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

Leming Shen[1], Qiang Yang[1,2], Kaiyan Cui[1,3], Yuanqing Zheng[1]
Xiao-Yong Wei[4,1], Jianwei Liu[5], Jinsong Han[5]
[1]Department of Computing, The Hong Kong Polytechnic University, Hong Kong SAR, China
[2]Department of Computer Science and Technology, University of Cambridge, Cambridge, UK
[3]School of Computer Science, Nanjing University of Posts and Telecommunications, Nanjing, China
[4]School of Computer Science, Sichuan University, Chengdu, China
[5]College of Computer Science and Technology, Zhejiang University, Hangzhou, China
{leming.shen,qiang.yang,kaiyan.cui}@connect.polyu.hk,csyqzheng@comp.polyu.edu.hk,
cswei@scu.edu.cn,{jianweiliu,hanjinsong}@zju.edu.cn

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

MobiSys'24, June 3–7, 2024, Tokyo, Japan

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

ACM Reference Format:
Leming Shen[1], Qiang Yang[1,2], Kaiyan Cui[1,3], Yuanqing Zheng[1], Xiao-Yong Wei[4,1], Jianwei Liu[5], Jinsong Han[5] . . FedConv: A Learning-on-Model Paradigm for Heterogeneous Federated Clients. In *Proceedings of ACM Conference (MobiSys'24).* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Federated Learning (FL) allows mobile devices to collaboratively train a shared global model without exposing their private data [8–10, 26, 36]. 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 [49].

In real-world deployments, federated clients typically have diverse system resources, calling for heterogeneous models with different sizes. 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 under-utilization and sub-optimal performance.

Previous solutions that generate heterogeneous models mainly include knowledge distillation (KD) [43, 45], parameter sharing [21], and parameter pruning [19, 39]. KD distills the knowledge from heterogeneous client models to a global model for aggregation. Nonetheless, it imposes additional compute overhead on clients [50] 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.*

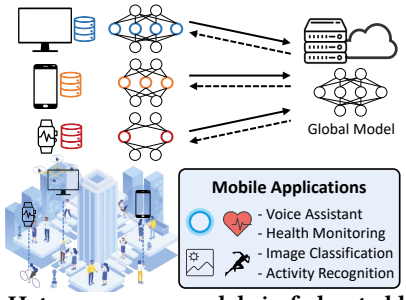


Figure 1: Heterogeneous models in federated learning.

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 [49] 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 *FedConv*[1] 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-based methods incur heavy overhead on clients (§ 6.3), which is not suitable for resource-constrained mobile devices.

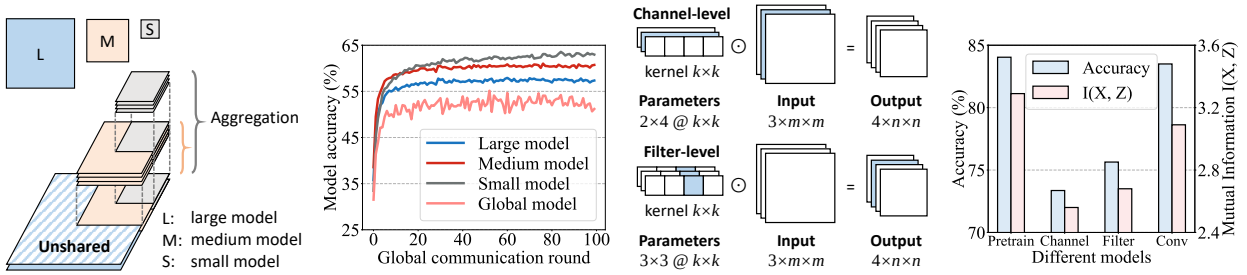
2.1 Necessity of Heterogeneous Models

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 [46, 59] when waiting for weaker clients in synchronized FL [42, 58, 64]. 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 [21] 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 [30] on the CIFAR10 dataset [37] 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, 77]. 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,

[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

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

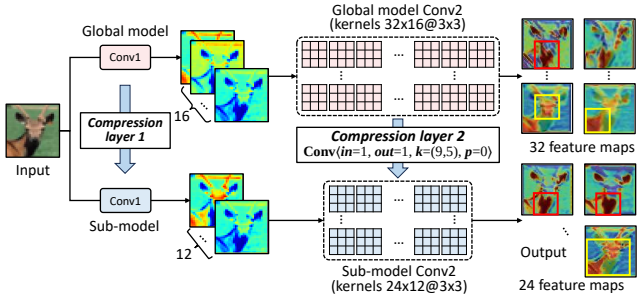


Figure 3: Convolutional compression process.

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 [60, 67], 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 [79], 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* (compression 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 [65]) 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 [43, 45] 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 [19] of the entire FL process and thus transfer more general information from the server to heterogeneous clients [3].

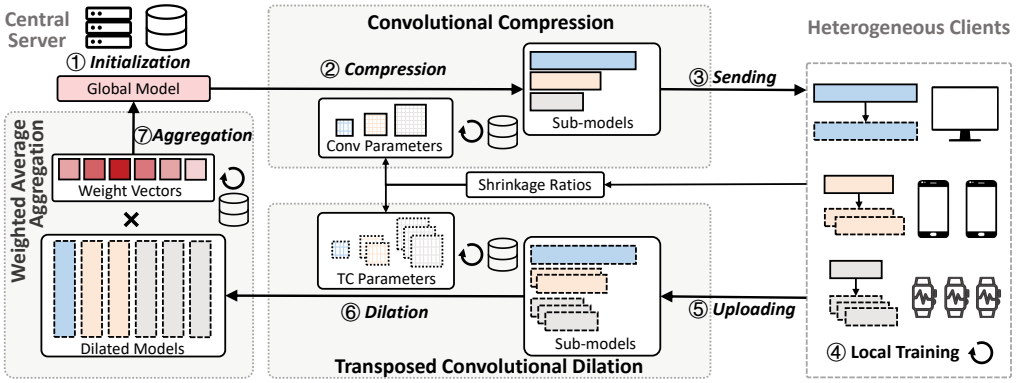


Figure 4: Framework architecture of *FedConv*.

