FedLEGO: Enabling Heterogenous Model Cooperation via Brick Reassembly in Federated Learning

ABSTRACT

This paper focuses on addressing the practical yet challenging problem of model heterogeneity in federated learning, where clients possess models with different network structures. To track this problem, we propose a novel framework called `FedLEGO`, which treats each client model as a LEGO toy, reassembles it into bricks, and assembles bricks back into personalized models accordingly. Moreover, `FedLEGO` automatically and dynamically generates informative and diverse personalized candidates with minimal human intervention. Furthermore, our proposed heterogeneous model reassembly technique mitigates the adverse impact introduced by using public data with different distributions from the client data to a certain extent. Experimental results demonstrate that `FedLEGO` outperforms baselines on three datasets under both IID and Non-IID settings. Additionally, `FedLEGO` effectively reduces the adverse impact of using different public data and dynamically generates diverse personalized models in an automated manner.

KEYWORDS

federated learning, model personalization, model heterogeneity

ACM Reference Format:

. 2018. FedLEGO: Enabling Heterogenous Model Cooperation via Brick Reassembly in Federated Learning. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX).* ACM, New York, NY, USA, 7 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Federated learning (FL) aims to enable collaborative machine learning without the need to share clients' data with others, thereby upholding data privacy [6]. However, traditional federated learning approaches [3, 4, 9, 12, 15, 18–20, 23, 28] typically enforce the use of an identical model structure for all clients during training. This constraint poses challenges in achieving personalized learning within the FL framework. In real-world scenarios, clients such as data centers, institutes, or companies often possess their own distinct models, which may have varying structures. Training on top of their original models should be a better solution than deploying new ones for collaborative purposes. Therefore, a practical solution lies in fostering **heterogeneous model cooperation** within FL, while preserving individual model structures. Only a few studies have attempted to address the challenging problem of heterogeneous model cooperation in FL [5, 8, 10, 22, 26], and most of them incorporate the use of a public dataset to facilitate both cooperation and personalization [5, 8, 10, 26].

However, these approaches still face several key issues: **(1) Undermining personalization through consensus**: Existing methods often generate consensual side information, such as class information [8], logits [5, 17], and label-wise representations [25], using public data. This information is then exchanged and used to conduct average operations on the server, resulting in a consensus representation. However, this approach poses privacy and security concerns due to the exchange of side information [11]. Furthermore, the averaging process significantly diminishes the unique characteristics of individual local models, thereby hampering model personalization. **(2) Excessive reliance on prior knowledge for distillation-based approaches**: Distillation-based techniques, such as knowledge distillation (KD), are commonly employed for heterogeneous model aggregation in FL [10, 21, 26]. However, these techniques necessitate the predefinition of a shared model structure based on prior knowledge [26]. Consequently, handcrafted models can heavily influence local model personalization. Additionally, a fixed shared model structure may be insufficient for effectively guiding personalized learning when dealing with a large number of clients with non-IID data. **(3) Sensitivity to the choice of public datasets**: Most existing approaches use public data to obtain guidance information, such as logits [5, 17] or a shared model [26], for local model personalization. The design of these approaches makes public data and model personalization **tightly bound together**. Thus, they usually choose the public data with the same distribution as the client data. Therefore, using public data with different distributions from client data will cause a significant performance drop in existing models.

We present a novel framework called `FedLEGO`, which aims to achieve personalized federated learning and address the issue of heterogeneous model cooperation (as depicted in Figure 1). The `FedLEGO` framework comprises two key updates: the server update and the client update. In particular, we approach the issue of heterogeneous model personalization from a model-matching optimization perspective on the **server** side. To solve this problem, we introduce a novel **heterogeneous model reassembly** technique to assemble models uploaded from clients, i.e., $\{\mathbf{w}_t^1, \cdots, \mathbf{w}_t^B\}$, where B is the number of active clients in the t-the communication round. We treat each local model as a LEGO toy and disassembly them into bricks by layers. After that, we reassemble the bricks into new toys to obtain models and we conduct similarly-based matching to distribute models back to the clients accordingly. At the client side, we treat the matched personalized model as a guidance mechanism for client parameter learning using knowledge distillation[1].

Our work makes the following key contributions: (1) We introduce the first personalized federated learning framework based on model reassembly, specifically designed to address the challenges

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

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY

© 2018 Association for Computing Machinery.

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

[1]Note that the network architecture of both local model and personalized model are known, and thus, there is no human intervention in the client update.

