Journal of Cloud Computing:
Advances, Systems and Applications

RESEARCH **Open Access**

Adaptive device sampling and deadline determination for cloud-based heterogeneous federated learning

Deyu Zhang[1], Wang Sun[1], Zi-Ang Zheng[1], Wenxin Chen[1] and Shiwen He[1*]

Abstract

As a new approach to machine learning, Federated learning enables distributned traiing on edge devices and aggregates local models into a global model. The edge devices that participate in federated learning are highly heterogeneous in terms of computing power, device state, and data distribution, making it challenging to converge models efficiently. In this paper, we propose FedState, which is an adaptive device sampling and deadline determination technique for cloud-based heterogeneous federated learning. Specifically, we consider the cloud as a central server that orchestrates federated learning on a large pool of edge devices. To improve the efficiency of model convergence in heterogeneous federated learning, our approach adaptively samples devices to join each round of training and determines the deadline for result submission based on device state. We analyze existing device usage traces to build device state models in different scenarios and design a dynamic importance measurement mechanism based on device availability, data utility, and computing power. We also propose a deadline determination module that dynamically sets the deadline according to the availability of all sampled devices, local training time, and communication time, enabling more clients to submit local models more efficiently. Due to the variability of device state, we design an experience-driven algorithm based on Deep Reinforcement Learning (DRL) that can dynamically adjust our sampling and deadline policies according to the current environment state. We demonstrate the effectiveness of our approach through a series of experiments with the FMNIST dataset and show that our method outperforms current state-of-the-art approaches in terms of model accuracy and convergence speed.

Keywords Federated learning, Deep reinforcement learning, State heterogeneity

Introduction

With the increase of computing power and memory size of mobile devices, more and more machine learning tasks are deployed on end devices [1–3]. However, due to the problem of bandwidth cost [4–6] and data privacy concerns, It is improper to collect data from clients and conduct centralized training with traditional machine learning paradigms [7–9] in cloud server. To solve these problems, federated learning (FL) [10] is proposed as a new distributed machine learning paradigm. A central cloud server can train a global model in federated learning without collecting private data on edge devices. Clients train a local model with their private data in each iteration and then upload local models instead of raw data to the central cloud server. Finally, the central cloud server will aggregate these local models into a global model according to a specific aggregation algorithm.

Using cloud computing technology, we can leverage cloud resources to facilitate distributed training and federated learning [11–13]. This approach entails training a global model on a central cloud server, without collecting private data on edge device. Federated learning

*Correspondence:
Shiwen He
shiwen.he.hn@csu.edu.cn
[1] School of Computer Science and Engineering, Central South University, Changsha, China

is becoming a popular and effective method of addressing the challenges of bandwidth cost and data privacy in machine learning. With cloud computing, we can more efficiently deploy and utilize federated learning algorithms. Federated learning still faces many challenges, the biggest of which is the problem of heterogeneity [14]. And heterogeneity can be subdivided into data, device, and state. Data heterogeneity refers to the difference in data distribution and data size on different devices. This will impact the direction of convergence and thus the aggregation of the global model [15]. Device heterogeneity means that mobile devices have different computing power and memory sizes. It takes shorter for devices with strong computing power and larger memory to complete the same local iteration round. In synchronous aggregation scenarios, this may cause waiting problems, affecting the time consumed by the aggregation model [16]. State heterogeneity means that the state of mobile devices is extremely variable. For example, the CPU may be busy or idle, the network status may also be unstable, and the battery power is variable. The criteria [17] points out that only devices that meet certain specific conditions can participate in the aggregation process of federated learning. Therefore, once the state of some sampled clients changes during the local training process, these clients cannot upload the local model, thus affecting the accuracy of the global model.

Recently, a lot of work has focused on the impact of heterogeneity on federated learning. Still, most works are based on simulated datasets and ignore the critical problem of state heterogeneity [7, 18–20]. However, state heterogeneity will not only greatly slow down the FL process but also more easily lead to a decline in accuracy. In a real environment, clients may not be able to complete local training for various reasons or upload models after the deadline. For example, the network connection is interrupted, or the CPU changes from idle to busy. Once the client's state changes, the local training cannot be completed, and the model cannot be uploaded in time. More than 11.6% of the devices in the real state dataset cannot upload their model in each round [21], and this proportion will increase with the difficulty of training tasks.

In this article, we consider the federated learning problem in heterogeneous scenarios, and we want to minimize the impact of heterogeneity by designing an efficient client sample strategy and dynamic deadline method. However, we are faced with many challenges. First, to protect the privacy of data on clients, we need to obtain the importance of data on different clients indirectly. Second, because of the heterogeneity of the state, the sampled device may be disconnected for various reasons. When designing the sampling strategy, we should consider this factor to sample devices that are unlikely to be disconnected in the future. Finally, due to the different data sample sizes and computing power of different clients, the time required to complete local training is also different. This will lead to the problem of fast devices waiting for slow devices. If the deadline is too loose, the number of clients that can successfully upload models may increase, but this will also lead to the risk that more users will be disconnected, and the training progress is also slowed down. If the deadline is too tight, the local model may not have learned the characteristics of the data, or many clients may not upload their model in time. Therefore, we need to dynamically adjust the deadline according to data distribution, device computing power and device state.

