Towards Personalized Federated Learning via Heterogeneous Model Reassembly (Supplementary Materials)

Anonymous Author(s)
Affiliation
Address
email

In this supplementary materials file, we provide additional content for the main manuscript due to the page limit and requirements.

1 Pseudo-code of pFedHR

Algorithm 1 shows the pseudo-code of the proposed pFedHR model, which contains two main updates: the server update (lines 3-11) and the client update (lines 12 - 17). The details of reassembly candidate generation can be found in the main manuscript (Algorithm 1 on Page 5).

Algorithm 1: Algorithm Flow of pFedHR.

Input: Local data $\mathcal{D}_1, \mathcal{D}_2, .., \mathcal{D}_N$, the number of active clients B, communication rounds T, the number of local training epochs E_c, the number of fine-tuning epochs at the server E_s, cluster number K.

1 **for** each communication round $t = 1, 2, \cdots, T$ **do**
2 Randomly sample B active clients and their model parameters are denoted as $\{\mathbf{w}_t^1, \mathbf{w}_t^2, \cdots, \mathbf{w}_t^B\}$;
3 **Server Update**
4 **for** each $n \in [1, \cdots, B]$ **do**
5 Conduct *layer-wise ecomposition* on $\mathbf{w}_t^n = [(\mathbf{L}_{t,1}^n, O_1^n), \cdots, (\mathbf{L}_{t,H}^b, O_E^n)]$;
6 **end**
7 Conduct *function-driven layer grouping* with Eq. (4) to obtain $\{\mathcal{G}_t^1, \mathcal{G}_t^2, \cdots, \mathcal{G}_t^K\}$;
8 Conduct *reassembly candidate generation* to obtain $\{\mathbf{c}_t^1, \cdots, \mathbf{c}_t^M\}$ following $\mathcal{R}_1, \mathcal{R}_2, \mathcal{R}_3$, and $\mathcal{R}_4$;
9 Conduct *layer stitching* on the generated candidates;
10 Conduct *similarity calculation* on pairs of models with Eq. (5);
11 Distribute the personalized models back to the clients accordingly.
12 **Client Update**
13 **for** each $n \in [1, \cdots, B]$ **do**
14 **for** each local epoch e from 1 to E_c **do**
15 Update $\mathbf{w}_t^n$ with Eq. (6);
16 **end**
17 **end**
18 Upload B models $\{\mathbf{w}_t^1, \mathbf{w}_t^2, \cdots, \mathbf{w}_t^B\}$ back to the server;
19 **end**

Submitted to 37th Conference on Neural Information Processing Systems (NeurIPS 2023). Do not distribute.

2 Implementation Details

The proposed pFedHR is implemented in Pytorch 2.0.1 and runs on NVIDIA A100 with CUDA version 12.0 on a Ubuntu 20.04.6 LTS server. The hyperparameter λ in Eq. (6) is 0.2. We use Adam as the optimizer. The learning rate of the local client learning and the server fine-tuning learning rate are both equal to 0.001.

2.1 Model Details

In our experiments, we have 4 CNN models with different complexity. The details are shown as follows. In each convolutional NN sequential block, there are 1 convolutional layer, a max pooling layer, and a ReLU function.

M1: *Cov1:{Conv2d (kernel size = 5)* $\rightarrow$ *ReLU* $\rightarrow$ *MaxPool2D (kernel size = 2, stride = 2) }* $\rightarrow$ *Cov2:{Conv2d (kernel size = 5)* $\rightarrow$ *ReLU* $\rightarrow$ *MaxPool2D (kernel size = 2, stride = 2) }* $\rightarrow$ *FC1:{Linear* $\rightarrow$ *ReLU}* $\rightarrow$ *Dropout*$\rightarrow$ *FC2:Linear*.