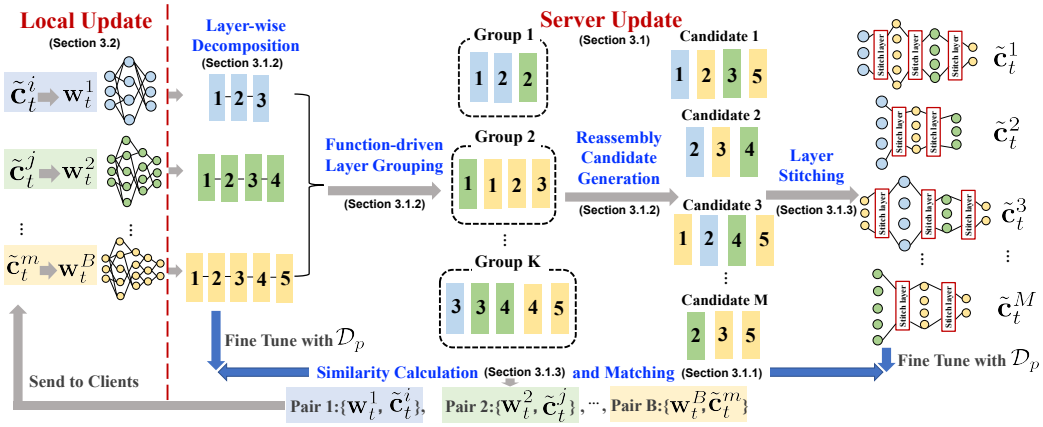

Figure 1: Overview of the proposed `FedLEGO`. K is the number of clusters.

of heterogeneous model cooperation. (2) The proposed `FedLEGO` framework demonstrates the ability to automatically and dynamically generate personalized candidates that are both informative and diverse, requiring minimal human intervention. (3) We present a novel heterogeneous model reassembly technique, which effectively mitigates the adverse impact caused by using public data with distributions different from client data. (4) Experimental results show that the `FedLEGO` framework achieves state-of-the-art performance on three datasets, exhibiting superior performance under both IID and Non-IID settings when compared to baselines employing labeled and unlabeled public datasets.

2 RELATED WORKS

Model Heterogeneity in Federated Learning. Most existing approaches **leverage public data** to facilitate model training. Among them, FedDF [10], FedKEMF [26], and FCCL [5] employ *unlabeled public data*. However, FedDF trains a global model with different settings compared to our approach. FedKEMF performs mutual knowledge distillation learning on the server side to achieve model personalization, requiring predefined model structures. FCCL averages the logits provided by each client and utilizes a consensus logit as guidance during local model training. It is worth noting that the use of logits raises concerns regarding privacy and security [11, 16]. FedMD [8] and FedGH [25] employ *labeled public data*. These approaches exchange class information or representations between the server and clients and perform aggregation to address the model heterogeneity issue. However, similar to FCCL, these methods also introduce privacy leakage concerns.

Neural Network Reassembly and Stitching. Conventional methods are primarily employed in existing federated learning approaches to obtain personalized client models, with limited exploration of model reassembly. Additionally, there is a lack of research investigating neural network reassembly and stitching [1, 2, 13, 14, 24] within the context of federated learning. For instance, the work presented in [1] proposes three algorithms to merge two models within the weight space, but it is limited to handling only two models as input. In our setting, multiple models need to be incorporated into the model reassembly or aggregation process. Furthermore, both [24] and [14] focus on pre-trained models, which differ from our specific scenario.

3 METHODOLOGY

Our model `FedLEGO` incorporates two key updates: the server update and the local update, as depicted in Figure 1. Next, we provide the details of our model design starting with the server update.

3.1 Server Update

At communication round t, the server will receive B heterogeneous client models with parameters denoted as $\{\mathbf{w}_t^1, \mathbf{w}_t^2, \cdots, \mathbf{w}_t^B\}$. As we discussed in Section 1, traditional approaches have limitations when applied in this context. To overcome these limitations and learn a personalized model $\hat{\mathbf{w}}_t^n$ that can be distributed to the corresponding n-th client, we propose a novel approach that leverages the publicly available data $\mathcal{D}_p$ stored on the server to find the **most similar** aggregated models learned from $\{\mathbf{w}_t^1, \mathbf{w}_t^2, \cdots, \mathbf{w}_t^B\}$ for $\mathbf{w}_t^n$.