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 (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 [19]. 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.

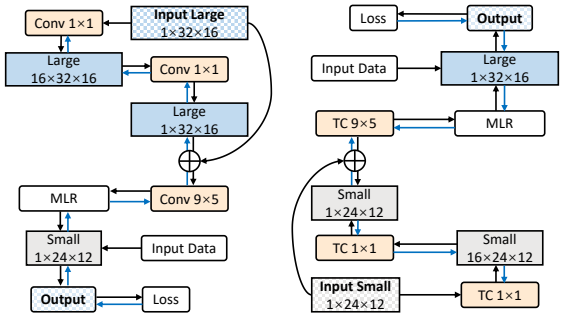
Convolution configurations. To determine the sizes of the sub-models, clients first specify their shrinkage ratios (SRs). Specifically, the server first broadcasts the size of the global model to all the clients. Then, each client will determine an appropriate SR for the corresponding sub-model to meet its own computing resource budget[2] (*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 $\boldsymbol{in}$, output channel $\boldsymbol{out}$, kernel size ($\boldsymbol{k}_1$, $\boldsymbol{k}_2$), stride $\boldsymbol{s}$, and padding $\boldsymbol{p}$) 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 $9 \times (1, 32, 16)$. Suppose the SR is 0.75, the shape of the parameter matrix becomes $9 \times (1, 24, 12)$ after compression. Therefore, we use nine separate 2D convolutional layers (*i.e.*, *compression layers*) to compress the reshaped matrix. The configuration[3] of each *compression layer* is Conv⟨$\boldsymbol{in}$=1, $\boldsymbol{out}$=1, $\boldsymbol{k}$=(9, 5), $\boldsymbol{s}$=1, $\boldsymbol{p}$=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 iteratively adapt the *convolution parameters* by minimizing the loss between the ground truth and the prediction result of the compressed model:

$$\min_{\boldsymbol{w}_{Conv,l}} \sum_{x} \mathcal{L}(f(x; \boldsymbol{W}_{G,l} \odot \boldsymbol{w}_{Conv,l}), y), \quad s.t.\ \forall l \in \{1, 2, \cdots, L\}, \forall (x, y) \in \mathcal{D}. \tag{1}$$

where $\mathcal{L}$ is the Cross-entropy loss [73] 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 $\mathcal{D}$. $\boldsymbol{W}_{G,l}$ is the parameters of the l-th layer in the global model. $\boldsymbol{w}_{Conv,l}$ is the *convolution parameters*. $\odot$ is the convolution operation. To fine-tune the *convolution*

[2]The resource profiling process can be performed by existing tools, *e.g.*, nn-Meter [81].
[3]By 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



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

Figure 5: An example of the convolution/TC process (black arrow: forward, blue arrow: backward, blue box: larger parameters, grey box: smaller parameters, orange: Conv/TC).

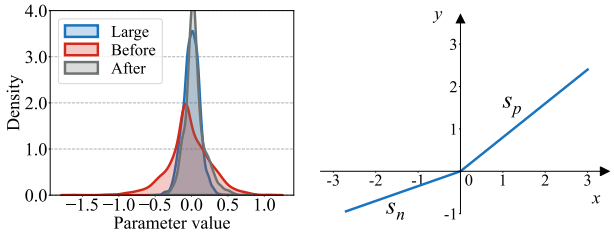
parameters, the compressed sub-model ($\boldsymbol{W}_{G,l} \odot \boldsymbol{w}_{Conv,l}$) is first evaluated on the server-side data. By back-propagating the calculated loss, the *convolution parameters* $\boldsymbol{w}_{Conv,l}$ is updated while others (*i.e.*, the parameters of the global model and 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* method 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. Nevertheless, several practical challenges emerge during this compression process. We use a pre-trained model on the MNIST [18] dataset (with an accuracy of 99.04%) as an example to show how we address these challenges and the progress we make.

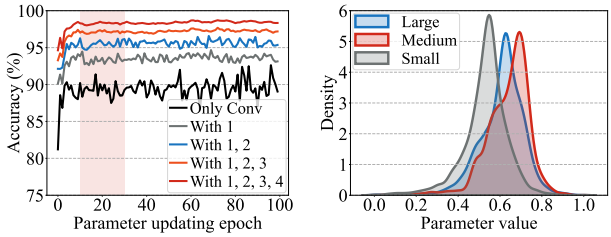
(1) Information loss. After fine-tuning the *convolution parameters* as aforementioned, we find that the amount of the parameter information in the sub-model inferred from the global model is still low (the mutual information between the parameters of the global model and the sub-model is only 0.84). 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 1×1 convolutional layers with biases before compression (Fig. 5(a)). Intuitively, the first Conv 1×1 increases the number of output channels to 16, capturing more diverse and complex parameter information in the global model. The second Conv 1×1 decreases the channel 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 Conv 1×1, 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



(a) Parameter distribution (b) Modified activation function

Figure 6: (a) After using the MLR, the distribution of the parameters in the sub-model becomes similar to the large global model (blue curve); (b) The MLR function.



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

Figure 7: (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 sizes of client models.

