 communication rounds. Fig. 15 shows the performance improvement on five datasets after applying personalization enhancement. Compared with FedMD, which achieves the highest client model accuracy (§ 6.2.2), 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.

### 6.7 Case Study with Real-World Heterogeneity

In our default configuration, both the server-side and client-side datasets originate from the same domain. To test with real-world heterogeneity and assess its impact, we conduct a case study where the Chars74K dataset [18] is kept on the server, while the MNIST dataset is used for heterogeneous clients. The Chars74K dataset contains images of digits from computer fonts with variations (italic, bold, and normal). In this case, the global model can learn and extract general features (e.g., different shapes of the digits from the Chars74K dataset), while heterogeneous clients can further fine-tune the compressed 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. 16(a) and Fig. 16(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, when we further enhance FedConv with transfer learning strategies [12, 59] on each client to narrow down the domain gap between the server-side and the client-side data, the client models will achieve higher accuracies and even outperform FedMD. This observation indicates that FedConv can be combined with existing federated transfer learning approaches to achieve better performance.
7 DISCUSSION

Privacy Concerns. In addition to transferring model parameters between the server and heterogeneous clients, FedConv requires all the clients to report their SRs before the FL training starts. To determine appropriate SRs, clients will perform resource profiling locally and report to the server. We note that same as conventional FL schemes, no client-side sensor data needs to be 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 [5] 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 encounter 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 [6] in Flower support federated learning setup with Android clients using TensorFlow Lite [1].

Convolutional Compression. As shown in our evaluation, the convolutional compression is effective in compressing large global models and achieves better performance compared with model pruning, parameter sharing, and knowledge distillation-based methods. Additionally, the compression and dilation process is performed on the server side, without imposing any extra burden on the clients. From the client’s perspective of view, they do not need to participate in the pre-training and fine-tuning process and can join throughout the FL processes, which is the same as the conventional FL systems.

8 RELATED WORK

Data heterogeneity. Recent works [56, 58, 65, 72] optimize FL performance under non-IID data. Clustering-based methods [11, 34, 56] group clients according to the distribution of their data or model parameters. For example, ClusterFL [56] captures the intrinsic clustering patterns among clients by measuring the similarity of client models. Shu et al. [64] propose a clustered multi-task federated learning on non-IID data. Personalized FL adopts local fine-tuning [25] or add task-specific layers on client side [39, 83]. For example, pFedMe [23] uses Moreau envelopes as a regularization loss function to decouple the task of optimizing a personalized model from the global model learning. Yosinski et al. [83] 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 model compression. To accommodate heterogeneous clients, recent works mainly compress the global model to reduce communication and computation costs. They can be divided into three categories: 1) Knowledge distillation-based methods [85, 88] generally regard heterogeneous client models as teacher models and learn an aggregated global student model via knowledge distillation (KD). Lin et al. [48] leverage KD and ensemble learning to combine the knowledge from heterogeneous client models. FedMD [46] computes an average consensus to substitute the aggregation process. However, the tuning of KD is performed on clients with a shared dataset, incurring extra overhead for clients; 2) Parameter sharing strategies [61] allow sub-models to share a part of the global model parameters to reduce computation overhead. HeteroFL [22] 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 [82]; 3) Pruning-based methods [73] have gained popularity in heterogeneous FL. Hermes [42] applies a channel-level pruning method to selectively prune out less important channels. TailorFL [20] 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 [51]. 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 [71]) 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.

Convolution and transposed convolution. Convolution can effectively extract useful features from input data by capturing local patterns and spatial relationships [41, 66]. In FedConv, we exploit a novel convolutional compression technique to generate sub-models for heterogeneous clients, which can capture key information embedded in the global model. TC is renowned for its capability of reconstructing super-resolution images from fuzzy ones [24, 26], which is widely adopted in image dilation [16] and semantic segmentation [55]. In our aggregation process, we leverage TC to resize the heterogeneous client models to a unified size for aggregation.

9 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 three key technical modules: 1) a novel model compression scheme that generates heterogeneous sub-models with convolutional compression on the global model; 2) a transposed convolutional dilation module that converts heterogeneous client models back to large models with a unified size; and 3) 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 inference 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.

ACKNOWLEDGMENTS

We sincerely thank our shepherd – Veljko Pejovic, and anonymous reviewers for their constructive comments and invaluable suggestions that helped improve this paper. This work is supported by Hong Kong GRF Grant No. 15206123. This work is also supported by NSFC (Grant No. 62372314, U21A20462, and 62372400), “Pioneer” and “Leading Goose” R&D Program of Zhejiang under grant No. 2024C03287. Yuanqing Zheng is the corresponding author.

APPENDIX

The research artifacts accompanying this paper are available at https://doi.org/10.5281/zenodo.11089994.
REFERENCES


[29] FedConv: A Learning-on-Model Paradigm for Heterogeneous Federated Clients MOBISYS ’24, June 3–7, 2024, Minato-ku, Tokyo, Japan