3.1.1 Similarity-based Model Matching. Let $g(\cdot,\cdot)$ denote the model aggregation function, which can automatically and dynamically obtain M aggregated model candidates as follows:

$$\{\mathbf{c}_t^1, \cdots, \mathbf{c}_t^M\} = g(\{\mathbf{w}_t^1, \mathbf{w}_t^2, \cdots, \mathbf{w}_t^B\}, \mathcal{D}_p), \tag{1}$$

where $g(\cdot,\cdot)$ will be detailed in Section 3.1.2, and $\mathbf{c}_k^m$ is the m-th generated model candidate learned by $g(\cdot,\cdot)$. Note that $\mathbf{c}_k^m$ denotes the model before network stitching. M is the total number of candidates, which is not a fixed number and is estimated by $g(\cdot,\cdot)$. In such a way, our goal is to optimize the following function:

$$\mathbf{c}_t^* = \underset{\mathbf{c}_t^m; m\in[1,M]}{\arg\max} \ \{\mathrm{sim}(\mathbf{w}_t^n, \mathbf{c}_t^1; \mathcal{D}_p), \cdots, \mathrm{sim}(\mathbf{w}_t^n, \mathbf{c}_t^M; \mathcal{D}_p)\}, \tag{2}$$

where $n \in [1, B]$, $\mathbf{c}_t^*$ is the best matched model for $\mathbf{w}_t^n$, which is also denoted as $\hat{\mathbf{w}}_t^n = \mathbf{c}_t^*$. $\mathrm{sim}(\cdot,\cdot)$ is the similarity function between two models, which will be detailed in Section 3.1.3.

3.1.2 Heterogeneous Model Reassembly – $g(\cdot,\cdot)$. To optimize Eq. (2), we need to obtain M candidates using the heterogenous model aggregation function $g(\cdot)$ in Eq. (1). To avoid the issue of predefined model architectures in the knowledge distillation approaches, we aim to automatically and dynamically learn the candidate architectures via a newly designed function $g(\cdot,\cdot)$. In particular, we propose a decomposition-grouping-reassembly method as $g(\cdot,\cdot)$, including layer-wise decomposition, function-driven layer grouping, and reassembly candidate generation.

Table 1: Performance comparison with baselines under the heterogeneous setting.

Public Data	Dataset	MNIST		SVHN		CIFAR-10	
	Approach	IID	Non-IID	IID	Non-IID	IID	Non-IID
Labeled	FedMD [8]	93.08%	91.44%	81.55%	78.39%	68.22%	66.13%
	FedGH [25]	94.10%	93.27%	81.94%	81.06%	72.69%	70.27%
	FedLEGO	**94.55%**	**94.41%**	**83.68%**	**83.40%**	**73.88%**	**71.74%**
Unlabeled	FedKEMF [26]	93.01%	91.66%	80.41%	79.33%	67.12%	66.93%
	FCCL [5]	93.62%	92.88%	82.03%	79.75%	68.77%	66.49%
	FedLEGO	**93.89%**	**93.76%**	**83.15%**	**80.24%**	**69.38%**	**68.01%**

Layer-wise Decomposition. Assume that each uploaded client model $\mathbf{w}_t^n$ contains H layers, i.e., $\mathbf{w}_t^n = [(\mathbf{L}_{t,1}^n, O_1^n), \cdots, (\mathbf{L}_{t,H}^n, O_E^n)]$, where each layer $\mathbf{L}_{t,h}^n$ is associated with an operation type O_e^n. For example, a plain convolutional neural network (CNN) usually has three operations: convolution, pooling, and fully connected layers. For different client models, H may be different. The decomposition step aims to obtain these layers and their corresponding operation types.

Function-driven Layer Grouping. After decomposing layers of client models, we group these layers based on their functional similarities. Due to the model structure heterogeneity in our setting, the dimension size of the output representations from layers by feeding the public data $\mathcal{D}_p$ to different models will be different. Thus, measuring the similarity between a pair of layers is challenging, which can be resolved by applying the commonly used centered kernel alignment (CKA) technique [7]. In particular, we define the distance metric between any pair of layers as follows:

$$\text{dis}(\mathbf{L}_{t,i}^n, \mathbf{L}_{t,j}^b) = (\text{CKA}(\mathbf{X}_{t,i}^n, \mathbf{X}_{t,i}^b) + \text{CKA}(\mathbf{L}_{t,i}^n(\mathbf{X}_{t,i}^n), \mathbf{L}_{t,i}^b(\mathbf{X}_{t,i}^b)))^{-1}, \tag{3}$$

where $\mathbf{X}_{t,i}^n$ is the input data of $\mathbf{L}_{t,i}^n$, and $\mathbf{L}_{t,i}^n(\mathbf{X}_{t,i}^n)$ denotes the output data from $\mathbf{L}_{t,i}^n$. This metric uses $\text{CKA}(\cdot,\cdot)$ to calculate the similarity between both input and output data of two layers.