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 [56] 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 [34] that dynamically varies the learning rate to avoid local optima and enables faster convergence [38]. The learning rate undergoes a cosine function decay as the epoch progresses:

$$lr = lr_{min} + 0.5(lr_{max} - lr_{min})(1 + \cos(e/T_{max} \cdot \pi)) \quad (2)$$

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}. 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 [43, 45] 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 *convolution compression*. In contrast, we apply different *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 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:

$$\min_{\boldsymbol{w}_{TC,l}} \sum_{x} \mathcal{L}(F(x; \boldsymbol{W}_{C,l} \circledcirc \boldsymbol{w}_{TC,l}), y), \quad s.t.\ \forall l \in \{1, 2, \cdots, L\}, \forall (x, y) \in \mathcal{D}. \tag{3}$$

where $F(\cdot)$ is the forward function of the dilated large model, $\boldsymbol{W}_{C,l}$ is the parameters of the client model, $\boldsymbol{w}_{TC,l}$ denotes the *TC parameters*, and $\circledcirc$ represents the TC operation. To further enhance the integration of personalized information into the dilated models, we also add two TC 1×1 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 [49] 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 [19]; 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.

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 $\{P_1, P_2, \cdots, P_n\}$

```
/* Configuration initialization */;
if r == 1 then
    Initialize the global model as w(r);
    Pre-train w(r) for e_p epochs on D;
    for device_type i ∈ {1, 2, ···, n} parallel do
        current_shrinkage_ratio ← SR_i ;
        Conv_i(r) ← Initialize_Conv(w(r), SR_i);
    end
else
    Conv_i(r) ← Conv_i(r − 1);
end
/* Convolution parameters fine-tuning */;
for device_type i ∈ {1, 2, ···, n} parallel do
    for e ∈ {1, 2, ···, e_c} do
        lr = lr_min + 0.5(lr_max − lr_min)(1 + cos(e/T_max · π));
        for each (x, y) in D do
            P_i ← MLR(w(r) ⊙ Conv_i(r));
            output ← f(P_i; x);
            loss ← Loss_fn(output, y);
            Back-propagate gradient g_i to Conv_i(r);
            Conv_i(r) ← Conv_i(r) − lr · g_i;
        end
    end
end
Send P_i to the corresponding client;
```

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:

$$\boldsymbol{W}_l = (\sum_{j=1}^{n} \boldsymbol{v}_{j,l} \cdot s_j \cdot \boldsymbol{w}_{j,l}) / \sum_{j=1}^{n} s_j \tag{4}$$

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 $\boldsymbol{v}_{j,l}$ 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) [44] as a practical criterion to measure the similarity between the parameters of the global model and the dilated models. The higher

Table 1: 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	-
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

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 formulated 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 expressed as:

$$\mathcal{L}(\boldsymbol{v}) = \mathcal{L}_{\mathcal{D}}(\boldsymbol{W}) + \lambda \sum_{j=1}^{n} KLD_j \tag{5}$$

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.

5 EXPERIMENT SETUP

5.1 Implementation

We implement *FedConv* with PyTorch [53] and Flower [5]. The `load_state_dict()` function in PyTorch is overridden to enable the gradient to back-propagate to the *convolution/TC parameters*. We evaluate *FedConv* with a cloud server, a router, and 20 heterogeneous mobile devices with different hardware and network conditions. Detailed configurations of the heterogeneous devices are described in Table 1. We deploy these edge devices in our offices and laboratories under real-world network conditions.

5.2 Datasets and Models

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

Application#1: Image Classification. Imagine classification is a popular computer vision application for FL. We choose three datasets: 1) **MNIST** [18] consists of 60,000 28×28 gray-scale images of ten handwritten digits. We use a convolutional neural network (CNN) with two convolutional layers and one fully connected layer as the classification model; 2) **CIFAR10** [37] consists of 60,000 32 × 32 color images in ten classes. We use ResNet18 [30] to perform the evaluation; 3) **CINIC10** [16] contains 180,000 32 × 32 color images in ten classes. We use GoogLeNet [66] for evaluation.

Application#2: Human Activity Recognition (HAR). HAR [14, 68, 72, 75, 76] is realized by analyzing different types of sensor data (*e.g.*, Depth camera, IMU, and the channel state information of WIFI signals). We select three datasets: 1) **WiAR** [28] contains 480 90 × 250 Wi-Fi CSI samples of 16 activities. We augment the dataset to 64,000 samples following OneFi [74]; 2) **Depth camera dataset** (DCD) [52] contains 5,000 36 × 36 gray-scale depth images of five common gestures; 3) **HARBox** [52] captures 9-axis IMU data of five daily activities. A sliding window of 2 seconds is applied to generate 900-dimensional features for each of the 30,000 data samples in total. The IMU data was collected from 121 users with 77 different smartphones, demonstrating a degree of sensing heterogeneity. As

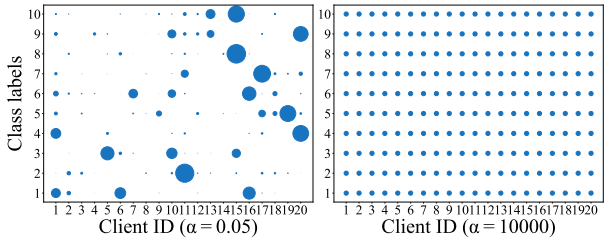


Figure 8: Visualization of Non-IID data. The size of scattered points indicates the number of data samples.

there is no standard model for these datasets, we use a CNN model with three convolutional layers and one FC layer, which can achieve high accuracy with a small number of parameters.

