Mobile Networks and Applications
https://doi.org/10.1007/s11036-022-01978-8

Adaptive Clustered Federated Learning for Heterogeneous Data in Edge Computing

Biyao Gong[1] · Tianzhang Xing[1] · Zhidan Liu[2] · Junfeng Wang[1] · Xiuya Liu[1]

Accepted: 9 February 2022

Abstract
Although federated learning has been widely used in collaborative training of machine learning models, its practical uses are still challenged by heterogeneous data across clients. To alleviate the impact of non-IID data issue, we present an adaptive clustered federated learning approach, `AdaCFL`, which can classify clients into suitable clusters according to their local data distribution and train a specialized model for the clients of each cluster. By exploiting the implicit connection between local model weights and data distribution on clients, `AdaCFL` relies on partial selected model weights to measure the data similarity between clients and adaptively groups them into the optimal number of clusters. Experimental results on three benchmark datasets with various non-IID data settings demonstrate that `AdaCFL` achieves comparably high model accuracy as the state-of-the-art works, yet with a significant reduction on the communication cost.

Keywords Federated learning · Clustered federated learning · Non-IID challenge · Edge computing

1 Introduction

Nowadays hundreds of millions of IoT devices around the world will keep generating huge amount of data every day. Edge computing [5, 30] is a more natural solution than sending huge amount of data over the network to cloud servers [17], due to bandwidth and privacy constraints. To further avoid the privacy leakage risk of edge computing applications, such as mobile application recommendation [14], federated learning becomes an effective solution to analyze and process the huge amount of data stored in edge nodes [27].

✉ Tianzhang Xing
xtz@nwu.edu.cn

✉ Zhidan Liu
liuzhidan@szu.edu.cn

Biyao Gong
gby@stumail.nwu.edu.cn

Junfeng Wang
wangjunfeng@stumail.nwu.edu.cn

Xiuya Liu
lxya@stumail.nwu.edu.cn

[1] School of Information Science and Technology, Northwest University, Xi'an 710127, China

[2] College of Computer Science and Software Engineering, Shenzhen University, Shenzhen 518060, China

Federated learning [18, 29] is a novel computing paradigm of distributed machine learning that is able to collectively train a globally shared model across multiple decentralized clients (*e.g.*, mobile devices) holding local data samples, without exchanging them. Thanks to its protections on data privacy and data security, federated learning has been widely used in many applications, *e.g.*, natural language processing [34], computer vision [15], medicine [6] and finance [33].

Despite the huge advantages, the applications of federated learning in practice still encounter several major challenges, one of which is the non-IID data issue [1, 18]. Since users may have different device usage patterns, thus the resultant data samples and corresponding labels are usually not independent and identically distribution (*i.e.*, non-IID). Previous studies demonstrate that joint learning over heterogeneous data will not only increase communication cost for convergence of the shared model, but also greatly degrade the model accuracy [21]. As a result, non-IID data across client devices will hinder further adoptions of federated learning in domains like recommender systems, where non-IID data is preferable for improving personalized services [7, 31].

Because data samples on client devices cannot be accessed or audited by the centralized server, previous works thus implicitly address non-IID data issue with novel model updating strategies. For example, McMahan *et al*. [18] presented the Federated Averaging (*FedAvg*) algorithm that

Published online: 05 May 2022

computes one shared model by averaging model parameters from all client devices, instead of applying traditional gradient decent updates. A recent work [25] optimized *FedAvg* by intelligently selecting the clients to participate in each round of federated learning using deep reinforcement learning. Due to the heterogeneous data distributions, a *single* model may not achieve the best performance for all clients, while multiple specialized models are required for providing better services. Therefore, an emerging framework, named as *Clustered Federated Learning (CFL)* [22], has been proposed to effectively alleviate the impact of non-IID data, while still preserving the performance in terms of model accuracy and communication cost. Most of existing CFL-based approaches [3, 7, 22] primarily group clients into clusters with jointly trainable data distributions and accordingly train a shared model for each cluster. Specifically, they measure the similarity between clients' data distributions indirectly based on their local model updates or the gradients.

Although CFL framework is attractive, the existing CFL-based approaches are still inefficient. First of all, they mainly exploit local model updates or the gradients to approximately represent the unknown data distribution of each client, while we find such an approximation is not effective and efficient to guide the clustering of participant clients. Abundant local model updates or gradients make it computationally hard to measure the similarity between clients' data distributions, and meanwhile cause a long process for the model convergence. In addition, the performance of CFL-based approaches heavily relies on the optimal number κ of clusters. The number κ, however, is empirically set in previous works. In principle, a method to automatically choose the optimal κ that can best reflect the data distributions of all client devices is desired.

To address the limitations of existing works, we propose an Adaptive Clustered Federated Learning (`AdaCFL`) to automatically group clients into suitable clusters. By considering the implicit connection between model weights and client's data distribution [4, 20, 25, 28], `AdaCFL` proposes to group clients based on the model weights derived by training on each client's local data. To reduce data amount for transferring and similarity calculation, `AdaCFL` further refines the selection of model weights on each client. Based on the representative model weights of clients, `AdaCFL` makes use of the hierarchical clustering to iteratively classify clients into the optimal number of clusters. Moreover, `AdaCFL` has a lightweight yet effective mechanism to incorporate the new-coming clients into suitable clusters, without obscuring already trained models. And `AdaCFL` can be applied to recommender systems by clustering users with similar preferences to recommend more comprehensive content for them. We experimentally evaluate `AdaCFL` on different benchmark datasets with various non-IID settings. Experimental results show that `AdaCFL` can achieve comparably high model accuracy as state-of-the-art works, yet with a great reduction on communication cost.

2 Related Work

We discuss the most related works on addressing the data heterogeneity issue in federated learning as follows.

Federated learning and non-IID data. Unbalanced data, including unbalanced distribution and unbalanced sample size, is quite common in the federated setting, where users usually have different device usage patterns, while non-IID data across client devices will significantly harm the performance of federated learning. Specifically, for a movie recommendation application, different users may have different types of preferences, *e.g.*, user A likes action movies, while user B prefers comedy movies, which leads to different distributions of data for different users, *i.e.*, non-IID data; in addition, different users may use the application at different frequencies, which also leads to different sample sizes for different users, *i.e.*, unbalanced data. Non-IID data can cause more serious performance loss to federal learning than unbalanced data. Thus, recent studies [9, 12] have showed that non-IID data will cause slow or even no convergence in the training phase of federated learning, resulting in much more communication rounds. Therefore, how to address the non-IID data issue becomes a hot research topic in federated learning. McMahan *et al*. [18] proposed the federated optimization algorithm *FedAvg*, which aggregates a global model by weighted averaging the parameters of each local model to alleviate the impact of non-IID data. However, Li *et al*. [13] theoretically proved that non-IID data will increase the communication cost of *FedAvg*. To speed up the training of federated learning, a recent work [25] improved *FedAvg* by actively selecting devices to participate in each round of training through a deep reinforcement learning model. Moreover, some works leveraged the heterogeneous data across client devices to train personalized models by exploiting techniques like transfer learning [26] and multi-task learning [24].