Based on the defined distance metric, we conduct the K-means-style algorithm to group the layers of B models into K clusters. This optimization process aims to minimize the sum of distances between all pairs of layers, denoted as $\mathcal{L}_t$. The procedure can be described as follows:

$$\min \mathcal{L}_t = \min_{\delta_{b,h}^a \in \{0,1\}} \sum_{k=1}^{K} \sum_{b=1}^{B} \sum_{h=1}^{H} \delta_{b,h}^k (\text{dis}(\mathbf{L}_t^k, \mathbf{L}_{t,h}^b)), \tag{4}$$

where $\mathbf{L}_t^k$ is the center of the k-th cluster. $\delta_{b,h}^k$ is the indicator. If the h-th layer of $\mathbf{w}_t^b$ belongs to the k-th cluster, then $\delta_{b,h}^k = 1$. Otherwise, $\delta_{b,h}^k = 0$. After the grouping process, we obtain K layer clusters denoted as $\{\mathcal{G}_t^1, \mathcal{G}_t^2, \cdots, \mathcal{G}_t^K\}$. There are multiple layers in each group, which have similar functions. Besides, each layer is associated with an operation type.

Reassembly Candidate Generation. The last step for obtaining personalized candidates $\{\mathbf{c}_t^1, \cdots, \mathbf{c}_t^M\}$ is to assemble the learned layer-wise groups $\{\mathcal{G}_t^1, \mathcal{G}_t^2, \cdots, \mathcal{G}_t^K\}$ based on their functions. Our goal is to automatically generate *informative* and *diverse* candidates.

Generally, an **informative** candidate needs to follow the design of handcrafted network structures. This is challenging since the candidates are automatically generated without human interventions and prior knowledge. To satisfy this condition, we require the layer orders to be guaranteed following $\mathcal{R}_1$. For example, the i-th layer from the n-th model, i.e., $\mathbf{L}_{t,i}^n$, in a candidate must be followed by a layer with an index $j > i$ from other models or itself. Besides, the operation type also determines the quality of a model. For a CNN model, the fully connected layer is usually used after the convolution layer, which motivates us to design the $\mathcal{R}_2$ operation order rule. To avoid computational issues and further obtain high-quality candidates, we use the **diversity** principle as the filtering rule. A diverse and informative model should contain all the operation types, i.e., the $\mathcal{R}_3$ complete operation rule in Line 16. Besides, the groups $\{\mathcal{G}_t^1, \cdots, \mathcal{G}_t^K\}$ are clustered based on their layer functions. The requirement that layers of candidates must be from different groups should significantly increase the diversity of model functions, which motivates us to design the $\mathcal{R}_4$ diverse group rule.

3.1.3 Similarity Learning with Layer Stitching – $sim(\cdot,\cdot)$.

After obtaining a set of candidate models $\{\mathbf{c}_t^1, \cdots, \mathbf{c}_t^M\}$, to optimize Eq. (2), we need to calculate the similary bettwen each client model $\mathbf{w}_t^n$ and all the cadidates $\{\mathbf{c}_t^1, \cdots, \mathbf{c}_t^M\}$ using the public data $\mathcal{D}_p$. However, this is non-trivial since $\mathbf{c}_t^m$ is assembled by layers from different client models, which is not a complete model architecture. We have to stitch these layers together before using $\mathbf{c}_t^m$.

Layer Stitching. Assume that $\mathbf{L}_{t,i}^n$ and $\mathbf{L}_{t,j}^b$ are any two consecutive layers in the candidate model $\mathbf{c}_t^m$. Let d_i denote the output dimension of $\mathbf{L}_{t,i}^n$ and d_j denote the input dimension of $\mathbf{L}_{t,j}^b$. d_i is usually not equal to d_j. To stitch these two layers, we follow existing work [14] by adding a nonlinear activation function $\text{ReLU}(\cdot)$ on top of a linear layer, i.e., $\text{ReLU}(\mathbf{W}^\top \mathbf{X} + \mathbf{b})$, where $\mathbf{W} \in \mathbb{R}^{d_i \times d_j}$, $\mathbf{b} \in \mathbb{R}^{d_j}$, and $\mathbf{X}$ represents the output data from the first layer. In such a way, we can obtain a stitched candidate $\tilde{\mathbf{c}}_t^m$.