To this end, we propose FedState, a federated learning framework that can dynamically select clients and set deadlines according to the current environment. Especially in the framework, we get the importance of the devices through the data quality, device status, and device computing power. And utilize deep reinforcement learning to adjust our importance measurement standards according to different stages of training. According to the historical communication time and local training time, we can dynamically adjust the proportion of training samples on each device and the deadline for uploading the model so as to improve the time-to-accuracy of the global model.

The main contributions of this paper are summarized as follows:

- By analyzing the user state dataset, we build the state model for different users. Through a large number of experiments, we quantify the influence of state heterogeneity on the accuracy of FL model.
- We propose a DRL-based sampling strategy and dynamic deadline algorithm in heterogeneous environments.
- We performed experiments on real datasets to evaluate the performance of the proposed algorithm. We compare our method with other methods, and the results further prove the effectiveness of our method.

The rest parts of this paper are organized as follows. The second section introduces the related work. The motivation for this paper is given in Motivation section. System design section presents the system design of our framework. Performance evaluation section comprehensively evaluates the performance of our method with the state-of-art method. And we conclude the paper in Conclusion section.

Related work

As a new distributed machine learning paradigm, Federated learning has attracted extensive research in recent years. The existing work mainly focuses on privacy protection [22, 23], communication efficiency improvement [20, 24], model aggregation [25, 26], Incentive mechanism [27, 28], and application in actual scenarios [28, 29]. In this section, we focus on the sampling strategy and deadline control, which are of great relevance to our work.

Sampling strategy

Which clients should be sampled is a basic problem in federated learning. Now, much work focuses on how to optimize the entire FL process through sampling strategy. For example, Wang et al. proposed a control framework based on reinforcement learning that intelligently chooses clients to maximize a cumulative reward during the training process [20]. Xu et al. designed a new device importance measurement mechanism based on data distribution and device computing power and designed an aggregation algorithm in a heterogeneous environment to reduce the deviation caused by heterogeneity [7]. Wang et al. considered the federated learning in the D2D scenario, proposed a data diversion scheme, provided g rigorous theoretical analysis of data diversion, and used theory to obtain the optimal diversion solution and used GCN to optimize the equipment sampling scheme [30]. Balakrishnan et al. considered the diversity of clients when sampling, which is used to measure how the selected subset of clients represents the whole when aggregated on the server [31]. Zhang et al. uses multi-agent reinforcement learning and adjusts the sampling strategy according to different optimization objectives by using the computing power and training behaviour of the device [32]. Cho et al. observed that selecting clients with high local loss will speed up the convergence. Based on this feature, a Power of Choice client selection strategy is proposed [33]. Wang et al. designed a sampling strategy based on deep reinforcement learning, which maximized the reward by selecting a subset of devices in each communication round. Because the agent needs to select K devices from N devices to participate in FL. But too large action space makes it difficult to train actor network. So the author only samples one device in each round when training and samples the devices with the largest Q value before testing. However, this also leads to different environments during network training and testing, and it may be difficult to sample the best subset of devices [20]. Oort is the most effective client selection scheme in heterogeneous environments. It considers both data and device utility and dynamically adjusts the sampling standard in different training stages [34].

Deadline control

Only a little work has been considered on the dynamic adjustment of the deadline. SmartPC [35] consists of two layers of speed control modules. The global control module sets the deadline by monitoring the status of each device so that more than 60% of devices can upload local models. The local control module adjusts the frequency of each device so that the device can complete the training task with the minimum energy consumption. Fedbalancer [19] defines an indicator called deadline efficiency(DDL-E). The higher deadline of the DDL-E indicator not only allows clients to complete as many tasks as possible but also has higher efficiency. And it selects training samples according to the statistical utility. By adjusting the total number of samples to be trained, the client can upload the local model before the deadline.

Nevertheless, no existing work focuses on state heterogeneity and optimizing time-to-accuracy. The existing methods do not perform well when the client state is heterogeneous. Most of the work assumes that the client state is stable and unchanged, which limits the application of these methods in the real world. In contrast, we quantify the impact of state heterogeneity on FL, design a sampling strategy in a heterogeneous environment, and propose an algorithm to dynamically adjust the deadline to reduce the impact of heterogeneity to a certain extent. In addition, the FedState we propose can automatically learn and apply it to real environments with variable scales and different levels of heterogeneity.

Motivation

Figure 1 shows the performance comparison of FL under the condition of whether state heterogeneity is considered. We can see that state heterogeneity has a great impact on time-to-accuracy.

If the local training time is less than or greater than this period, the time-to-accuracy will certainly be greatly reduced. In order to quantify the impact of device heterogeneity on FL, we conducted a control experiment, as shown in Figure A. The red line refers to the result without considering the state heterogeneity, and the blue line refers to the result under the heterogeneous environment. We adopt the random sampling algorithm and Fedavg aggregation strategy in both scenarios. We can see that state heterogeneity has a great impact on FL. On average, some devices will be disconnected in each round. If a mechanism can be set to reduce the probability of device failure, the time-to-accuracy will be greatly improved. Yang et al. first mentioned the problem of