Clustered federated learning (CFL) is an emerging framework to attack data heterogeneity for efficient federated learning [22]. The CFL framework proposes to group all client devices into clusters based on their local data distributions and trains an independent model for the clients of each cluster. Since data samples of clients cannot be accessed, [3, 22] thus suggested to group clients based on the cosine similarity of their local model updates or gradients. More specifically, Sattler *et al*. [22] proposed *FMTL*, which iteratively divides clients into clusters. Initially, all clients form a cluster to train a model. Latter, the server calculates the cosine similarity between local model updates of clients in the same cluster, and divides the cluster into two

new clusters according to clients' similarities. *FMTL* repeats above operations until no new cluster is generated. Ghosh *et al*. [7] proposed *IFCA* that evaluates the experience loss of κ global models over clients' local data and then assigns a client to the cluster, where the global model with the lowest experience loss is located. However, the performance of *IFCA* heavily relies on number κ that is empirically set. Although these CFL-based approaches have shown attractive performance, they are still not sufficiently efficient, just as discussed in the next section.

3 Background and Motivation

3.1 Federated Learning

Federated learning is a privacy-protected framework that allows distributed clients to train machine learning model collaboratively without exchanging their local data [18]. In federated settings, there are usually m clients and one central server, where clients are responsible for collecting and storing data, and using the data to train local models, and the server is responsible for aggregating local models in some way to build a globally shared model. The server and clients communicate according to a predetermined communication protocol. Because all raw data are stored by distributed clients, the central server cannot access them and thus has no prior knowledge of the data distributions across devices.

To implement the idea of federated learning, McMahan *et al*. [18] firstly proposed a federated averaging algorithm (*FedAvg*) to optimize the communication efficiency of federated learning over real-world data, which are usually heterogeneous across different client devices. Specifically, *FedAvg* trains a globally shared model by weighted averaging the parameters of clients' local models, and the goal of *FedAvg* is typically to minimize the following objective:

$$\min_{\theta} \left\{ \mathbf{F}(\theta) \triangleq \sum_{i=1}^{m} \frac{n_i}{N} f_i(\theta) \right\}, \tag{1}$$

where we assume client i has local dataset $\mathcal{D}_i$, $n_i = \left\|\mathcal{D}_i\right\|$ and $N = \sum_{i=1}^{m} n_i$. The local objective f_i can be defined as calculating local empirical risk on the dataset $\mathcal{D}_i$. Note that the empirical loss function in federated learning usually depends on the specific task. For instance, for multi-class tasks, cross-entropy loss is defined as $f_i(\theta) = \sum_{a=1}^{C} p(y=a)\mathbb{E}_{x|y=a}[\log(f_a(x,\theta))]$, where f_a represents the probability of predicting sample x as the class a, and C is number of label classes.

It is experimentally proved that *FedAvg* can approximate the model trained on centrally collected data given the data across all clients are IID [18]. In practice, however, the actual data produced by different clients are usually heterogeneous and thus non-IID. The data heterogeneity issue will severely harm the convergence and performance of federated learning in practical applications. Recently, a promising framework, named *clustered federated learning (CFL)* [22], has been proposed to address the data heterogeneity issue, and already attracted some research efforts. We formally define the CFL framework as follows.

Definition 1 (**Clustered Federated Learning (CFL)**) Given m client devices, $\mathbb{C} = \{c_1, \cdots, c_m\}$, owning non-IID data, *CFL* aims to group these devices into a set of disjoint clusters $\mathcal{G} = \{\mathbf{g_1}, \cdots, \mathbf{g_\kappa}\}$, where $\bigcup_{i=1}^{\kappa} \mathbf{g_i} = \mathbb{C}$ and $\mathbf{g_i} \cap \mathbf{g_j} = \varnothing$ $(i \neq j)$. The device data of each cluster are approximately IID and can be used to train a shared model with good performance in terms of communication cost and model accuracy.

There exist some novel CFL-based approaches [3, 7, 22], and they mainly differ in obtaining clusters $\mathcal{G}$ with different similarity measures and κ settings. Specifically, [22] utilizes the cosine similarity between local model updates or gradients to measure the similarity between clients' data distributions. [7] randomly generates κ global models, and then iteratively assigns client c_i into cluster $\mathbf{g_j}$ once the j-th model can derive the highest accuracy for c_i's data. In addition, [3] also adopts the cosine similarity between local model updates, and divides clients into κ clusters through the k-means algorithm.

3.2 Motivation

Although CFL-based solutions [3, 7, 22] have shown great advantages than *FedAvg* over non-IID data, they are still insufficiently efficient due to their clustering strategies. We examine their performance and illustrate the limitations with experiments.

In our experiments, we will train several CNN models with PyTorch on the CIFAR-10 dataset. For a clear illustration, we train CNN models over $m = 100$ client devices with non-IID setting $\alpha = 0.8$ through federated learning (Please see more experimental settings in Sect. 6.1). By analyzing the design and experimental results of existing CFL-based approaches, we observe their limitations as follows:

(1) **Hard to set the optimal κ**. Most of the existing approaches [3, 7] need to specify the number of clusters κ in advance, while in reality the data distributions across clients are unknown and thus the optimal κ is difficult to determine. We conduct an experiment to study the impact of κ on model accuracy by running the approach in [7] for 20 communication rounds. Figure 1(a) plots the results. When we only train one model for all clients (*i.e.*, $\kappa = 1$ that works similarly as *FedAvg*), the model accuracy is the lowest, which

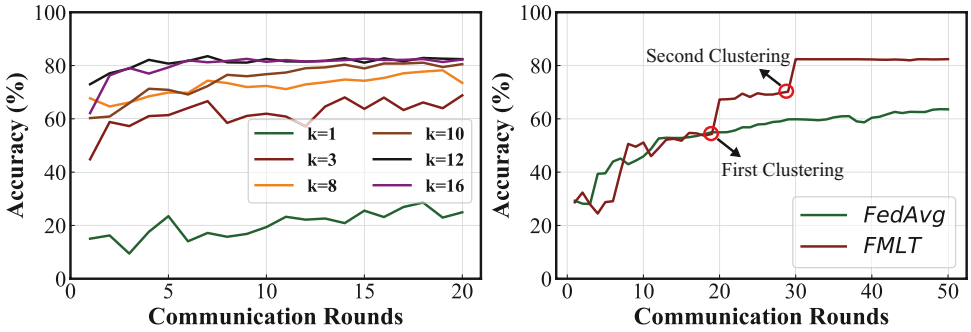

Fig. 1 The empirical study of existing CFL-based approaches (**a**) Impact of κ (**b**) Clustering efficiency

implies that training one single model across non-IID data is inadequate. In addition, when we increase the cluster number κ, the average model accuracy improves as well and achieves the best when $\kappa = 12$. When we further increase κ, the average accuracy, however, drops. This experiment demonstrates that κ is highly relevant with model accuracy and an adaptive mechanism to find the optimal κ is important for CFL-based approaches.