Similarity Calculation. We propose to use the cosine score $\cos(\cdot,\cdot)$ to calculate the similarity between a pair of models $(\mathbf{w}_t^n, \tilde{\mathbf{c}}_t^m)$ as follows:

$$\text{sim}(\mathbf{w}_t^n, \mathbf{c}_t^m; \mathcal{D}_p) = \text{sim}(\mathbf{w}_t^n, \tilde{\mathbf{c}}_t^m; \mathcal{D}_p) = \frac{1}{P} \sum_{p=1}^{P} \cos(\boldsymbol{\alpha}_t^n(\mathbf{x}_p), \boldsymbol{\alpha}_t^m(\mathbf{x}_p)), \tag{5}$$

where P denotes the number of data in the public dataset $\mathcal{D}_p$ and $\mathbf{x}_p$ is the p-th data in $\mathcal{D}_p$. $\boldsymbol{\alpha}_t^n(\mathbf{x}_p)$ and $\boldsymbol{\alpha}_t^m(\mathbf{x}_p)$ are the logits output from models $\mathbf{w}_t^n$ and $\tilde{\mathbf{c}}_t^m$, respectively. To obtain the logits, we need to finetune $\mathbf{w}_t^n$ and $\tilde{\mathbf{c}}_t^m$ using $\mathcal{D}_p$ first. In our design, we can use both labeled and unlabeled data to finetune models but with different loss functions.

3.2 Client Update

The obtained personalized model $\hat{\mathbf{w}}_t^n$ (i.e., $\mathbf{c}_t^*$ in Eq. (2)) will be distributed to the n-th client if it is selected in the next communication

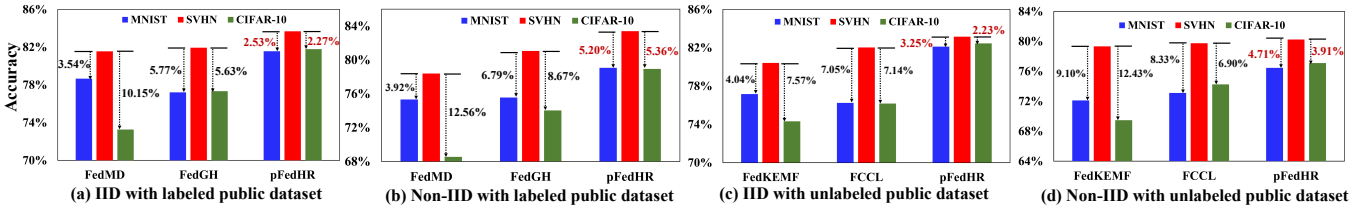

Figure 2: Performance changes when using different public data. `FedLEGO` is our proposed model.

round $t+1$. $\hat{\mathbf{w}}_t^n$ is a reassembled model that carries external knowledge from other clients, but its network structure is different from the original $\mathbf{w}_t^n$. To incorporate the new knowledge without training $\mathbf{w}_t^n$ from scratch, we propose to apply knowledge distillation on the client following [27].

Let $\mathcal{D}_n = \{(\mathbf{x}_i^n, \mathbf{y}_i^n)\}$ denote the labeled data, where $\mathbf{x}_i^n$ is the data feature and $\mathbf{y}_i^n$ is the coresponding ground truth vector. The loss of training local model with knowledge distillation is defined as follows:

$$\mathcal{J}_n = \frac{1}{|\mathcal{D}_n|} \sum_{i=1}^{|\mathcal{D}_n|} \left[\mathrm{CE}(\mathbf{w}_t^n(\mathbf{x}_i^n), \mathbf{y}_i^n) + \lambda \mathrm{KL}(\boldsymbol{\alpha}_t^n(\mathbf{x}_i^n), \hat{\boldsymbol{\alpha}}_t^n(\mathbf{x}_i^n)) \right], \tag{6}$$

where $|\mathcal{D}_n|$ denotes the number of data in $\mathcal{D}_n$, $\mathbf{w}_t^n(\mathbf{x}_i^n)$ means the predicted label distribution, λ is a hyperparameter, $\mathrm{KL}(\cdot,\cdot)$ is the Kullback–Leibler divergence, and $\boldsymbol{\alpha}_t^n(\mathbf{x}_i^n)$ and $\hat{\boldsymbol{\alpha}}_t^n(\mathbf{x}_i^n)$ are the logits from the local model $\mathbf{w}_t^n$ and the downloaded personalized model $\hat{\mathbf{w}}_t^n$, respevtively.