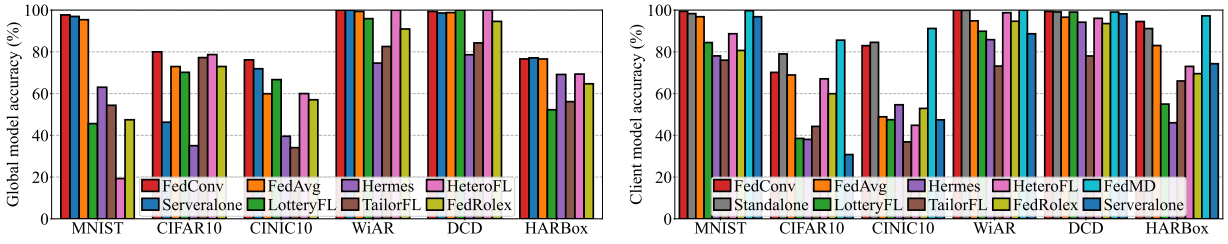
We divide these datasets into four parts: 1) the IID server-side global data for *convolution/TC parameters* and *weight vectors* tuning, 2) IID test data for evaluating the aggregated global model, 3) client-side training and 4) testing data (IID or non-IID). Each part counts for 5%, 20%, 70%, and 5% 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 (§ 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** [49] 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** [43] utilizes knowledge distillation to reach a consensus among heterogeneous client models through training on a public dataset. 5) **LotteryFL** [40] generates sub-models by exploiting the Lottery Ticket hypothesis on heterogeneous clients for personalization. 6) **Hermes** [39] finds a sparse sub-model for each client by using a channel-wise pruning scheme to reduce the communication overhead. 7) **TailorFL** [19] produces sub-models by filter-level pruning based on the learned importance value of each filter. 8) **HeteroFL** [21] is a parameter sharing method that allows each client to select a subset of the parameters from the global model. 9) **FedRolex** [4] adopts dynamic rolling windows when extracting sub-models for heterogeneous clients.

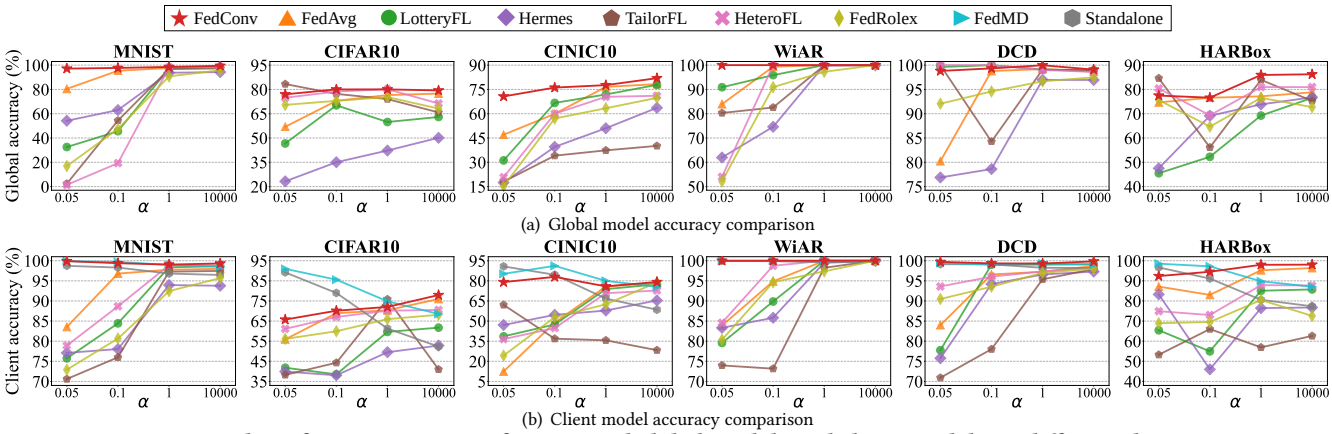
5.4 Heterogeneity Consideration

For model heterogeneity, we consider four SRs: 0.25, 0.5, 0.75, and 1.0, according to the resource profiles of the heterogeneous



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

Figure 9: Accuracy comparison under model heterogeneity ($\alpha = 0.1$).



(a) Global model accuracy comparison

(b) Client model accuracy comparison

Figure 10: The inference accuracy of aggregated global models and client models on different datasets.

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 [32], 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}, lr_{min}, lr_{max} in the cosine annealing scheduler are 4, 0.00001, and 0.001, respectively. s_p and s_n (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 λ 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 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 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 α decreases, meaning that *FedConv* can better cope with the increased data heterogeneity. Although *FedConv* does not obviously outperform FedAvg with homogeneous

Table 2: System resource overhead.

Metric	System	Heterogeneous Data (α = 0.05)						Homogeneous Data (α = 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**

Table 3: 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**

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.

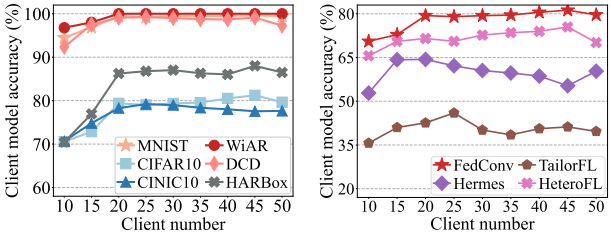
Additionally, Fig. 10(b) shows the client model accuracy of *FedConv* and all baselines with different data heterogeneity. We can see that the performance disparities become more substantial as α decreases, implying that *FedConv* is more robust and can achieve consistently high accuracy across diverse 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 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 α 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 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 [35] (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

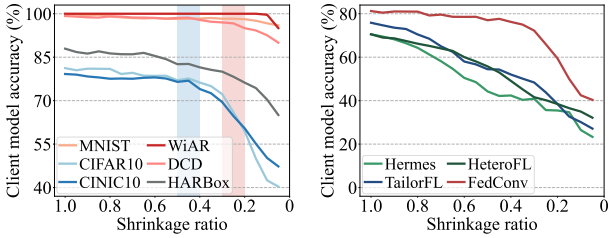
We evaluate the memory footprint, wall-clock time, and communication overhead of each client in *FedConv* and baselines with both homogeneous (α=10000) and heterogeneous (α=0.05) data across clients. Table 2 provides an overview of the average memory usage and the average wall-clock time of each client. 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