(2) **Unstable clustering efficiency.** There indeed exist a few CFL-based approaches [22] that are able to group clients into a suitable number of clusters. Taking *FMTL* proposed in [22] as an example, once an initial cluster has achieved a stationary model, it would be further separated into two smaller clusters for training more specialized models. Therefore, *FMTL* serves as a post-processing method of federated learning and it requires a long time to derive the optimal κ clusters. We implement *FMTL* and conduct an experiment to test its clustering efficiency. For a clear illustration, we select 30 out of 100 clients, which can be divided into three clusters, to participate in the federated learning for 50 communication rounds. Figure 1(b) compares the performances of *FMTL* and *FedAvg*. It sees that *FedAvg* performs much stably than *FMTL*. Although *FMTL* can search for the best setting of κ, it needs to cluster clients for several times and the efficiency is not good.

(3) **Should all model weights be used?** When we adopt federated learning to train deep leaning models for distributed client devices, CFL-based approaches usually involve the calculation of model similarity which reflects the data distribution, using model weights or model updates. For example, Table 1 summarizes the number of model weights for three simple CNN models designed for different datasets (See more details about the datasets and models in Sect. 6.1). These models are simple, while in practice they are usually more complex, but in mobile computing, federated learning systems usually have to deal with tens of thousands of models. If the similarity between a large number of models is calculated at the same time, it will undoubtedly put a huge pressure on the server. In addition, existing literature has already shown that there are differences between different layers in the same model and that the weights at higher levels are more task-related compared to the weights at lower levels [16, 19, 32]. So, is it possible that the model similarity calculated using partial weights is more conducive to clustering than using all weights?

Considering above limitations, we thus propose an adaptive clustered federated learning approach – `AdaCFL`, which owns the following valuable merits:

- A better indicator of data distribution similarity to guide stable and efficient client clustering.
- An automatic mechanism to find the optimal κ for better model accuracy and communication cost.

4 Observation

Although the existing literature has shown that there are differences between different layers in the same model [16, 19, 32], there is no literature available to study how such differences affect federated learning. In this section, we further experimentally investigate the differences between different layers of the model in federated learning.

We construct a multi-classification task for images over CIFAR-10 [10] with VGG16 [23], Fig. 2 shows four different distance matrices are computed based on the weights

Table 1 Statistics of three CNN model weights

Model	Total weights	Weights per layer
MnistCNN	33500	500 / 25000 / 8000
CifarCNN	6850	450 / 2400 / 4000
FmnistCNN	21200	400 / 12800 / 8000

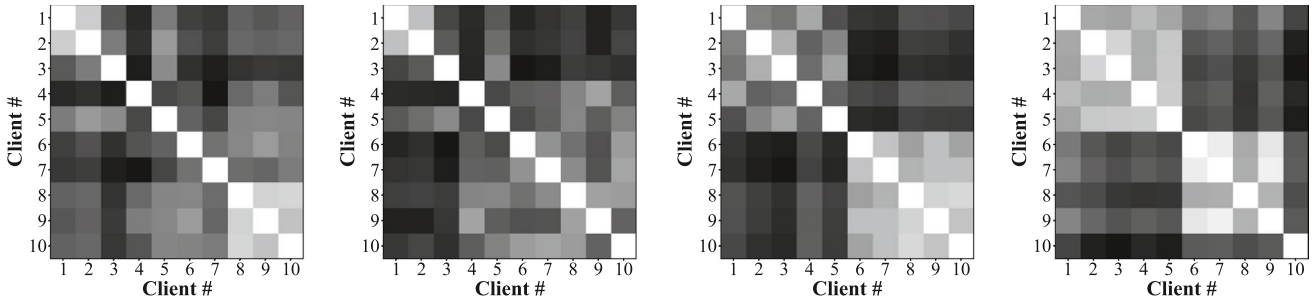

Fig. 2 Visualization of the distance matrices based on the weights of the four different layers, where Conv denotes convolutional layer and FL denotes fully connected layer. The lighter color in the figure indicates the smaller value of the corresponding position in the distance matrix, *i.e.*, the more similar the two models are (**a**) Layer 1 (Conv) (**b**) Layer 7 (Conv) (**c**) Layer 14 (FL) (**d**) Layer 16 (FL)

of the different four layers in VGG16. To simulate non-IID data across clients, we set up 10 different clients and directly divided them into two groups based on the kind of labels that were assigned.

From Fig. 2, we can observe that the distance matrix based on the weights of the different layers in the model reflects the different clustering of clients. Specifically, Fig. 2(a) and (b) show the distance matrices based on two convolutional layers respectively. However, we cannot obviously acquire the cluster structure of the clients from them, while the clustering phenomenon of clients can be clearly observed from Fig. 2(c) and (d). Combining the above experimental results and existing literature [16, 19, 32], we can conclude that the model difference caused by non-IID data are mainly in the fully connected layers or layers with classifier function.

5 Methodology

5.1 Overview

Our proposal AdaCFL follows CFL framework's workflow as well. At the high level, AdaCFL adaptively classifies m clients into κ clusters, *i.e.*, $\mathcal{G} = \{\mathbf{g_1}, \cdots, \mathbf{g_\kappa}\}$, based on a novel similarity measure on their underlying data distributions. Instead of training only one single global model for all clients, the clients of each group $\mathbf{g_i}$ will collectively train one shared model with objective defined as follows:

$$\min_{\theta_{\mathbf{g_i}}} \left\{ \mathbf{F}(\theta_{\mathbf{g_i}}) \triangleq \sum_{c_j \in \mathbf{g_i}} \frac{n_{c_j}}{n_{\mathbf{g_i}}} f_{ij}(\theta_{\mathbf{g_i}}) \right\} \tag{2}$$

where $n_{\mathbf{g_i}}$ and n_{c_j} represent the number of data samples for cluster $\mathbf{g_i}$ and client c_j, respectively, and $f_{ij}(\theta_{\mathbf{g_i}})$ calculates the empirical risk on c_j's local dataset $\mathcal{D}_j$.