M2: *Cov1:{Conv2d (kernel size = 5)* $\rightarrow$ *ReLU* $\rightarrow$ *MaxPool2D (kernel size = 2, stride = 2) }* $\rightarrow$ *Cov2:{Conv2d (kernel size = 5)* $\rightarrow$ *ReLU* $\rightarrow$ *MaxPool2D (kernel size = 2, stride = 2) }* $\rightarrow$ *Cov3:{Conv2d (kernel size = 5)* $\rightarrow$ *ReLU}* $\rightarrow$ *FC1:{Linear* $\rightarrow$ *ReLU}* $\rightarrow$ *Dropout*$\rightarrow$ *FC2:Linear*.

M3: *Cov1:{Conv2d (kernel size = 5)* $\rightarrow$ *ReLU* $\rightarrow$ *MaxPool2D (kernel size = 2, stride = 2) }* $\rightarrow$ *Cov2:{Conv2d (kernel size = 5)* $\rightarrow$ *ReLU* $\rightarrow$ *MaxPool2D (kernel size = 2, stride = 2) }* $\rightarrow$ *Cov3:{Conv2d (kernel size = 5)* $\rightarrow$ *ReLU}* $\rightarrow$ *Cov4:{Conv2d (kernel size = 5)* $\rightarrow$ *ReLU* $\rightarrow$ *MaxPool2D (kernel size = 2, stride = 2) }* $\rightarrow$ *Cov4:{Conv2d (kernel size = 5)* $\rightarrow$ *ReLU}* $\rightarrow$ *FC1:{Linear* $\rightarrow$ *ReLU* $\rightarrow$ *Dropout}* $\rightarrow$ *FC2:{Linear* $\rightarrow$ *ReLU}*$\rightarrow$ *FC3:{Linear* $\rightarrow$ *ReLU}*$\rightarrow$*FC4:Linear*.

M4: *Cov1:{Conv2d (kernel size = 5)* $\rightarrow$ *BatchNorm2d* $\rightarrow$ *ReLU}* $\rightarrow$ *Cov2:{Conv2d (kernel size = 3)* $\rightarrow$ *ReLU* $\rightarrow$ *MaxPool2D (kernel size = 2, stride = 2) }* $\rightarrow$ *Cov3:{Conv2d (kernel size = 3)* $\rightarrow$ *BatchNorm2d* $\rightarrow$ *ReLU}* $\rightarrow$ *Cov4:{Conv2d (kernel size = 5)* $\rightarrow$ *ReLU* $\rightarrow$ *MaxPool2D (kernel size = 2, stride = 2)* $\rightarrow$ Dropout } $\rightarrow$ *Cov5:{Conv2d (kernel size = 3)* $\rightarrow$ *BatchNorm2d* $\rightarrow$ *ReLU }* $\rightarrow$ *Cov6:{Conv2d (kernel size = 3)* $\rightarrow$ *ReLU* $\rightarrow$ *MaxPool2D (kernel size = 2, stride = 2) }* $\rightarrow$ *FC1:{Linear* $\rightarrow$ *ReLU* $\rightarrow$ *Dropout}* $\rightarrow$ *FC2:{Linear* $\rightarrow$ *ReLU}*$\rightarrow$ *FC3:{Linear* $\rightarrow$ *ReLU}*$\rightarrow$*FC4:Linear*.

2.2 Details of Rule $\mathcal{R}_2$ (Operation Order)

We introduce the rule $\mathcal{R}_2$ (operation order) in Line 11 of the algorithm in the main manuscript. We follow existing work [1–5] to define the following rules:

1. A CNN typically has convolutional layers, pooling layers, and fully connected layers, which requires the generated candidate to have these functional layers.
2. The typical operation order is convolution layers $\rightarrow$ ReLU layers $\rightarrow$ pooling layers $\rightarrow$ fully connected layers.

2.3 Public Data Sensitivity Analysis

2.3.1 Layer Stitching Study

One of our major contributions is to develop a new layer stitching strategy to reduce the adverse impacts of introducing public data, even with different distributions from the client data. Our proposed strategy includes two aspects: (1) using a simple layer to stitch layers and (2) reducing the number of finetuning epochs. To validate the correctness of these assumptions, we conduct the following experiments on SVHN with 12 clients, where both client data and **labeled** public data are extracted from SVHN.