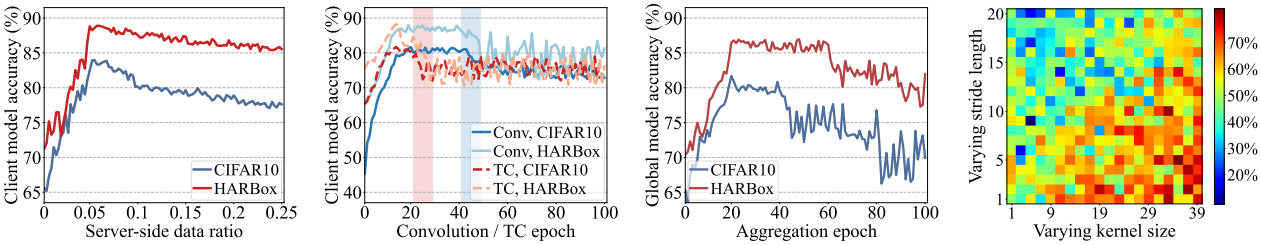
(a) *FedConv* on different datasets (b) *FedConv* vs. baselines (CIFAR10)

Figure 11: Varying number of clients.



(a) *FedConv* on different datasets (b) *FedConv* vs. baselines (CIFAR10)

Figure 12: Varying shrinkage ratios.



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

Figure 13: Sensitivity analysis of varying hyper-parameters in *FedConv*.

clients. Moreover, it holds significant potential that exiting quantization techniques [2, 20] and masking method [41] 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.

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 (α = 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.

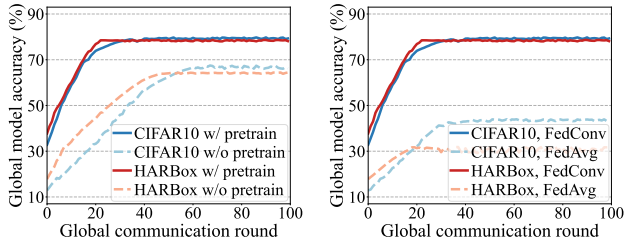
6.4.2 Varying shrinkage ratios. To investigate the trade-off between the SR and model performance, we set the SR for 10 clients as 1.0 and set the SR for the remaining 10 clients as r. We then vary r from 1.0 to 0.05 and record the average client model accuracy (α = 10000). From Fig. 12(a), we can see 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 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. 12(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.

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 [48] or incremental learning [29] techniques can be further applied as more server-side data become available.

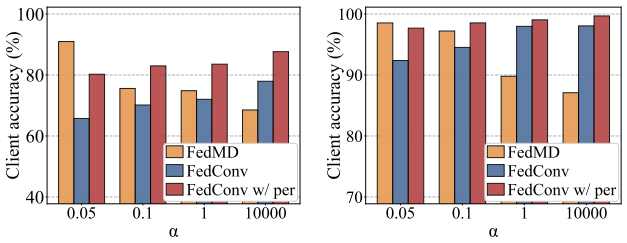
6.4.4 Varying hyper-parameters. We vary the number of epochs for fine-tuning the model compression, dilation, and aggregation to evaluate their impact on the personalization performance of client models. We select two datasets (CIFAR10 and HARBox) for demonstration. We first vary the number of epochs 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



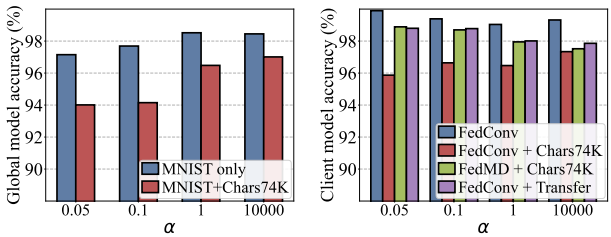
(a) Server-side pre-training (b) Weight vectors

Figure 14: Ablation study of *FedConv*.



(a) CIFAR10 (b) HARBox

Figure 15: *FedConv* with personalized FL.



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

Figure 16: Case study with real-world heterogeneity.

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 [35] 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 [17] 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, 55] 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 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 [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 [52, 54, 62, 70] optimize FL performance under non-IID data. *Clustering-based* methods [11, 33, 52] group clients according to the distribution of their data or model parameters. For example, ClusterFL [52] captures the intrinsic clustering patterns among clients by measuring the similarity of client models. Shu *et al.* [61] propose a clustered multi-task federated learning on non-IID data. *Personalized FL* adopts local fine-tuning [24] or add task-specific layers on client side [35, 78]. For example, pFedMe [22] 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.* [78] 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* [80, 82] generally regard heterogeneous client models as teacher models and learn an aggregated global student model via knowledge distillation (KD). Lin *et al.* [45] leverage KD and ensemble learning to combine the knowledge from heterogeneous client models. FedMD [43] 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* [57] allow sub-models to share a part of the global model parameters to reduce computation overhead. HeteroFL [21] 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 [77]; 3) *Pruning-based methods* [31, 71] have gained popularity in heterogeneous FL. Hermes [39] applies a channel-level pruning method to selectively prune out less important channels. TailorFL [19] 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 [47]. 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 [69]) 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 [38, 63]. 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 [23, 25], which is widely adopted in image dilation [15] and semantic segmentation [51]. 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. 62306313, U21A20462, and 62372400), "Pioneer" and "Leading Goose" R&D Program of Zhejiang under grant No. 2024C03287. Yuanqing Zheng is the corresponding author.