Figure 3 illustrates the framework of our proposal AdaCFL. The cycle of AdaCFL is similar to FedAvg, and we embed the client clustering process into the cycle of FedAvg. Specifically, after several communication rounds between the server and the clients, as described in FedAvg, the server calculates the similarity matrix between models based on the partial model weights uploaded by each client. Based on the similarity matrix of model weights, our proposed adaptive clustering can group the clients with similar data distribution into the same cluster. It is noted that the above clustering process is done in one communication round. Given m participate clients, AdaCFL lets each client train a local model using its own data. Different from local model updates or gradients, it is believed that the model weights can better represent c_i's data distribution and AdaCFL proposes to cluster clients based on the similarity of their model weights. Instead of employing all weights of a model, AdaCFL carefully selects partial model weights as the representation of a client's data distribution (Sect. 5.2). At the server side, AdaCFL adaptively classifies all clients into κ clusters based on the similarity of their partial model weights by leveraging the hierarchical clustering algorithm (Sect. 5.3). The clustering results are then sent back to clients and each group will collaboratively train a shared model. Furthermore, AdaCFL can also incorporate new client c_{new}, and intelligently groups it to cluster $\mathbf{g}^*$ that introduces the minimum empirical risk after accepting c_{new} (Sect. 5.4).

5.2 Selection of Model Weights

It is observed that there exist an implicit connection between data distribution on a client c_i and the model weights trained on c_i's dataset [25]. Therefore, AdaCFL makes use of client c_i's model weights to comprehensively represent its data distribution. Once the local model of each client is trained, the distribution difference between clients' local data can be measured by the difference between their model weights,

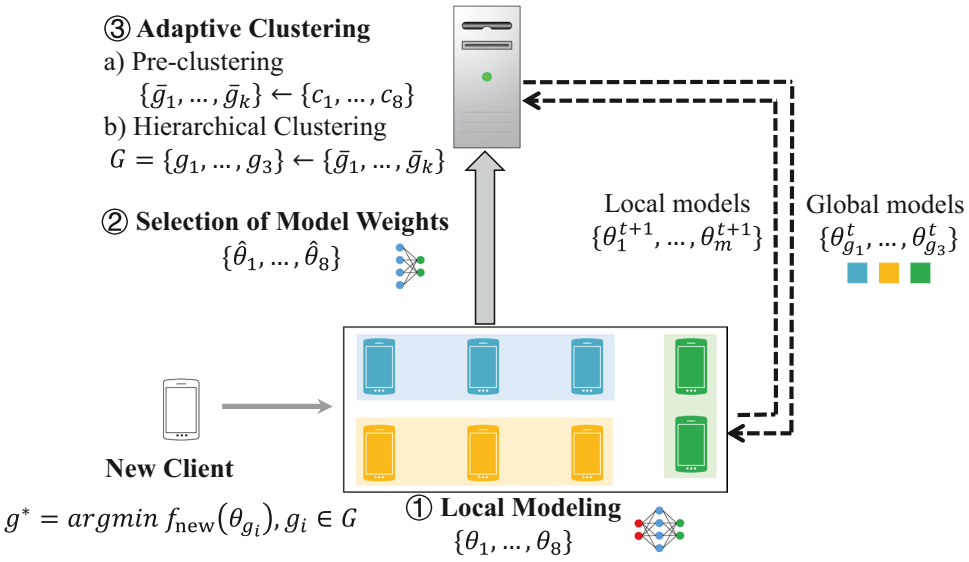

Fig. 3 The framework of our proposal AdaCFL

which is referred as *model distance*. To quantify the model distance between any two clients c_i and c_j, we calculate the l_2 distance between their model weights as:

$$dist(c_i, c_j) = \left\| \theta_{c_i} - \theta_{c_j} \right\|_{l_2}. \tag{3}$$

In principle, if two clients have similar data distributions, they tend to train models with more similar weights compared to clients with dissimilar data, and thus the distance between model weights will be smaller [20, 22, 25, 28]. Hence, model distance can be used as an effective indicator to guide the clustering of clients. After obtaining model weights of all clients, the server will calculate a distance matrix $\mathcal{M}$ of size $m \times m$. Each item $\mathcal{M}_{ij}$ represents the model distance $dist(c_i, c_j)$ between clients c_i and c_j. Assuming the model weight size for each client is p, then the computation overheads for calculating $\mathcal{M}$ is $\mathcal{O}(m^2p^2)$.

Considering that federated learning usually involves tens of thousands of client devices, while target machine learning models (in particular deep learning models) could be complex with numerous parameters, *e.g.*, the *VGG16* model contains $138M$ weights [23]. As a result, the computation cost for model weights based clustering would be extremely huge and thus affects the clustering efficiency.

We have experimentally demonstrated that the higher-layers weights of the model reflect the differences caused by having non-IID data better than the lower-layers weights. Therefore, if the model similarity is calculated using all weights when the number of low-layers weights is a high proportion of the model, a bad similarity matrix will be obtained thus reducing the accuracy of clustering.

To reduce the computation overheads and improve clustering accuracy, we choose partial weights $\hat{\theta}_{c_i}$ of the local model on client c_i, rather than all model weights θ_{c_i}, for similarity measures. For deep learning models proposed for classification tasks, *e.g.*, CNN models, the convolutional layers are designed to extract features of the input, while the fully connected layer aims to achieve the goal of classification. Therefore, the weights of fully connected layer are task-specific, and thus are more task-related. In AdaCFL, we thus select the partial weights, which include weights and bias from fully connected layers with the least number of parameters, to approximately represent the whole model. We use a convolutional neural network as an example. In fact, we select the layer with the lowest number of weights from the layers with classifier function in the model, including CNN, RNN and MLP, as a representative of the all model weights.

Specifically, the partial model weights are determined by $\hat{\theta}_{c_i} = \arg\min ||\theta_{c_i}^{[a]}||_{l_1}, a \in \mathbb{L}$, where $\mathbb{L}$ represents the set of layers for client c_i's model and $||\theta^{[a]}||_{l_1}$ returns the number of weights in the a-th fully connected layer. We select partial model weights for each clients, and use these weights to calculate distance matrix $\mathcal{M}$. Since the size of $\hat{\theta}_{c_i}$ is much smaller than θ_{c_i}, the computation cost is thus greatly reduced.