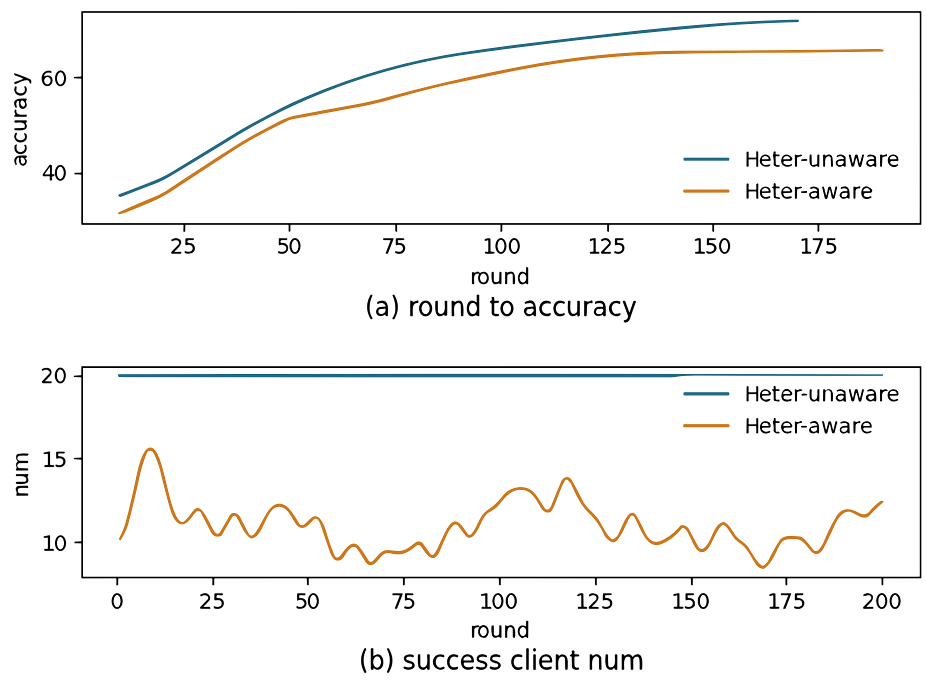

Fig. 1 The effect of state heterogeneity on federated learning algorithms

state heterogeneity in his paper and verified the influence of state heterogeneity on FL through a large number of experiments [21]. However, the author does not give a definition of the degree of state heterogeneity, so it is difficult for us to quantify its impact and give a general solution.

We analyzed a large-scale user behaviour dataset [21] with more than 136K users to obtain the real user behaviour state. According to FL settings, we define the devices with idle CPU and WIFI connected and charging as available. We counted the rate of available devices within 120 hours. We find that the ratio of available devices changes regularly over time. For example, the ratio of available devices is the highest in the early morning and the lowest in the evening. This observation is consistent with our common sense because most users will put down their mobile phones and charge them in the early morning, while the evening is the golden time to stay up late surfing.

We also visualized the proportion of available users in different periods in the behaviour trace. Users with more than 80% available time are defined as high-quality users, those with less than 20% are low-quality users, and those in between are ordinary users. Figure 3 consists of four pie charts, representing four parts of a 24-hour day, with each period covering a 6-hour time span, and showing he proportion of available users over that time period. The size of each slice in the pie chart corresponds to the proportion of users in different state during that time period. We can draw a conclusion from the figure that in different periods, different types of users account for different proportions. In the early morning, high-quality users account for a large proportion, and there is only a small probability that users will be disconnected during this time period. In the afternoon, ordinary users account for a large proportion. Most users have extremely variable statuses, and there is a high probability that the model cannot be uploaded normally. So we can define the degree of state heterogeneity in the system according to the proportion of different types of users.

After classifying the users, we counted the probability of available time slots for different types of users in different time periods. From Figs. 2 and 3, we can find that the probability density curves of the three types of users are highly similar in heterogeneous environments with different proportions, so we can fit their probability density cures, and thus construct different degrees of heterogeneous environment.

System design

In this section, we will discuss the system design of FedState. The system is divided into two modules: the first module is responsible for sampling the client, and the second is responsible for setting a specific deadline. FedState balances model accuracy and training time through these two modules. Specially, we first briefly introduce the system overview and then introduce the specific design of the two modules in detail.