4 EXPERIMENT

4.1 Experiment Setup

We conduct experiments for the image classification task on MNIST, SVHN, and CIFAR-10 datasets under both IID and non-IID data distribution settings, respectively. We split the datasets into 80% for training and 20% for testing. During training, we randomly sample 10% training data to put in the server as $\mathcal{D}_p$ and the remaining 90% to distribute to the clients. The training and testing datasets are randomly sampled for the IID setting. For the non-IID setting, each client randomly holds two classes of data. To test the personalization effectiveness, we sample the testing dataset following the label distribution as the training dataset. We compare `FedLEGO` with the baselines. To make fair comparisons, we use FedMD [8] and FedGH [25] as baselines when using the *labeled public data*, and FCCL [5] and FedKEMF [26] when testing the *unlabeled public data*.

4.2 Experiment Evaluation

Similar to existing work [5], to test the performance with a small number of clients, we set the client number $N = 12$ and active client number $B = 4$ in each communication round. We design 4 types of models with different structures and randomly assign each type of model to 3 clients. The *Conv* operation contains convolution, max pooling, batch normalization, and ReLu, and the *FC* layer contains fully connected mapping, ReLU, and dropout. We set the number of clusters $K = 4$. Then local training epoch and the server finetuning epoch are equal to 10 and 3, respectively. The public data and client data are from the same dataset. Table 1 shows the experimental results for the heterogeneous setting using both labeled and unlabeled public data. We can observe that the proposed `FedLEGO` achieves state-of-the-art performance on all datasets and settings. We also find the methods using labeled public datasets can boost the performance compared with unlabeled public ones in general, which aligns with our expectations and experiences.

4.3 Public Dataset Analysis

In the previous experiments, the public and client data are from the same dataset, i.e., having the same distribution. To validate the effect of using different public data during model learning for all baselines and our model, we conduct experiments by choosing public data from different datasets and report the results on the SVHN dataset. Other experimental settings are the same as those in the scenario of the small number of clients.

Figure 2 shows the experimental results for all approaches using labeled and unlabeled public datasets. We can observe that replacing the public data will make all approaches decrease performance. This is reasonable since the data distributions between public and client data are different. However, compared with baselines, the proposed `FedLEGO` has the **lowest performance drop**. *Even using other public data, `FedLEGO` can achieve comparable or better performance with baselines using SVHN as the public data*. This advantage stems from our model design. As described in Section 3.1.3, we keep more information from original client models by using a simple layer as the stitch. Besides, we aim to search for the most similar personalized candidate with a client model. We propose to calculate the average logits in Eq. (5) as the criteria. To obtain the logits, we do not need to finetune the models many times. In our experiments, we set the number of finetuning epochs as 3. This strategy can also help the model reduce the adverse impact of public data during model training.

Due to the page limit, more content can be found in **supplementary materials**.

5 CONCLUSION

We propose a novel framework, named `FedLEGO`, for personalized federated learning, focusing on solving the problem of heterogeneous model cooperation. The experimental results conducted on three datasets, under both IID and Non-IID settings, have verified the effectiveness of our proposed `FedLEGO` framework in addressing the model heterogeneity issue in federated learning. The achieved state-of-the-art performance serves as evidence of the efficacy and practicality of our approach.

REFERENCES

[1] Samuel K Ainsworth, Jonathan Hayase, and Siddhartha Srinivasa. 2022. Git re-basin: Merging models modulo permutation symmetries. *arXiv preprint arXiv:2209.04836* (2022).

[2] Yamini Bansal, Preetum Nakkiran, and Boaz Barak. 2021. Revisiting model stitching to compare neural representations. *Advances in neural information processing systems* 34 (2021), 225–236.

[3] Fengwen Chen, Guodong Long, Zonghan Wu, Tianyi Zhou, and Jing Jiang. 2022. Personalized federated learning with graph. *arXiv preprint arXiv:2203.00829* (2022).

[4] Avishek Ghosh, Jichan Chung, Dong Yin, and Kannan Ramchandran. 2020. An efficient framework for clustered federated learning. *Advances in Neural Information Processing Systems* 33 (2020), 19586–19597.

[5] Wenke Huang, Mang Ye, and Bo Du. 2022. Learn from others and be yourself in heterogeneous federated learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 10143–10153.

[6] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. 2021. Advances and open problems in federated learning. *Foundations and Trends® in Machine Learning* 14, 1–2 (2021), 1–210.