Stitching Layer Numbers. In this experiment, we add the complexity of layers for stitching. In our model design, we only use $\text{ReLU}(\mathbf{W}^\top \mathbf{X} + \mathbf{b})$. Now, we increase the number of linear layers from 1 to 2 to 3. The results are depicted in Figure 1. We can observe that under both IID and Non-IID settings, the performance will decrease with the increase of the complexity of the stitching layers. These results demonstrate our assumption that more complex stitching layers will introduce more information about public data but reduce the personalized information of each client model maintained. Thus, using a simple layer to stitch layers is a reasonable choice.

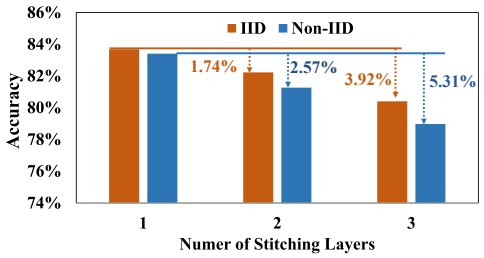

Figure 1: Stitching layer number study.

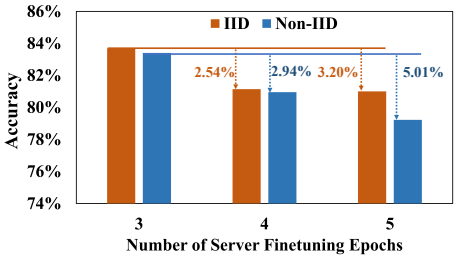

Figure 2: Server finetuning number study.

Stitched Model Finetuning Numbers. We further explore the influence of the number of finetuning epochs on the stitched model. The results are shown in Figure 2. We can observe that increasing the number of finetuning epochs can also introduce more public data information and reduce model performance. Thus, setting a small number of finetuning epochs benefits keeping the model's performance.

2.3.2 Experiment results with Different Numbers of Clusters

In our model design, we need to group functional layers into K groups by optimizing Eq. (4). Where K is a predefined hyperparameter. In this experiment, we aim to investigate the performance influence with regard to K. In particular, we conduct the experiments on the SVHN dataset with 12 local clients, and the public data are also the SVHN data.

Figure 3 shows the results on both IID and Non-IID settings with labeled and unlabeled public data. X-axis represents the number of clusters, and Y-axis denotes the accuracy values. We can observe that with the increase of K, the performance will also increase. However, in the experiments, we do not recommend setting a large K since a trade-off balance exists between K and M, where M is the number of candidates automatically generated by Algorithm 1 in the main manuscript. If K is large, then M will be small due to Rule $\mathcal{R}_4$. In other words, a larger K may make the empty $\mathcal{C}_t$ returned by Algorithm 1 in the main manuscript.

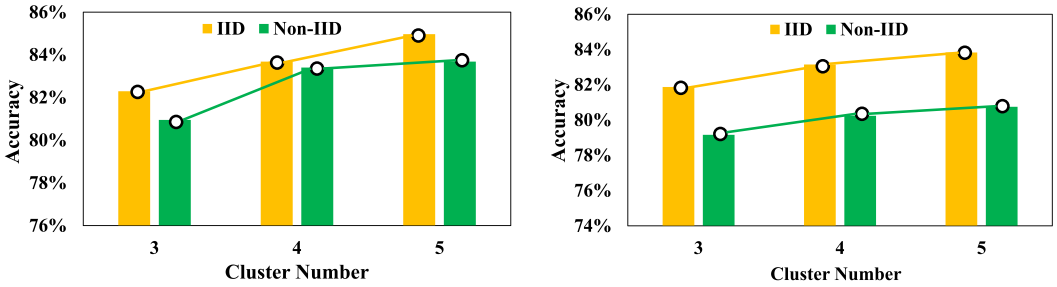

Figure 3: Results on different number of clusters K's.