5.3 Adaptive Clustering

To boost the clustering of all client devices, AdaCFL requires each client c_i to report some statistics about its local dataset $\mathcal{D}_i$, including the number of sample classes L_{c_i} and Shannon Entropy s_{c_i}. Specifically, s_{c_i} is computed as

$$s_{c_i} = -\sum_{j \in \{1, \cdots L_{c_i}\}} \frac{b_j}{\|\mathcal{D}_i\|} \log_2\left(\frac{b_j}{\|\mathcal{D}_i\|}\right) \tag{4}$$

where b_j is the number of data samples for class j and $\|\mathcal{D}_i\|$ is the total number of data samples on client c_i. Intuitively, if client devices have similar number of sample classes and Shannon Entropy, they may have similar data distributions and will be grouped together with a higher probability. Thus, we can pre-cluster such clients and then adjust the initial clusters through their partial model weights. `AdaCFL` classifies all clients into clusters with the following two stages:

(1) *Pre-clustering stage.* The server collects simple statistics of data distribution from each client, and roughly classifies all clients into clusters $\mathcal{G}_o = \{\bar{\mathbf{g}}_1, \cdots, \bar{\mathbf{g}}_k\}$ according to their Shannon Entropy with a gap of 10%.
(2) *Hierarchical clustering stage.* For each cluster $\bar{\mathbf{g}}_j$ belonging to $\mathcal{G}_o$, the server will calculate a distance matrix $\mathcal{M}^j$ for its cluster members. Compared to compute model distances among m clients, the total computation costs for deriving distance matrices for all initial clusters are further reduced. Based on the distance matrices, *i.e.*, $\{\mathcal{M}^1, \cdots \mathcal{M}^k\}$, for all initial clusters, the server runs hierarchical clustering algorithm [2] to perform fine-grained clustering on all clients, without specifying the desired number of clusters. In general, hierarchical clustering relies on an agglomerative strategy. It calculates the similarity between any two data points, merges the two most similar data points and iterates this process until the termination condition is met. In our work, the distance matrices are used as the input of hierarchical clustering, and initially all clients have been pre-clustered. The nearest clusters are merged iteratively until the distance between them is greater than the distance threshold λ. The distance threshold λ in hierarchical clustering can be set enlighteningly, depending on the obtained hierarchy, which is simple and less costly compared to the setting of parameters in other algorithms. Finally, we will obtain κ clusters $\mathcal{G} = \{\mathbf{g}_1, \cdots, \mathbf{g}_\kappa\}$, where clients of each cluster should have the most similar data distributions and can train a shared model achieving the best performance. Unlike existing methods that require the developer to set the number of clusters κ, adaptive clustering can find the optimal κ from the similarity matrix, as long as the similarity matrix can reflect the client cluster signs. Algorithm 1 shows the details of the clustering process.

Algorithm 1 Adaptive Clustering

Input: m participate clients, threshold λ.
Output: $\mathcal{G} = \{\mathbf{g}_1, \cdots, \mathbf{g}_\kappa\}$.

for each client c_i **in parallel do**
  $L_{c_i} \leftarrow$ the number of sample classes in client c_i
  Calculate s_{c_i} using Eq. (4)
  Client c_i uploads L_{c_i} and s_{c_i} to the server
end for
Server pre-clusters clients with the same L_{c_i} and similar s_{c_i} with gap of 10%: $\mathcal{G}_0 = \{\bar{\mathbf{g}}_1, \cdots, \bar{\mathbf{g}}_k\} \leftarrow \{c_1, \cdots, c_m\}$
Server calculates the similarity matrix of each initial cluster: $\{\mathcal{M}^1, \cdots \mathcal{M}^k\} \leftarrow dist(\mathcal{G}_0)$
Server runs hierarchical clustering (HC) on clusters $\mathcal{G}_0$: $\mathcal{G} = \{\mathbf{g}_1, \cdots, \mathbf{g}_\kappa\} \leftarrow HC(\{\mathcal{M}^1, \cdots \mathcal{M}^k\})$

5.4 Incorporating New Client

In practice, due to unstable client communication or other resource constraints (*e.g.*, energy power on mobile devices), client devices may join in or drop out of the federated learning process. The client quit events actually have no influences on the model training of their original clusters. However, we need to carefully handle the events of new-coming clients. New-coming clients include clients that are not in the existing client set and clients whose local data has changed. If the clients with changed local data are not re-clustered,

serious non-IID data issue may occur within the clients cluster. In order to group each new client c_{new} into one appropriate cluster, c_{new} is required to train a local model using its own data, and then transmits partial selected weights to the server. For each cluster, AdaCFL maintains a copy of its partial model weights. Once receiving the partial model weights from new client c_{new}, the server will calculate the model distances between c_{new}'s model and all global models of existing clusters. The cluster $\mathbf{g}^*$, which has the minimum model distance with c_{new} as expressed in Eq. (5), will accept c_{new}, and updates its shared model with data owned by client c_{new}.

$$\mathbf{g}^* = \arg\min_{\mathbf{g_j}} dist(\hat{\theta}_{c_{new}}, \hat{\theta}_{\mathbf{g_j}}), \quad \mathbf{g_j} \in \mathcal{G}. \tag{5}$$

It is possible that c_{new} has distinct data distribution from existing client devices, thus it can form a new cluster individually once the model distances between c_{new} and any existing cluster is larger than a threshold ε. In practice, the threshold ε is usually set to the maximum distance between existing cluster models, *i.e.*, $\varepsilon = \max dist(\hat{\theta}_{\mathbf{g_i}}, \hat{\theta}_{\mathbf{g_j}})$. Moreover, when some client devices have significant changes on the usage behaviors, they will have quite different data distributions accordingly. Such clients can also be treated as new clients to join in other existing clusters or form new ones. Algorithm 2 depicts the pseudocode of incorporating a new client.

Algorithm 2 Incorporating New Client

Input: Partial model weights $\{\hat{\theta}_{\mathbf{g_1}}, \cdots, \hat{\theta}_{\mathbf{g}_\kappa}\}$, client c_{new}.
Output: A cluster $\mathbf{g}^*$ to incorporate c_{new}.

$\theta_{new} \leftarrow$ New client c_{new} trains a local model
Select partial model weights: $\hat{\theta}_{new} \leftarrow \arg\min \theta^{[a]}_{new l_1}, a \in \mathbb{L}$
Server classifies the new client c_{new}:
$\mathbf{g}^* = \arg\min_{\mathbf{g_j}} dist(\hat{\theta}_{c_{new}}, \hat{\theta}_{\mathbf{g_j}}), \quad \mathbf{g_j} \in \mathcal{G}$
$\mathbf{g}^* = \mathbf{g}^* \cup c_{new}$

Privacy and security analysis: Combining our proposed weight selection, adaptive clustering and incorporating new client can achieve a more efficient and more adaptive CFL approach. In addition, similar to FedAvg, AdaCFL only requires individual clients to report local model weights and the Shannon entropy computed locally. Since the Shannon entropy is computed locally at each client, the server cannot know the specific data distribution of the client nor can it infer the raw data of the client.