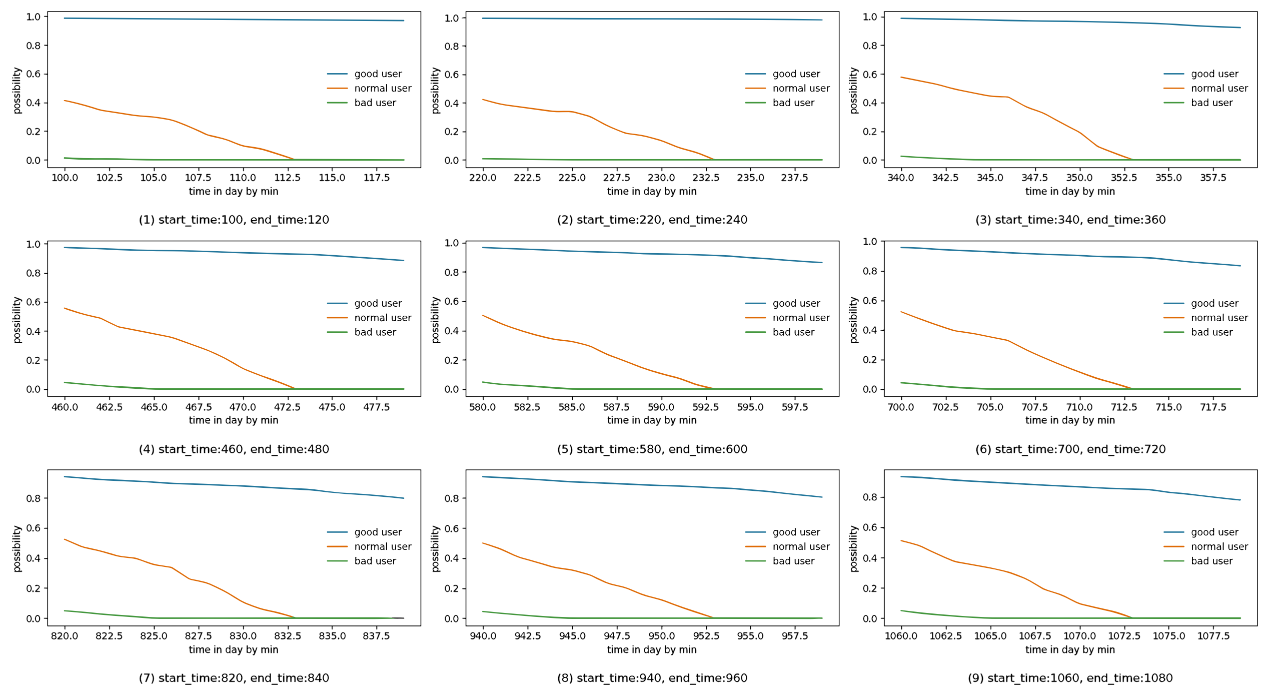

Fig. 2 Probability density plot of available slots

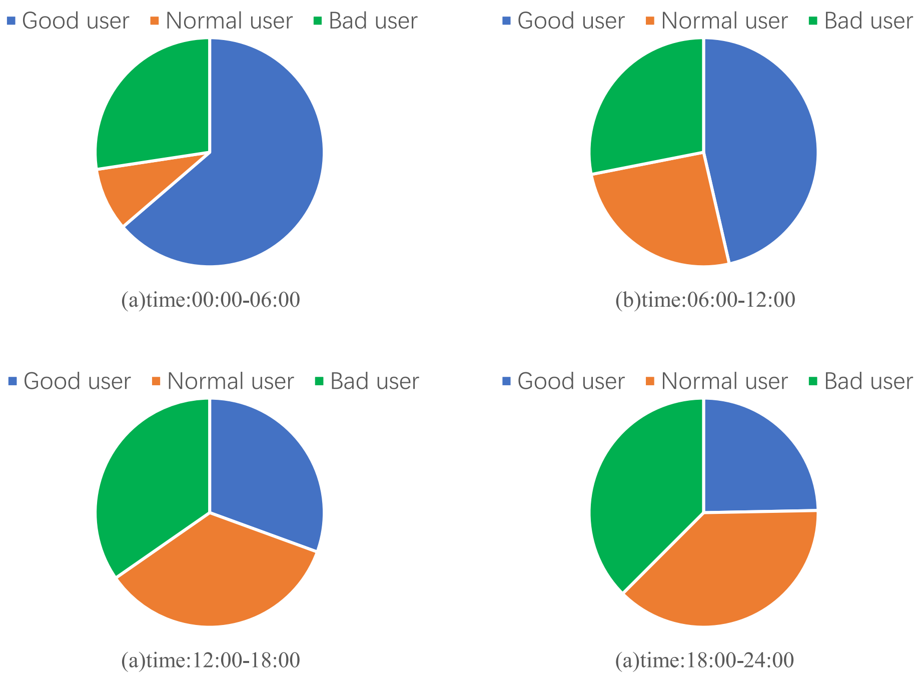

Fig. 3 Proportion of available device in different time slot

System overview

Figure 4 shows workflow and overall architecture of the framework, which follows the synchronization strategy in [36]. The Fl server mainly includes two sub-modules. The client sampling module collects environment information to sample the client with the greatest gain for the current round of the model aggregation algorithm. After determining the sampling client, the dynamic deadline module