2.3.3 Experiment Results of Public Data Sensitivity on SVHN with 100 Clients

Figure 4 shows the experimental results with 100 clients. The clients hold the SVHN data, and we alternatively change the (labeled and unlabeled) public data and report the results for both IID and Non-IID settings. We can observe the same results as those on the 12 clients (Figure 1 in the main manuscript). All approaches can achieve the best performance when using SVNH as the public data. Although using other public data, the performance of all approaches drops, the performance change of the proposed `pFedHR` is lowest. Besides, even using other public data, `pFedHR` can achieve comparable or better performance to baselines. For example, in Figure 4(b), when `pFedHR` uses the MNIST as the public data, its performance is comparable to FedMD and better than FedGH using SVHN as the public data. These results confirm that the proposed `pFedHR` can handle the public data sensitivity issue.

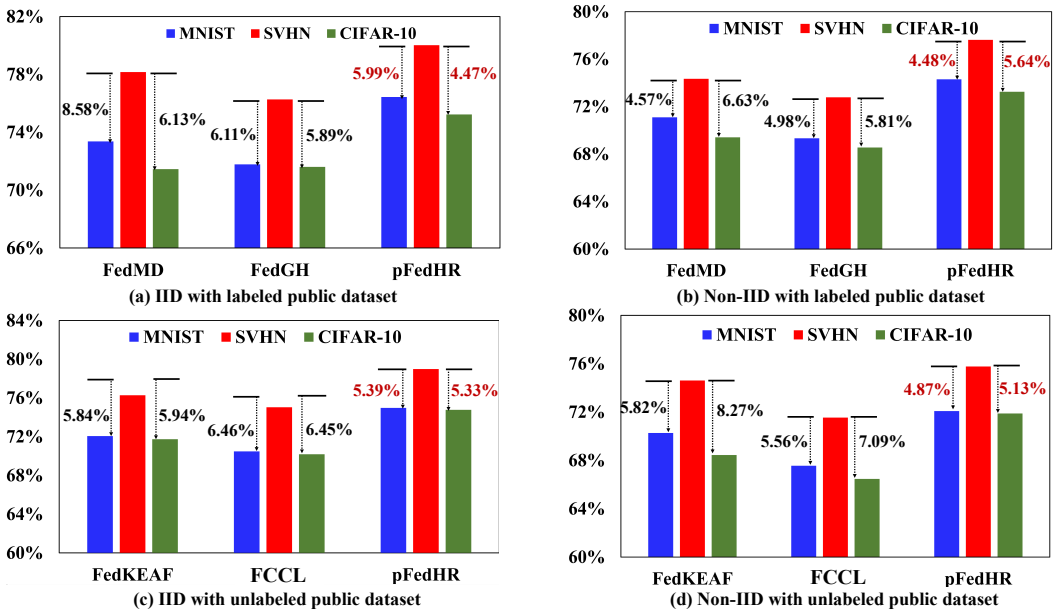

Figure 4: Results with 100 clients.

2.3.4 Experiment Results of Public Data Sensitivity on CIFAR-10 with 12 Clients

We also validate the public data sensitivity on CIFAR-10, where clients hold data from CIFAR-10, and we alternatively change the public data. The results are shown in Figures 5 and 6. We have the same observations as we discussed before. Using public data with different distributions makes the performance drop, but the proposed `pFedHR` has the smallest drop ratio.