6 Evaluation

6.1 Experimental Setup

We have implemented AdaCFL with PyTorch, and conducted performance evaluation in a powerful server that is equipped with AMD 2600X CPU and GTX 1660Ti GPU. In AdaCFL, we set threshold $\lambda = 1.4$ and $\varepsilon = 1.8$. We compared AdaCFL with baseline approaches by training popular CNN models on benchmark datasets under different non-IID data settings.

Baseline approaches. We choose *FedAvg* [18] and two representative CFL-based approaches, *i.e.*, *FMTL* [22] and *IFCA* [7], for performance comparisons. As introduced in Sect. 2, *FedAvg* trains a single model by averaging the weights of all clients' local models, while both *FMTL* and *IFCA* follow CFL framework. In particular, *FMTL* works with an empirical κ setting, while *IFCA* can form the suitable clusters.

Datasets and models. We evaluate all approaches on three publicly open datasets with well-tuned model parameters and different data distributions, following non-IID data settings similar as [18] and [25].

- **MNIST** [11] contains 10 classes of handwritten digits, where the size of each sample is 28×28. We train a CNN model, *i.e.*, *MnistCNN*, which has two 5×5 convolutional layers and each convolutional layer is followed by a 2×2 max-pooling layer. The numbers of output channels for the two convolutional layers are 20 and 50, respectively. Each output channel has a bias. Each client uses the batch size as 100 for model training.
- **CIFAR-10** [10] contains 10 classes of RGB images, where the size of each sample is 32×32. We train a CNN model, *i.e.*, *CifarCNN*, which has two 5×5 convolutional layers and each convolutional layer is followed by a 2×2 max-pooling layer. The numbers of output channels for

the two convolutional layers are 6 and 16, respectively. Each output channel has a bias. On each client, the batch size is set as 50.

- **FashionMNIST** [8] contains 10 classes of images, where the size of each sample is 28×28. We train a CNN model, *i.e.*, *FmnistCNN*, which has two 5×5 convolutional layers and each convolutional layer is followed by a 2×2 max-pooling layer. The numbers of output channels for the two convolutional layers are 16 and 32, respectively. Each output channel has a bias. Each client uses the batch size as 150 for model training.

Table 1 summarizes the weight statistics about the three models. We adopt two strategies to simulate the non-IID data distributions. (1) Similar as [25], we use a parameter α to produce different levels of non-IID data. For instances, $\alpha = 1.0$ means that each client only contains samples of one class, and $\alpha = 0.8$ means that 80% of the samples in each client belong to the same class while the remaining 20% belong to other classes. (2) Similar as [18], each client contains two classes of samples, and there are 100 clients, which can be divided into 10 groups according to their local data. We use $\alpha = B$ to denote this setting. For all datasets and models, we set the learning rate η at training to 0.01, which allows the local model to converge quickly and correctly. And the number of local epochs is to 1, which is consistent with the *FMTL*.

Performance metrics: We use the *number of communication rounds* and *model accuracy* as the performance metrics. In general, fewer communication rounds and higher model accuracy are preferable.

6.2 Results and Analysis

Performance comparison. Table 2 shows the performance comparisons among all approaches by training models in 50 rounds. For each experiment, we record the final model accuracy and the communication rounds to achieve this accuracy. For example, the result 97.03%(15) in Table 2 means that the final model accuracy is 97.03% and *FedAvg* achieves this accuracy at the 15-th round. In general, when we increase α (*i.e.*, from 0.5 to 1.0), the average model accuracy of all approaches except *FedAvg* improves. It is because when α increases, clients' data are more heterogeneous, and one single model trained by *FedAvg* becomes inadequate to accurately model all distributed data. In contrary, CFL-based approaches perform better by clustering clients owning similar data distributions to train an independent model. For the second setting (*i.e.*, $\alpha = B$), CFL-based approaches also significantly outperform *FedAvg* on both model accuracy and communication rounds. Among the three CFL-based approaches, although *IFCA* has won more times of the highest accuracy than `AdaCFL`, we find the accuracy gap between `AdaCFL` and *IFCA* is extremely small, *i.e.*, 0.04% $\sim$ 1.05%. On the other hand, we find `AdaCFL` can always have the fewest rounds to achieve the converged accuracy, which implies that our clustering results are much better than both *FMTL* and *IFCA* through correctly distinguishing clients' data distributions. Table 3 shows the ground truth of number of clusters and the actual number of clusters under different experimental settings. Only at $\alpha = 0.5$, there is a slight gap between the actual value and ground truth, while at other experimental settings ground truth and actual values agree. This shows that our method can find the optimal number of clusters in most experimental settings.

In summary, our proposal `AdaCFL` is able to achieve comparably higher model accuracy as the state-of-the-art works, *e.g.*, *IFCA*, yet with a significant reduction on communication rounds, *e.g.*, on average by 6.26×.

Table 2 Performance comparisons of different approaches over different datasets and various non-IID data settings. The number in brackets indicates communication rounds to converge to final accuracy. The best results in each experiment setting are marked in bold

α	Approach	MNIST	CIFAR-10	FMNIST
0.5	*FedAvg*	**97.03%** (15)	35.40% (38)	81.94% (22)
	FMTL	96.77% (10)	53.94% (31)	83.12% (16)
	IFCA	96.12% (8)	**58.53%** (6)	**85.19%** (14)
	`AdaCFL`	96.53% (**4**)	57.51% (**2**)	84.97% (**7**)
0.8	*FedAvg*	96.62% (14)	31.20% (42)	76.03% (18)
	FMTL	97.22% (12)	63.43% (25)	85.56% (16)
	IFCA	98.06% (8)	79.61% (3)	91.23% (11)
	`AdaCFL`	**99.12%** (**5**)	**79.87%** (**2**)	**91.56%** (**9**)
1.0	*FedAvg*	94.37% (20)	26.15% (33)	79.07% (30)
	FMTL	97.64% (15)	82.27% (38)	97.03% (36)
	IFCA	**99.83%** (5)	**99.81%** (6)	**99.96%** (4)
	`AdaCFL`	99.67% (**2**)	99.73% (**3**)	99.92% (**2**)
B	*FedAvg*	95.95% (16)	44.61% (32)	81.80% (26)
	FMTL	97.65% (24)	78.85% (37)	93.73% (33)
	IFCA	**99.38%** (5)	**86.95%** (14)	97.59% (6)
	`AdaCFL`	99.12% (**3**)	85.90% (**10**)	**98.93%** (**3**)