[7] Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. 2019. Similarity of neural network representations revisited. In *International Conference on Machine Learning*. PMLR, 3519–3529.

[8] Daliang Li and Junpu Wang. 2019. Fedmd: Heterogenous federated learning via model distillation. *arXiv preprint arXiv:1910.03581* (2019).

[9] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. 2020. Federated optimization in heterogeneous networks. *Proceedings of Machine learning and systems* 2 (2020), 429–450.

[10] 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* 33 (2020), 2351–2363.

[11] Lingjuan Lyu, Han Yu, Xingjun Ma, Chen Chen, Lichao Sun, Jun Zhao, Qiang Yang, and S Yu Philip. 2022. Privacy and robustness in federated learning: Attacks and defenses. *IEEE transactions on neural networks and learning systems* (2022).

[12] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. 2017. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*. PMLR, 1273–1282.

[13] Dang Nguyen, Trang Nguyen, Khai Nguyen, Dinh Phung, Hung Bui, and Nhat Ho. 2023. On Cross-Layer Alignment for Model Fusion of Heterogeneous Neural Networks. In *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 1–5.

[14] Zizheng Pan, Jianfei Cai, and Bohan Zhuang. 2023. Stitchable Neural Networks. *arXiv preprint arXiv:2302.06586* (2023).

[15] Krishna Pillutla, Kshitiz Malik, Abdel-Rahman Mohamed, Mike Rabbat, Maziar Sanjabi, and Lin Xiao. 2022. Federated learning with partial model personalization. In *International Conference on Machine Learning*. PMLR, 17716–17758.

[16] Jiankai Sun, Xin Yang, Yuanshun Yao, Aonan Zhang, Weihao Gao, Junyuan Xie, and Chong Wang. 2021. Vertical federated learning without revealing intersection membership. *arXiv preprint arXiv:2106.05508* (2021).

[17] Lichao Sun and Lingjuan Lyu. 2020. Federated model distillation with noise-free differential privacy. *arXiv preprint arXiv:2009.05537* (2020).

[18] Yue Tan, Guodong Long, Lu Liu, Tianyi Zhou, Qinghua Lu, Jing Jiang, and Chengqi Zhang. 2022. Fedproto: Federated prototype learning across heterogeneous clients. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 36. 8432–8440.

[19] Yue Tan, Guodong Long, Jie Ma, Lu Liu, Tianyi Zhou, and Jing Jiang. 2022. Federated learning from pre-trained models: A contrastive learning approach. *arXiv preprint arXiv:2209.10083* (2022).

[20] Jianyu Wang, Qinghua Liu, Hao Liang, Gauri Joshi, and H Vincent Poor. 2020. Tackling the objective inconsistency problem in heterogeneous federated optimization. *Advances in neural information processing systems* 33 (2020), 7611–7623.

[21] Jiaqi Wang, Shenglai Zeng, Zewei Long, Yaqing Wang, Houping Xiao, and Fenglong Ma. 2023. Knowledge-Enhanced Semi-Supervised Federated Learning for Aggregating Heterogeneous Lightweight Clients in IoT. In *Proceedings of the 2023 SIAM International Conference on Data Mining (SDM)*. SIAM, 496–504.

[22] Chuhan Wu, Fangzhao Wu, Lingjuan Lyu, Yongfeng Huang, and Xing Xie. 2022. Communication-efficient federated learning via knowledge distillation. *Nature communications* 13, 1 (2022), 2032.

[23] Yue Wu, Shuaicheng Zhang, Wenchao Yu, Yanchi Liu, Quanquan Gu, Dawei Zhou, Haifeng Chen, and Wei Cheng. 2023. Personalized Federated Learning under Mixture of Distributions. *arXiv preprint arXiv:2305.01068* (2023).

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

[25] Liping Yi, Gang Wang, Xiaoguang Liu, Zhuan Shi, and Han Yu. 2023. FedGH: Heterogeneous Federated Learning with Generalized Global Header. *arXiv preprint arXiv:2303.13137* (2023).

[26] Sixing Yu, Wei Qian, and Ali Jannesari. 2022. Resource-aware Federated Learning using Knowledge Extraction and Multi-model Fusion. *arXiv preprint arXiv:2208.07978* (2022).

[27] Ying Zhang, Tao Xiang, Timothy M Hospedales, and Huchuan Lu. 2018. Deep mutual learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 4320–4328.

[28] Tianfei Zhou and Ender Konukoglu. 2023. FedFA: Federated Feature Augmentation. *arXiv preprint arXiv:2301.12995* (2023).