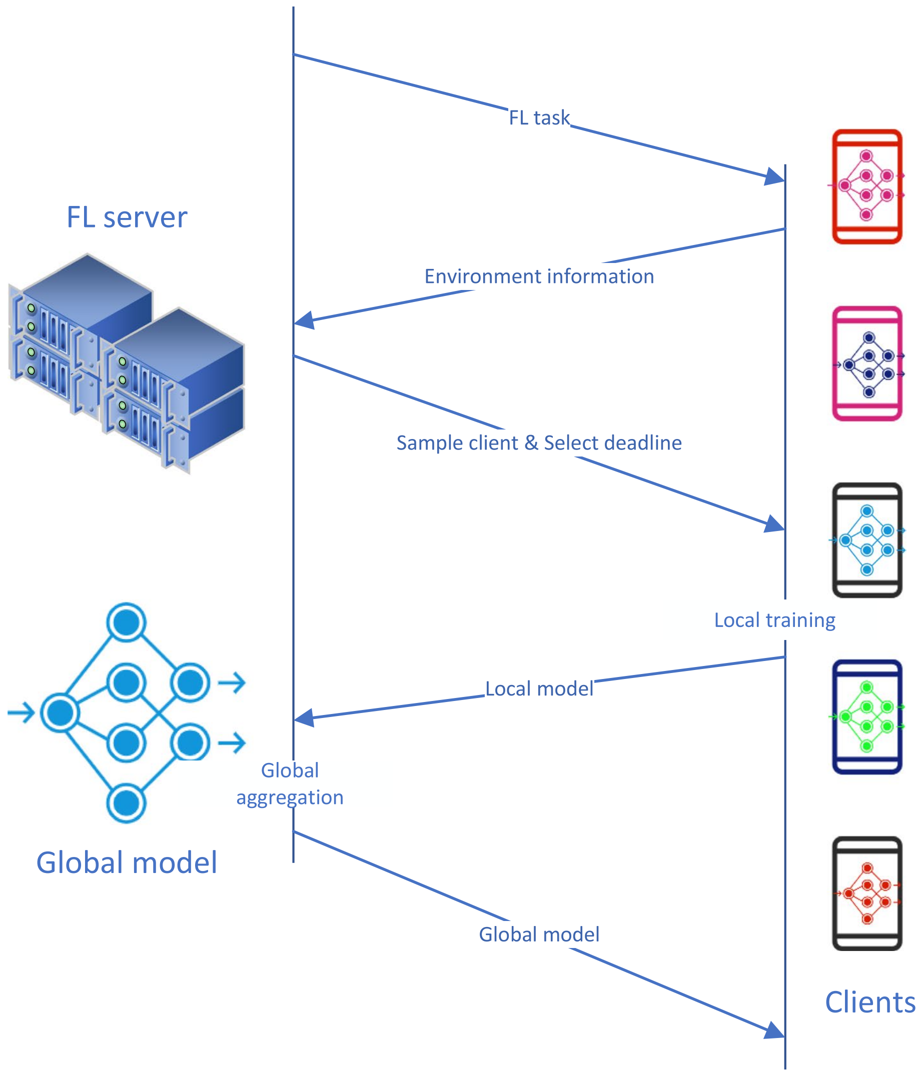

Fig. 4 The federated learning with our strategy

sets the most appropriate deadline according to the heterogeneity information and historical training information of different clients so that each client can upload the local model before the deadline with the maximum probability.

The system workflow of FedState can be expressed as the following main steps.

(1) *Task definition*: After receiving the task, the server confirms all registered clients in the current system and initializes the device information(e.g., phone model, computing power, availability...) and data information(e.g., data size) of all clients.
(2) *Client sampling*: After collecting the environment information returned by the clients, the client sampling module will select several qualified clients from the available clients according to our importance measurement algorithm and distribute the global model to the mobile device.
(3) *deadline controlling*: After determining the subset of sampling devices, the deadline control module will collect the historical training information(e.g., training time, model upload and download time, state heterogeneity) of these devices and determine the final deadline based on these information.
(4) *Local training*: The client will get the available training time for the current round according to the historical information. We adopt the data importance sorting algorithm in Fedbalancer [19] to dynamically adjust the trainable data subset on the mobile device. If the current device terminates the local training or delays uploading the model due to the

state change, it will be deemed as the local training failure.

(5) *Global aggregation*: Fl server will aggregate the models uploaded by clients. The commonly used aggregation algorithms are Fedavg [10], FedProx [37], etc. After aggregation, the server will send the global model parameters back to all participants for the next iteration(return to step2).

(6) *Finish task*: Once the global model reaches the target accuracy on the testset, our training task is deemed to be completed.

Client sampling module

Heterogeneity mainly includes state heterogeneity, device heterogeneity and data heterogeneity, and these three types of heterogeneity affect the time-to-accuracy of FL in different aspects. We need to design a mechanism that can measure the three kinds of heterogeneity and sample the best client based on this information.

The data size and distribution on each client are different on the data heterogeneity level, so the models uploaded by different clients have different gains for the global model. To measure the data utility, we refer to the importance sampling method in machine learning and the calculation method in oort [34]. We define the data utility of client i as:

$$DU(i) = |D_i| \sqrt{\frac{1}{|D_i|}\sum_{k \in D_i} Loss(k)^2} \tag{1}$$

D_i refers to the data on the ith client, Loss(k) refers to the loss value of the kth sample. The higher the loss of the sample k, the worse the performance of the current model in it. In the next round of sampling clients, more clients containing this sample should be sampled. On the device heterogeneity level, due to the existence of mobile devices with various computing capabilities in the FL platform and the difference in the amount of data on different devices, the local training time of the client is also different. In synchronous mode, the training time of each round depends on the device with the longest training time among the sampled clients. Therefore, mobile devices with strong computing power should have higher device utility. When other states are the same, we should give priority to the clients with more effective devices.

For the estimation of computing power of mobile devices, we refer to AI Benchmark [38] and FedScale [39], and get a mapping model that maps the models of mainstream mobile phones to computing power. According to the mapping model, the device utility(CU) of each client is obtained.

In terms of state heterogeneity, due to the different behavior habits of each client, their availability with you varies greatly at different times. For devices that have been registered in the system for a period of time, we can obtain the availability of the user in the future period according to the historical status information of the device. The greater the probability value, the greater the availability. As illustrated in motivation, we have obtained the probability density curves of high-quality users, ordinary users and low-quality users. Therefore, we can get the available probability values of different users at the current time according to the probability density function, which is its state utility(SU).

When designing the sampling algorithm, we should not only consider a certain utility. If only clients with greater data utility are selected, although the model will reach the target accuracy in fewer rounds, each round may take longer. If only clients with higher device utility are selected, the data on the same batch of devices may be sampled all the time, thus affecting round-to-accuracy. If only the state utility is considered, the sampling algorithm may only tend to sample devices with stable state, and the probability of devices with large state changes being sampled is extremely low, which will also have a great impact on the data distribution of the model. Therefore, we need to consider these three influencing factors comprehensively. To sample the best subset of devices, we have designed the utility formula below, sorted the clients according to the utility value, and finally sampled the k clients with the largest utility value.

$$\begin{gathered} U(i) = SU(i)(\mu_1 DU(i) + \mu_2 CU(i)) \\ 0 <= \mu_1 =< 1 \\ 0 <= \mu_2 =< 1 \\ \mu_1 + \mu_2 = 1 \end{gathered} \tag{2}$$

Inspired by the use of reinforcement learning to solve the optimization problem in [18, 20, 40–43], we want to use RL to solve the problem of device sampling and deadline selection.