Table 3 The ground truth of number of clusters and the actual number of clusters under different experimental settings

α	Approach	MNIST	CIFAR-10	FMNIST
0.5	Ground truth	5	10	5
	Practical	5	8	4
0.8	Ground truth	5	10	5
	Practical	5	10	5
1.0	Ground truth	5	10	5
	Practical	5	10	5
B	Ground truth	10	10	10
	Practical	10	10	10

Table 4 Performance comparisons between clustering with complete weights and clustering with only partial weights

α	Weights	MNIST	CIFAR-10	FMNIST
0.8	*Complete*	95.94% (12)	32.47% (37)	77.68% (19)
	Partial	99.12% (5)	79.87% (3)	91.56% (9)
B	*Complete*	95.84% (14)	38.81% (23)	77.51% (19)
	Partial	99.12% (3)	85.90% (10)	98.93% (3)

Complete *vs* partial model weights. We conduct an experiment to examine the effectiveness of model weight selection, and compare the performances of AdaCFL with complete weights and partial weights. Table 4 presents the results in non-IID settings $\alpha = 0.8$ and $\alpha = B$ on the three datasets. Surprisingly, we find that AdaCFL with partial weights outperforms the version with complete weights in terms of both model accuracy and communication rounds. The reason may be that the majority of CNN model weights come form convolutional layers, while they have little distinction and thus interfere the similarity measures among clients. As a result, the complete model weights cannot be a good indicator to guide the client clustering. In contrary, the model weights selected by AdaCFL are more task-specific and can effectively distinguish the underlying data distributions among clients.

To further verify whether using the fully connected layer with the least number of weights gives similar clustering results as the fully connected layer with a high number of weights, we add a fully connected layer to the MnistCNN, CifarCNN and FmnistCNN. The number of weights of the two fully connected layers are 80000 and 8000 for MnistCNN, 40000 and 4000 for CifarCNN and 80000 and 8000 for FmnistCNN, respectively. Figure 5 shows the model accuracy when the model similarity matrix is calculated using the first fully connected layer and the second fully connected layer, respectively.

Because AdaCFL selects partial model weights for clustering, it thus can reduce the computation overhead of calculating the similarity between clients. As shown in Fig. 4, the computation time is significantly reduced to within 1 second when compared to clustering with complete model weights.

Impact of new clients. To study the impact of new clients, we add 10 new clients after the training phase of all

Table 5 Performance comparisons between clustering with weights of first fully connected layer and second fully connected layer. The first fully connected layer has more weights and the second fully connected layer has fewer weights

α	Weights	MNIST	CIFAR-10	FMNIST
0.8	*First*	99.17%	79.79%	91.60%
	Second	99.14%	79.84%	91.56%
B	*First*	99.14%	85.95%	99.06%
	Second	99.12%	86.10%	98.93%

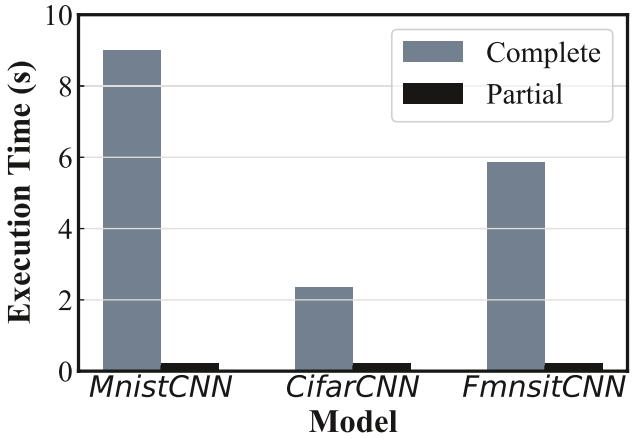

Fig. 4 Computation overhead

shared models (*i.e.*, in the 20-th round), where we set $\alpha = 0.5$ and $\varepsilon = 1.8$ for this experiment. Figure 5 plots the results on the three datasets. The accuracy on each dataset will slightly decrease when new clients are added, but AdaCFL can quickly include new clients to the appropriate clusters and meanwhile new clients will not affect the final model accuracy.

7 Conclusion

This paper proposes an efficient clustered federated learning framework, AdaCFL, to address the data heterogeneity issue. By exploiting the implicit connections between data distribution and model weights, AdaCFL proposes to use partial well-selected model weights to group clients with non-IID data into clusters and trains a specialized model for clients of each cluster. AdaCFL can solve the non-IID problem in federated learning well and can be applied to several scenarios, especially in recommender systems. For example, in recommender system applications, users with similar preferences are divided into groups so that more comprehensive content

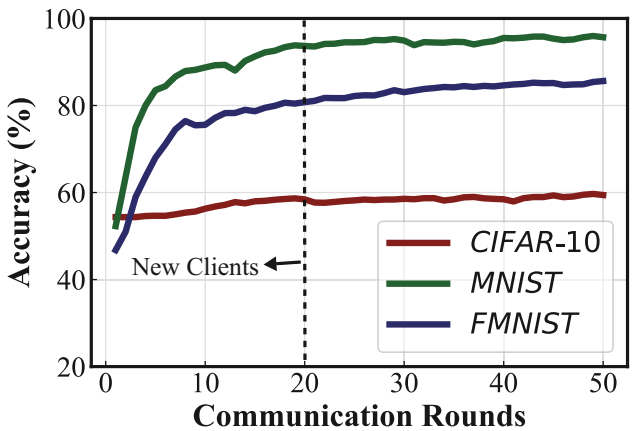

Fig. 5 Impact of new clients

can be recommended for users. We evaluate AdaCFL on three open datasets with various non-IID data settings. Experimental results demonstrate that AdaCFL achieves comparably high model accuracy as state-of-the-art approaches, yet with a significant reduction on the communication cost, *e.g.*, on average by 6.26×.

Acknowledgements This work was supported in part by International Cooperation Project of Shaanxi Province (No. 2020KW-004), the China Postdoctoral Science Foundation (No. 2017M613187), the Shaanxi Science and Technology Innovation Team Support Project under grant agreement (No. 2018TD-026), the China NSFC Grant (No.62172284) and the Natural Science Foundation of Guangdong (General Program No.2020A1515011502).