APPENDIX

Pseudo-code of FedLEGO

Algorithm 1 shows the pseudo-code of the proposed FedLEGO model, which contains two main updates: the server update (lines 3-11) and the client update (lines 12 - 17).

Algorithm 1: Algorithm Flow of FedLEGO.

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 **ServerUpdate:**
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 **ClientUpdate:**
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**

5.1 Implementation Details

The proposed FedLEGO 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.

Large Number of Clients

We also test the performance of models with a large number of clients. When the number of active clients B is large, calculating layer pair-wise distance values using Eq. (3) will be highly time-consuming. To avoid this issue, a straightforward solution is to conduct FedAvg [12] for averaging the models with the same structures first and then do the function-driven layer grouping based on the averaged structures via Eq. (4). The following operations are the same as FedLEGO. In this experiment, we set $N = 100$ clients and the active number of clients $B = 10$. Other settings are the same as those in the small number client experiment.

Table 2: Evaluation using a large number of clients on the SVHN dataset ($N = 100$).

Public Data	Model	IID	Non-IID
Labeled	FedMD	78.16%	74.34%
	FedGH	76.27%	72.78%
	pFedHR	**80.02%**	**77.63%**
Unlabeled	FedKEAF	76.27%	74.61%
	FCCL	75.03%	71.54%
	pFedHR	**78.98%**	**75.77%**

Table 2 shows the results on the SVHN dataset for testing the proposed FedLEGO for a large number of clients setting. The results show similar patterns as those listed in Table 1, where the proposed FedLEGO achieves the best performance under IID and Non-IID settings whether it uses labeled or unlabeled public datasets. Compared to the results on the SVHN dataset in Table 1, we can find that the performance of all the baselines and our models drops. Because the number of training data is fixed, allocating these data to 100 clients will make each client use fewer data for training, which leads to a performance drop. The results on both small and large numbers of clients clearly demonstrate the effectiveness of our model for addressing the heterogeneous model cooperation issue in federated learning.

Sensitivity Study of Public Data Percentage

Several factors can affect model performance. In this experiment, we aim to investigate whether the percentage of public data is a key factor in influencing performance change. Toward this end, we still use the small number of clients setting, i.e., the client and public data are from the SVHN dataset, but we adjust the percentage of public data. In the original experiment, we used 10% data as the public data. Now, we reduce this percentage to 2% and 5%. The results are shown in Figure 3. We can observe that with the increase in the percentage of public data, the performance of the proposed FedLEGO also improves. These results align with our expectations since more public data used for finetuning can help FedLEGO obtain more accurately matched personalized models, further enhancing the final accuracy.

Personalized Model Visualization

FedLEGO can automatically and dynamically generate candidates for clients using the proposed heterogeneous model reassembly techniques in Section 3.1.2. We visualize the generated models for a client at two different epochs (t and t') to make a comparison in Figure 4. We can observe that at epoch t, the layers are "[*Conv2* from M1, *Conv3* from M2, *Conv4* from M3, *Conv5* from M3, *FC3* from M2]", which is significantly different the model structure at epoch t'. These models automatically reassemble different layers from different models learned by the proposed FedLEGO instead of using predefined structures or consensus information.

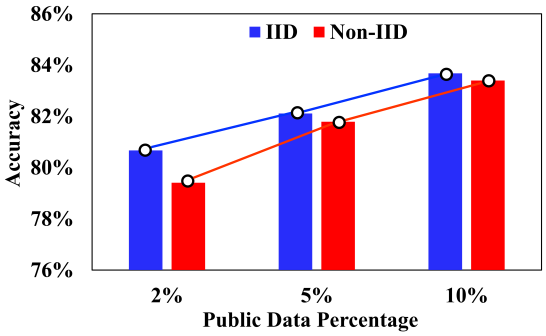

Figure 3: Performance change w.r.t. the percentage of public data.

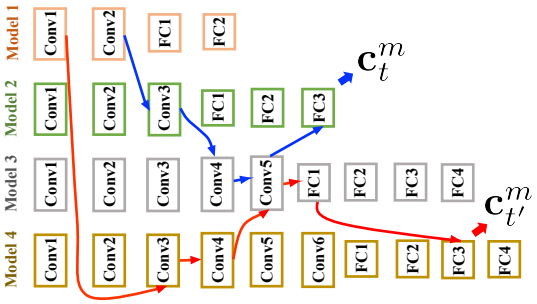

Figure 4: Candidate models.

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009