Favor [20] selects a client with the largest Q value in the training process and the first K devices with the largest Q value in the testing process to avoid the problem of too much action space. But there is a big problem. The training and testing environments are different. During training, the agent will always select a single optimal client, but combining the top k optimal clients is not necessarily optimal. Auction [40] indirectly reduces the complexity of the problem by turning the sampling client into a continuous action. However, the action space is still large, which makes it challenging to train the agent network.

We turn the device sampling in federated learning into deep reinforcement learning(DRL) problems. The DRL agent is trained by the double Deep Q-learning

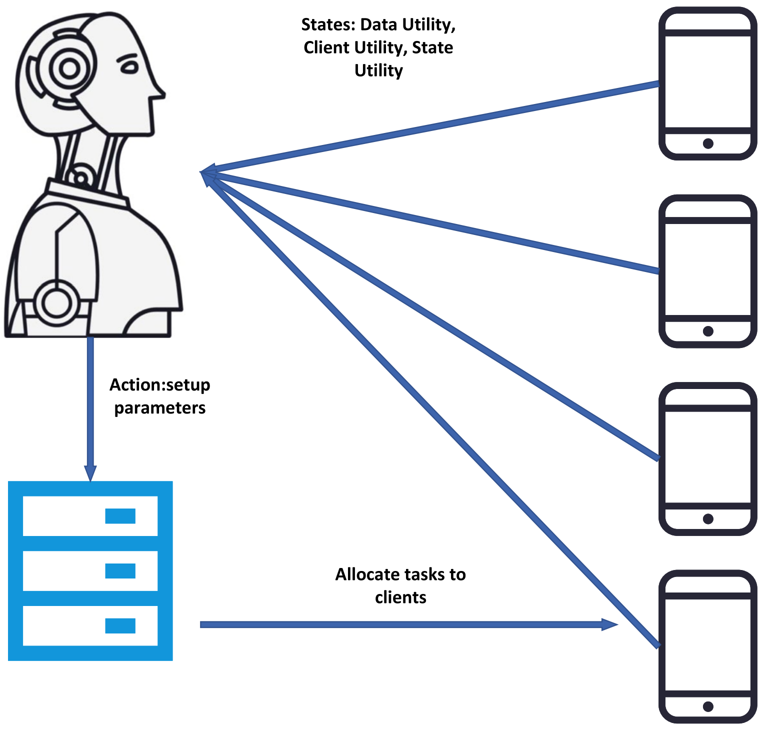
States: Data Utility, Client Utility, State Utility
Action:setup parameters
Allocate tasks to clients

Fig. 5 DRL agent overview

Network(DDQN) [44]. We introduce the design detail as follows.

Sampling agent

(1) *State*: We mentioned in the previous chapter that three different Heterogeneity will have a great impact on the sampling strategy. So our state space is composed of the data utility, device utility and state utility of these clients, which can be defined as $s = \{s_1, s_2, \ldots, s_n\}$. s is a three-dimensional vector expressed as $s_i = \{d_i, e_i, f_i\}$, where d_i is the data utility of client i, e_i is the device utility and f_i is the state utility.

(2) *Action*: If the agent directly selects k clients from N candidate clients, the action space of the sampling algorithm will grow to $O(2^N)$, and as the number of candidate clients increases, the action space will also grow exponentially. In order to avoid this problem, we have designed a criterion to measure the importance of clients in formula 2. We can use this formula to sample the most appropriate clients. Since the sampling algorithm has a different tendency towards data utility and device utility at different training stages, we can adjust the weight of these two factors in the action. And we can define action as the value of μ_1 and μ_2, where μ_1 increases from 0 to 1 with a step size of 0.1. And μ_2 can be obtained by subtracting μ_1 from 1.

(3) *Reward*: When the global model accuracy of round t is greater than that of round t-1, we set the reward value of the round to $r_t = c^{(acc_t - acc_{t-1})}$. On the contrary, we set it to $r_t = -c^{(acc_{t-1} - acc_t)}$. Where c is a normal number to ensure that the reward value increases with the improvement of the accuracy of the global model. The cumulative discounted reward is defined as follows:

$$r_t = \begin{cases} c^{(acc_t - acc_{t-1})} & acc_t > acc_{t-1} \\ -c^{(acc_{t-1} - acc_t)} & acc_t < acc_{t-1} \end{cases} \tag{3}$$

$$R = \sum_{t=1}^{T} \gamma^{t-1} r_t \tag{4}$$

In summary, as shown in the Fig. 5, our DRL agent system observe the utilities as the states and allocate parameter to our central server to select agents.