REFERENCES

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. 2015. TensorFlow: Large-scale machine learning on heterogeneous systems.

[2] Ahmed M Abdelmoniem and Marco Canini. 2021. Towards mitigating device heterogeneity in federated learning via adaptive model quantization. In *Proceedings of the 1st Workshop on Machine Learning and Systems*. 96–103.

[3] Andrei Afonin and Sai Praneeth Karimireddy. 2021. Towards model agnostic federated learning using knowledge distillation. *arXiv preprint arXiv:2110.15210* (2021).

[4] Samiul Alam, Luyang Liu, Ming Yan, and Mi Zhang. 2022. Fedrolex: Model-heterogeneous federated learning with rolling sub-model extraction. *Advances in neural information processing systems* 35 (2022), 29677–29690.

[5] Daniel J. Beutel, Taner Topal, Akhil Mathur, Xinchi Qiu, Titouan Parcollet, and Nicholas D. Lane. 2020. Flower: A Friendly Federated Learning Research Framework. *CoRR* (2020).

[6] Daniel J. Beutel, Taner Topal, Akhil Mathur, Xinchi Qiu, Titouan Parcollet, and Nicholas D. Lane. 2023. Flower Android Example (TensorFlowLite). https://github.com/adap/flower/tree/main/examples/android.

[7] Cody Blakeney, Yan Yan, and Ziliang Zong. 2020. Is pruning compression?: Investigating pruning via network layer similarity. In *IEEE CVPR*. 914–922.

[8] Dongqi Cai, Shangguang Wang, Yaozong Wu, Felix Xiaozhu Lin, and Mengwei Xu. 2023. Federated few-shot learning for mobile NLP. In *Proceedings of the 29th Annual International Conference on Mobile Computing and Networking*. 1–17.

[9] Dongqi Cai, Yaozong Wu, Shangguang Wang, Felix Xiaozhu Lin, and Mengwei Xu. 2022. FedAdapter: Efficient Federated Learning for Modern NLP. *arXiv preprint arXiv:2205.10162* (2022).

[10] Dongqi Cai, Yaozong Wu, Shangguang Wang, Felix Xiaozhu Lin, and Mengwei Xu. 2023. Efficient federated learning for modern nlp. In *Proceedings of the 29th Annual International Conference on Mobile Computing and Networking*. 1–16.

[11] Ming Chen, Jinze Wu, Yu Yin, Zhenya Huang, Qi Liu, and Enhong Chen. 2022. Dynamic Clustering Federated Learning for Non-IID Data. In *Artificial Intelligence - Second CAAI International Conference, CICAI*.

[12] Yiqiang Chen, Xin Qin, Jindong Wang, Chaohui Yu, and Wen Gao. 2020. Fedhealth: A federated transfer learning framework for wearable healthcare. *IEEE Intelligent Systems* 35, 4 (2020), 83–93.

[13] Thomas M. Cover and Joy A. Thomas. 2006. *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)*.

[14] Kaiyan Cui, Yanwen Wang, Yuanqing Zheng, and Jinsong Han. 2021. ShakeReader: 'Read' UHF RFID using Smartphone. In *IEEE INFOCOM*.

[15] Ryan Dahl, Mohammad Norouzi, and Jonathon Shlens. 2017. Pixel Recursive Super Resolution. In *IEEE ICCV*. 5449–5458.

[16] Luke N Darlow, Elliot J Crowley, Antreas Antoniou, and Amos J Storkey. 2018. Cinic-10 is not imagenet or cifar-10. *arXiv preprint arXiv:1810.03505* (2018).

[17] Teófilo E de Campos, Bodla Rakesh Babu, and Manik Varma. 2009. Character recognition in natural images. In *International conference on computer vision theory and applications*, Vol. 1. SCITEPRESS, 273–280.

[18] Li Deng. 2012. The MNIST Database of Handwritten Digit Images for Machine Learning Research. *IEEE Signal Process. Mag.* (2012).

[19] Yongheng Deng, Weining Chen, Ju Ren, Feng Lyu, Yang Liu, Yunxin Liu, and Yaoxue Zhang. 2022. TailorFL: Dual-Personalized Federated Learning under System and Data Heterogeneity. In *ACM SenSys*.

[20] Tim Dettmers. 2015. 8-bit approximations for parallelism in deep learning. *arXiv preprint arXiv:1511.04561* (2015).

[21] Enmao Diao, Jie Ding, and Vahid Tarokh. 2021. HeteroFL: Computation and Communication Efficient Federated Learning for Heterogeneous Clients. In *ICLR*. OpenReview.net.

[22] Canh T. Dinh, Nguyen Hoang Tran, and Tuan Dung Nguyen. 2020. Personalized Federated Learning with Moreau Envelopes. In *NeurIPS*.

[23] Vincent Dumoulin and Francesco Visin. 2016. A guide to convolution arithmetic for deep learning. *CoRR* abs/1603.07285 (2016).

[24] Alireza Fallah, Aryan Mokhtari, and Asuman E. Ozdaglar. 2020. Personalized Federated Learning with Theoretical Guarantees: A Model-Agnostic Meta-Learning Approach. In *NeurIPS*.

[25] Hongyang Gao, Hao Yuan, Zhengyang Wang, and Shuiwang Ji. 2020. Pixel Transposed Convolutional Networks. *IEEE Trans. Pattern Anal. Mach. Intell.* (2020).

[26] Jiechao Gao, Mingyue Tang, Tianhao Wang, and Bradford Campbell. 2022. PFed-LDP: A Personalized Federated Local Differential Privacy Framework for IoT Sensing Data. In *ACM SenSys*. 835–836.

[27] Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*.