Data Availability The datasets *i.e.*, MNIST [11], CIFAR-10 [10] and FashionMNIST [8], analysed during the current study are available from http://yann.lecun.com/exdb/mnist/, http://www.cs.toronto.edu/ ~kriz/cifar.html and https://github.com/zalandoresearch/fashion-mnist

References

1. Bonawitz K, Eichner H, Grieskamp W, Huba D, Ingerman A, Ivanov V, Kiddon C, Konečnỳ J, Mazzocchi S, McMahan B et al (2019) Towards federated learning at scale: System design. Proceedings of Machine Learning and Systems 1:374–388
2. Dasgupta S, Long PM (2005) Performance guarantees for hierarchical clustering. J Comput Syst Sci 70(4):555–569
3. Duan M, Liu D, Ji X, Liu R, Liang L, Chen X, Tan Y (2020) Fedgroup: Ternary cosine similarity-based clustered federated learning framework toward high accuracy in heterogeneous data. arXiv preprint arXiv:201006870
4. Gao H, Liu C, Yin Y, Xu Y, Li Y (2021a) A hybrid approach to trust node assessment and management for vanets cooperative data communication: Historical interaction perspective. IEEE Transactions on Intelligent Transportation Systems
5. Gao H, Yin Y, Han G, Zhao W (2021b) Edge computing: Enabling technologies, applications, and services. Transactions on Emerging Telecommunications Technologies 32(6)
6. Gao H, Zhou L, Kim JY, Li Y, Huang W (2021c) Applying probabilistic model checking to the behavior guidance and abnormality detection for mci patients under wireless sensor network. ACM Transactions on Sensor Networks https://doi.org/10.1145/3499426
7. Ghosh A, Chung J, Yin D, Ramchandran K (2020) An efficient framework for clustered federated learning. Adv Neural Inf Process Syst 33:19586–19597
8. Han X, Kashif R, Roland V (2017) The fashionmnist dataset. online: https://github.com/zalandoresearch/fashion-mnist
9. Hsieh K, Phanishayee A, Mutlu O, Gibbons P (2020) The non-iid data quagmire of decentralized machine learning. In: International Conference on Machine Learning, pp 4387–4398
10. Krizhevsky A, Hinton G, et al. (2014) The cifar-10 dataset. online: http://www.cstoronto.edu/~kriz/cifar.html
11. LeCun Y (1998) The mnist database of handwritten digits. online: http://yann lecun com/exdb/mnist/
12. Li T, Sahu AK, Zaheer M, Sanjabi M, Talwalkar A, Smith V (2020a) Federated optimization in heterogeneous networks. vol 2, pp 429–450
13. Li X, Huang K, Yang W, Wang S, Zhang Z (2020b) On the convergence of fedavg on non-iid data. In: International Conference on Learning Representations
14. Liang T, Sheng X, Zhou L, Li Y, Gao H, Yin Y, Chen L (2021) Mobile app recommendation via heterogeneous graph neural network in edge computing. Appl Soft Comput 103:107162
15. Liu Y, Huang A, Luo Y, Huang H, Liu Y, Chen Y, Feng L, Chen T, Yu H, Yang Q (2020) Fedvision: An online visual object detection platform powered by federated learning. Proceedings of the AAAI Conference on Artificial Intelligence 34:13172–13179
16. Long M, Cao Y, Cao Z, Wang J, Jordan MI (2018) Transferable representation learning with deep adaptation networks. IEEE Trans Pattern Anal Mach Intell 41(12):3071–3085
17. Ma X, Xu H, Gao H, Bian M (2021) Real-time multiple-workflow scheduling in cloud environments. IEEE Trans Netw Serv Manage. https://doi.org/10.1109/TNSM.2021.3125395
18. McMahan B, Moore E, Ramage D, Hampson S, Arcas BA (2017) Communication-efficient learning of deep networks from decentralized data. In: Artificial intelligence and statistics, pp 1273–1282
19. Mou L, Meng Z, Yan R, Li G, Xu Y, Zhang L, Jin Z (2016) How transferable are neural networks in nlp applications? In: Conference on Empirical Methods in Natural Language Processing, pp 479–489
20. Ouyang X, Xie Z, Zhou J, Huang J, Xing G (2021) Clusterfl: a similarity-aware federated learning system for human activity recognition. In: Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services, pp 54–66
21. Sattler F, Wiedemann S, Müller KR, Samek W (2019) Robust and communication-efficient federated learning from non-iid data. IEEE transactions on neural networks and learning systems 31(9):3400–3413
22. Sattler F, Müller KR, Samek W (2020) Clustered federated learning: Model-agnostic distributed multitask optimization under privacy constraints. IEEE transactions on neural networks and learning systems 32(8):3710–3722
23. Simonyan K, Zisserman A (2014) Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:14091556
24. Smith V, Chiang CK, Sanjabi M, Talwalkar AS (2017) Federated multi-task learning. vol 30
25. Wang H, Kaplan Z, Niu D, Li B (2020) Optimizing federated learning on non-iid data with reinforcement learning. In: IEEE Conference on Computer Communications, pp 1698–1707
26. Wang K, Mathews R, Kiddon C, Eichner H, Beaufays F, Ramage D (2019) Federated evaluation of on-device personalization. arXiv preprint arXiv:191010252
27. Wang Z, Xu H, Liu J, Huang H, Qiao C, Zhao Y (2021) Resource-efficient federated learning with hierarchical aggregation in edge computing. In: IEEE Conference on Computer Communications, IEEE, pp 1–10
28. Xie M, Long G, Shen T, Zhou T, Wang X, Jiang J, Zhang C (2021) Multi-center federated learning. arXiv preprint arXiv:210808647
29. Yang Q, Liu Y, Chen T, Tong Y (2019) Federated machine learning: Concept and applications. ACM Transactions on Intelligent Systems and Technology 10(2):1–19
30. Yin Y, Cao Z, Xu Y, Gao H, Li R, Mai Z (2020a) Qos prediction for service recommendation with features learning in mobile edge computing environment. IEEE Transactions on Cognitive Communications and Networking 6(4):1136–1145
31. Yin Y, Huang Q, Gao H, Xu Y (2020b) Personalized apis recommendation with cognitive knowledge mining for industrial systems. IEEE Transactions on Industrial Informatics
32. Yosinski J, Clune J, Bengio Y, Lipson H (2014) How transferable are features in deep neural networks? vol 27
33. Zheng W, Yan L, Gou C, Wang FY (2021) Federated meta-learning for fraudulent credit card detection. In: Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence, pp 4654–4660
34. Zhu X, Wang J, Hong Z, Xiao J (2020) Empirical studies of institutional federated learning for natural language processing. In: Findings of the Association for Computational Linguistics: EMNLP, pp 625–634