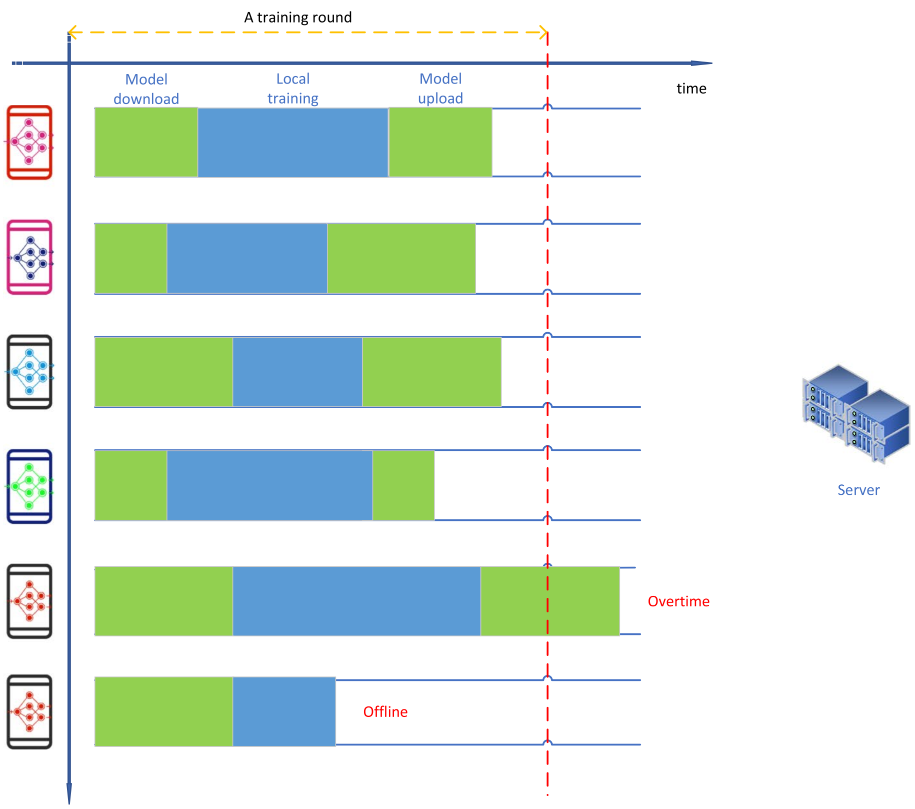

Fig. 6 Local training process

Dynamic deadline module

As shown in Fig. 6, the time of training round i consists of the upload time, download time, and local training time of the model. The upload and download time of the model is affected by the network bandwidth, and the network quality of most mobile devices is always changing. Therefore, we refer to the real-time mobile phone bandwidth measured by Hofft et al. from December 16, 2015, to February 4, 2016, in Ghent, Belgium, and surrounding areas [4]. When our framework simulates the network situation of the client, we will randomly select a curve in the dataset as the user's network bandwidth. So the communication time can be calculated as follows:

$$t_{com,k} = \frac{\xi_{upload} + \xi_{download}}{s_k} \ \forall k \in \{1, ...N\} \tag{5}$$

Where ξ_{upload} is the size of the upload model, $\xi_{download}$ is the size of the download model, and s is the bandwidth of user k.

The local training time of the model is affected by the computing power of the device and the number of client samples. So the training time of user k is defined as:

$$tcomp, k = \frac{n_k e}{c_k} \tag{6}$$

where n_k is the sample size of user k, e is the local training epoch, c_k is the computation power of device k. And we can calculate the total time required for each round of local training.

$$t_{train} = \max\left(tcomp, k + tcom, k\right) k \in \{1 \ldots N\} \tag{7}$$

Through the above three formulas, we can calculate the total time required for the local training of each client. However, due to state heterogeneity and device heterogeneity, each device has different computing power, sample size, and availability probability. Therefore, the local training time of different clients varies greatly. Without considering the state heterogeneity, we can set the deadline according to the suggestion of SmartPC [35] so that 80% of devices can upload their models. However, in a heterogeneous state environment, the length of training time will affect the probability that the client is available. A shorter deadline can prevent most clients from being disconnected due to state changes, but it will also prevent them from having enough time to finish training the local data. Although a more extended deadline can allow most clients to submit the local model, the longer the duration, the greater the probability of the client being disconnected.

For the deadline setting, due to the variability of user status in the system, we also use agents to adjust the deadline of the current round according to the environment status information. The detailed design of the deadline agent is as follows.

Deadline agent

(1) *State*: In a heterogeneous environment, there are three factors that affect the deadline: communication time, training time, and state utility. The communication time can be obtained indirectly according to the historical bandwidth information. So our state space can be defined as $s = \{s_1, s_2, \ldots, s_k\}$, where k represents all clients sampled. S is a three-dimensional vector expressed as $s_i = \{b_i, t_i, u_i\}$, the first dimension is the historical bandwidth information of the client, and the second dimension is its training time. And the third dimension is the state utility of the client.
(2) *Action*: We counted the time consumed by each round of local training and communication of all clients in the current system. After removing the outlier, we obtained the upper and lower limits of the deadline. Finally, we select some column values from 80 to 200 with a step size of 10 as action.
(3) *Reward*: We adjust the deadline to enable more devices to successfully submit the model within the specified time, so as to improve the time-to-accuracy of the global model. The design of reward mechanism refer to the sampling module.