[28] Linlin Guo, Silu Guo, Lei Wang, Chuang Lin, Jialin Liu, Bingxian Lu, Jian Fang, Zhonghao Liu, Zeyang Shan, and Jingwen Yang. 2019. Wiar: A Public Dataset for Wifi-Based Activity Recognition. *IEEE Access* 7 (2019), 154935–154945.

[29] Haibo He, Sheng Chen, Kang Li, and Xin Xu. 2011. Incremental learning from stream data. *IEEE Transactions on Neural Networks* 22, 12 (2011), 1901–1914.

[30] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *IEEE CVPR*. 770–778.

[31] Samuel Horváth, Stefanos Laskaridis, Mário Almeida, Ilias Leontiadis, Stylianos I. Venieris, and Nicholas D. Lane. 2021. FjORD: Fair and Accurate Federated Learning under heterogeneous targets with Ordered Dropout. In *NeurIPS*.

[32] Tzu-Ming Harry Hsu, Hang Qi, and Matthew Brown. 2019. Measuring the Effects of Non-Identical Data Distribution for Federated Visual Classification. *CoRR* abs/1909.06335 (2019).

[33] Gang Hu, Yinglei Teng, Nan Wang, and F. Richard Yu. 2023. Clustered Data Sharing for Non-IID Federated Learning over Wireless Networks. *CoRR* (2023).

[34] Gao Huang, Yixuan Li, Geoff Pleiss, Zhuang Liu, John E. Hopcroft, and Kilian Q. Weinberger. 2017. Snapshot Ensembles: Train 1, Get M for Free. In *ICLR*. OpenReview.net.

[35] Cihat Keçeci, Mohammad Shaqfeh, Hayat Mbayed, and Erchin Serpedin. 2022. Multi-Task and Transfer Learning for Federated Learning Applications. *CoRR* (2022).

[36] Jakub Konečný, H. Brendan McMahan, Felix X. Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. 2016. Federated Learning: Strategies for Improving Communication Efficiency. *CoRR* abs/1610.05492 (2016).

[37] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. Learning multiple layers of features from tiny images. (2009).

[38] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *NeurIPS*.

[39] Ang Li, Jingwei Sun, Pengcheng Li, Yu Pu, Hai Li, and Yiran Chen. 2021. Hermes: an efficient federated learning framework for heterogeneous mobile clients. In *ACM MobiCom*. 420–437.

[40] Ang Li, Jingwei Sun, Binghui Wang, Lin Duan, Sicheng Li, Yiran Chen, and Hai Li. 2021. LotteryFL: Empower Edge Intelligence with Personalized and Communication-Efficient Federated Learning. In *6th IEEE/ACM Symposium on Edge Computing, SEC*. 68–79.

[41] Ang Li, Jingwei Sun, Xiao Zeng, Mi Zhang, Hai Li, and Yiran Chen. 2021. FedMask: Joint Computation and Communication-Efficient Personalized Federated Learning via Heterogeneous Masking. In *ACM SenSys*.

[42] Chenning Li, Xiao Zeng, Mi Zhang, and Zhichao Cao. 2022. PyramidFL: a fine-grained client selection framework for efficient federated learning. In *ACM MobiCom*. 158–171.

[43] Daliang Li and Junpu Wang. 2019. FedMD: Heterogenous Federated Learning via Model Distillation. *CoRR* (2019).

[44] Jianhua Lin. 1991. Divergence measures based on the Shannon entropy. *IEEE Transactions on Information theory* 37, 1 (1991), 145–151.

[45] Tao Lin, Lingjing Kong, Sebastian U Stich, and Martin Jaggi. 2020. Ensemble distillation for robust model fusion in federated learning. *Advances in Neural Information Processing Systems, NeurIPS* (2020).

[46] Juncai Liu, Jessie Hui Wang, Chenghao Rong, Yuedong Xu, Tao Yu, and Jilong Wang. 2021. FedPA: An adaptively partial model aggregation strategy in Federated Learning. *Comput. Networks* (2021).

[47] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. 2019. Rethinking the Value of Network Pruning. In *ICLR*.

[48] Xinyue Ma, Suyeon Jeong, Minjia Zhang, Di Wang, Jonghyun Choi, and Myeongjae Jeon. 2023. Cost-effective On-device Continual Learning over Memory Hierarchy with Miro. In *Proceedings of the 29th Annual International Conference on Mobile Computing and Networking*. 1–15.

[49] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. 2017. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, AISTATS*.

[50] Alessio Mora, Irene Tenison, Paolo Bellavista, and Irina Rish. 2022. Knowledge distillation for federated learning: a practical guide. *arXiv preprint arXiv:2211.04742* (2022).

[51] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. 2015. Learning Deconvolution Network for Semantic Segmentation. In *IEEE ICCV*. IEEE Computer Society, 1520–1528.

[52] Xiaomin Ouyang, Zhiyuan Xie, Jiayu Zhou, Jianwei Huang, and Guoliang Xing. 2021. ClusterFL: a similarity-aware federated learning system for human activity recognition. In *ACM MobiSys*.

[53] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems, NeurIPS*. 8024–8035.

[54] Boris Radovič and Veljko Pejović. 2023. REPA: Client Clustering without Training and Data Labels for Improved Federated Learning in Non-IID Settings. *arXiv preprint arXiv:2309.14088* (2023).

[55] Sudipan Saha and Tahir Ahmad. 2021. Federated transfer learning: concept and applications. *Intelligenza Artificiale* 15, 1 (2021), 35–44.