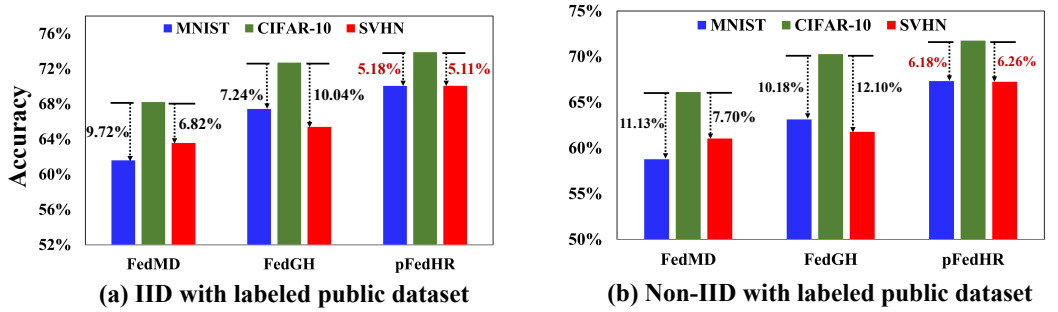

Figure 5: Results with CIFAR-10 private dataset and labeled public datasets.

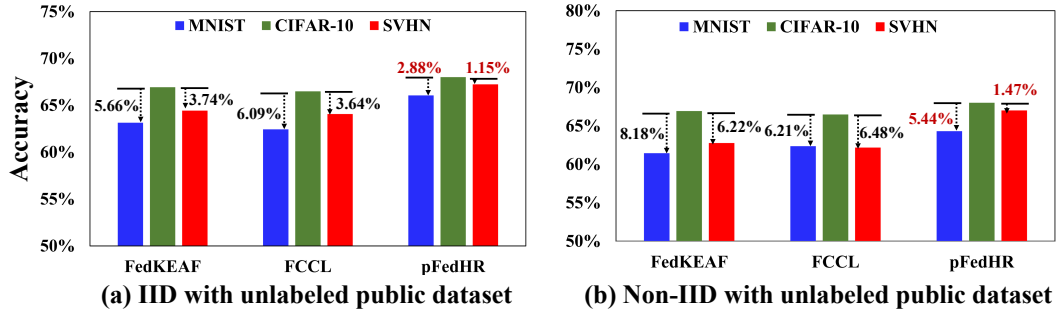

Figure 6: Results with CIFAR-10 private dataset and unlabeled public datasets.

3 Limitation

All the experimental results demonstrate the effectiveness of the proposed pFedHR. However, pFedHR still have the following limitations:

1. In our experiments, we use four different CNN-based models and randomly send them to clients. The reason for using these simple models is that they can save computational resources. In our model design, we will calculate the CKA score between any pair of layers. In other words, if the local model is very deep (with multiple layers), the computational complexity will be very high.
2. In real-world applications, the client models may be more complicated. To deploy the proposed pFedHR in a real environment, we need to slightly modify the layer-wise decomposition and use block-wise decomposition as [6] and redefine the rules used for generating candidates accordingly.
3. In our experiments, we only test the designed model on the image datasets. In real-world applications, multiple types of data may be stored in each client. How to handle other types of data with heterogeneous model reassembly is one of our major future works.

4 Broader Impacts

The proposed pFedHR addresses the practical challenge of model heterogeneity in federated learning. By introducing heterogeneous model reassembly and personalized federated learning, this research contributes to the advancement of federated learning techniques, potentially improving the efficiency and effectiveness of collaborative machine learning in distributed systems.

The pFedHR framework automatically generates informative and diverse personalized candidates with minimal human intervention. This has the potential to reduce the burden on human experts and practitioners, making the process of model personalization more efficient and scalable. It opens up possibilities for deploying federated learning systems in real-world scenarios where manual customization is impractical or time-consuming.

The proposed heterogeneous model reassembly technique in pFedHR try to mitigate the adverse impact caused by using public data with different distributions from the client data. This can be beneficial in scenarios where privacy concerns limit the availability of extensive client data, such as healthcare, enabling the utilization of publicly available data while maintaining a certain level of model performance. It promotes the ethical and responsible use of data and encourages collaboration between organizations and researchers in a privacy-preserving manner.

References

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

[2] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[4] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[5] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[6] Xingyi Yang, Daquan Zhou, Songhua Liu, Jingwen Ye, and Xinchao Wang. Deep model reassembly. *Advances in neural information processing systems*, 35:25739–25753, 2022.