[56] Tim Salimans and Durk P Kingma. 2016. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *Advances in neural information processing systems, NeurIPS* 29 (2016).
[57] Leming Shen and Yuanqing Zheng. 2023. FedDM: Data and Model Heterogeneity-Aware Federated Learning via Dynamic Weight Sharing. In *2023 IEEE ICDCS*. IEEE, 975–976.
[58] Guomei Shi, Li Li, Jun Wang, Wenyan Chen, Kejiang Ye, and Chengzhong Xu. 2020. HySync: Hybrid Federated Learning with Effective Synchronization. In *22nd IEEE International Conference on High Performance Computing and Communications*. 628–633.
[59] Jaemin Shin, Yuanchun Li, Yunxin Liu, and Sung-Ju Lee. 2022. FedBalancer: data and pace control for efficient federated learning on heterogeneous clients. In *ACM MobiSys*. 436–449.
[60] Farhad Mortezapour Shiri, Thinagaran Perumal, Norwati Mustapha, and Raihani Mohamed. 2023. A comprehensive overview and comparative analysis on deep learning models: CNN, RNN, LSTM, GRU. *arXiv preprint arXiv:2305.17473* (2023).
[61] Jiangang Shu, Tingting Yang, Xinying Liao, Farong Chen, Yao Xiao, Kan Yang, and Xiaohua Jia. 2023. Clustered Federated Multitask Learning on Non-IID Data With Enhanced Privacy. *IEEE Internet Things J.* (2023).
[62] Xian Shuai, Yulin Shen, Siyang Jiang, Zhihe Zhao, Zhenyu Yan, and Guoliang Xing. 2022. BalanceFL: Addressing class imbalance in long-tail federated learning. In *ACM/IEEE IPSN*. 271–284.
[63] Karen Simonyan and Andrew Zisserman. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *3rd International Conference on Learning Representations, ICLR*.
[64] Jingwei Sun, Ang Li, Lin Duan, Samiul Alam, Xuliang Deng, Xin Guo, Haiming Wang, Maria Gorlatova, Mi Zhang, Hai Li, et al. 2022. FedSEA: A Semi-Asynchronous Federated Learning Framework for Extremely Heterogeneous Devices. In *ACM SenSys*. 106–119.
[65] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. 2017. Axiomatic Attribution for Deep Networks. In *ICML*, Vol. 70. 3319–3328.
[66] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1–9.
[67] Elina Thibeau-Sutre, Sasha Collin, Ninon Burgos, and Olivier Colliot. 2023. Interpretability of Machine Learning Methods Applied to Neuroimaging. *Machine Learning for Brain Disorders* (2023), 655–704.
[68] Mi Tian, Yannwen Wang, Zheng Wang, Junhua Situ, Xiaoqi Sun, Xiaokang Shi, Chenwei Zhang, and Jiaxing Shen. 2023. RemoteGesture: Room-scale Acoustic Gesture Recognition for Multiple Users. In *IEEE SECON*. 231–239.
[69] Nicola Tonellotto, Alberto Gotta, Franco Maria Nardini, Daniele Gadler, and Fabrizio Silvestri. 2021. Neural network quantization in federated learning at the edge. *Information Sciences* 575 (2021), 417–436.
[70] Linlin Tu, Xiaomin Ouyang, Jiayu Zhou, Yuze He, and Guoliang Xing. 2021. FedDL: Federated Learning via Dynamic Layer Sharing for Human Activity Recognition. In *ACM SenSys*. 15–28.
[71] Saeed Vahidian, Mahdi Morafah, and Bill Lin. 2021. Personalized federated learning by structured and unstructured pruning under data heterogeneity. In *2021 IEEE 41st international conference on distributed computing systems workshops (ICDCSW)*. IEEE, 27–34.
[72] Yanwen Wang, Jiaxing Shen, and Yuanqing Zheng. 2020. Push the Limit of Acoustic Gesture Recognition. In *IEEE INFOCOM*.
[73] Wikipedia contributors. 2023. Cross-entropy — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=Cross-entropy&oldid=1170369413
[74] Rui Xiao, Jianwei Liu, Jinsong Han, and Kui Ren. 2021. Onefi: One-shot recognition for unseen gesture via cots wifi. In *ACM SenSys*.
[75] Huatao Xu, Pengfei Zhou, Rui Tan, and Mo Li. 2023. Practically Adopting Human Activity Recognition. In *Proceedings of the 29th Annual International Conference on Mobile Computing and Networking*. 1–15.
[76] Qiang Yang and Yuanqing Zheng. 2023. AquaHelper: Underwater SOS Transmission and Detection in Swimming Pools. In *ACM SenSys*.
[77] Dezhong Yao, Wanning Pan, Yao Wan, Hai Jin, and Lichao Sun. 2021. FedHM: Efficient Federated Learning for Heterogeneous Models via Low-rank Factorization. *CoRR* abs/2111.14655 (2021).
[78] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. 2014. How transferable are features in deep neural networks?. In *Advances in Neural Information Processing Systems 27 2014, NeurIPS*.
[79] Matthew D. Zeiler and Rob Fergus. 2014. Visualizing and Understanding Convolutional Networks. In *IEEE ECCV*.
[80] Jie Zhang, Song Guo, Xiaosong Ma, Haozhao Wang, Wenchao Xu, and Feijie Wu. 2021. Parameterized Knowledge Transfer for Personalized Federated Learning. In *NeurIPS*. 10092–10104.
[81] Li Lyna Zhang, Shihao Han, Jianyu Wei, Ningxin Zheng, Ting Cao, Yuqing Yang, and Yunxin Liu. 2021. nn-Meter: towards accurate latency prediction of deep-learning model inference on diverse edge devices. In *ACM MobiSys*. ACM, 81–93.
[82] Zhuangdi Zhu, Junyuan Hong, and Jiayu Zhou. 2021. Data-Free Knowledge Distillation for Heterogeneous Federated Learning. In *ICML*. PMLR.