DRL training methodology

Current deep reinforcement learning methods are generally divided into two types: value learning and strategy learning. We train the DRL agent by using the DDQN method(Double Deep Q Network), which well matches our federal learning context, and has been successfully applied in our experiment. The Q-learning algorithm uses $max_a q(s_{t+1}, a)$ to update the action value, which leads to "maximization bias" and makes the estimated action value large. The DDQN algorithm is identical to the DQN algorithm in all other aspects. Double Q-learning requires the construction of two action value functions, one for estimating the action and the other for estimating the value of that action. However, considering that two networks, the evaluation network, and the target network, are already available in the DQN algorithm, the DDQN algorithm only needs to use the evaluation network to determine the action and the target network to determine the action value when estimating the return, without constructing a new network separately. The DRL agent training procedure starts with randomly initializing the parameters of the qeval network and qnext network, and using the MSELoss as the loss function. Before the DRL agent training of each episode, we load the real-world dataset and mobile devices' information, this will construct a simulated FL training environment, and each training episode is a simulation independent of the other. the FL system starts to train the model round by round. At the start of each round, the DRL collect system profile states such as the reward, bandwidth, and state utility of each client, into a vector v_{states}. Then we use the $v_{states-t}$ as the input of qeval network, and then get the output q_{pred} (used to sample the clients) . The q_{pred} is a one dimension vector, which contains the q value of each action. Then we select the action which has the max q value. At the end of each round, after clients finished training, the aggregator calls an evaluation event and calculates the accuracy of the current model after this round of training. In the meantime, we get the system profile after training as the $v_{states-t+1}$, and use it as the input of qnext, and get the output q_{next}. And then we use the reward R_t to calculate the q of this round:

$$q_{target} = R_t + \gamma * q_{next} \tag{8}$$

With q_{target} and q_{pred}, we can use the MSELoss to calculate the loss, and update the qeval. Every $C_{replace}$ round, the DRL agent will update the qnext network using the current qeval network.

```
Require: Accuracy requirement $\Lambda$, number of clients $N$, DRL agent
    $agent$, action label list $L$, over sample factor $osf$.
1:  Initialize $agemt$
2:  for episode in range($num\_episodes$) do
3:      Initialize global model
4:      r ← 0
5:      while current accuracy $<$ $\Lambda$ do
6:          $r \leftarrow r + 1$
7:          $data\_utility \leftarrow get\_data(client\_states)$
8:          $device\_utility \leftarrow get\_device(client\_states)$
9:          $state\_utility \leftarrow get\_state(model, client\_states)$
10:         Agent decide action
11:         $agent \leftarrow data\_utility, device\_utility, state\_utility$
12:         $data\_factor, device\_factor = agent(L)$
13:         Choose clients and allocate task
14:         $weighted\_utility \leftarrow state_{utility} * (data_{utility} * data_factor + device_{utility} * device_{factor})$
15:         $sorted\_clients \leftarrow sort\_clients(clients, weighted\_utility, target\_num * osf)$
16:         $agent \leftarrow globalmodel$
17:         Evaluate the reward and update agent
18:         $create\_evaluation\_tasks(clients)$
19:         $new\_accuracy \leftarrow evaluate\_model(evaluation\_tasks, model)$
20:         $reward \leftarrow calculate\_reward(previous\_acc, new\_acc)$
21:         $agent \leftarrow reward$
22:         $agent\ update$
23:     end while
24:     $X \leftarrow L[\arg\max\ (scores[1...N_L])]$
25:     return $X$
26: end for
```

Algorithm 1 FedState online training methodology

```
Require: Replay memory $\mathcal{D}$, network parameters $\theta$, target network
    parameters $\theta^-$, learning rate $\alpha$, discount factor $\gamma$, batch size
    $N$, replay start size $N_0$
1: if $|\mathcal{D}| > N_0$ then
2:     Sample a random minibatch of transitions
       $(s_j, a_j, r_{j+1}, s_{j+1})$ from the replay memory $\mathcal{D}$
3:     Calculate the target value for each transition: $y_j = r_{j+1} +
       \gamma Q(s_{j+1}, \arg\max_{a'} Q(s_{j+1}, a', \theta), \theta^-)$
4:     Update the network parameters using gradient descent: $\theta \leftarrow
       \theta - \alpha \nabla_\theta \frac{1}{N} \sum_{j=1}^{N} (Q(s_j, a_j, \theta) - y_j)^2$
5: end if
```

Algorithm 2 DDQN Network Update

Performance evaluation

In this section, we compare our method with other state-of-art methods. First, we will describe the experimental settings and then analyze the experimental results.

Experimental setting

We use FedScale [39] as our experimental platform and use the probability density functions of different types of users obtained in the previous chapter to construct user trajectories of different proportions as the real-time status of all users. We use a real-world 4G/LTE network [4] to build our simulation environment for the simulation of network bandwidth. This dataset is collected through Huawei P8 lite, and the scenarios cover walking, bicycles, buses, trams, trains and cars. We selected the bandwidth trace for three scenarios as shown in Fig. 7. We will randomly initialize a bandwidth trace for each simulated client in the experiment. In addition, to select a more accurate network traffic prediction model, we conducted experiments to evaluate machine learning models including LSTM, SVM, RandomForest and Autoencoder for network traffic prediction.To evaluate the performance of each model. We summarized the performance of each model in Table 1. This indicates that LSTM performed better than the other models in predicting network traffic. So we decided to use lstm as our network traffic prediction model.

Datasets and models: We tested our algorithm and other algorithms on several different datasets using different CNN models.

- **FMNIST**. We use the resnet18 model. The total number of samples on each client is 300, and the local training round is 5.

Performance metrics: Fewer communication rounds and shorter local training time are our optimization goals, so in the experiment, we use time-to-accuracy to measure the effectiveness of different algorithms.

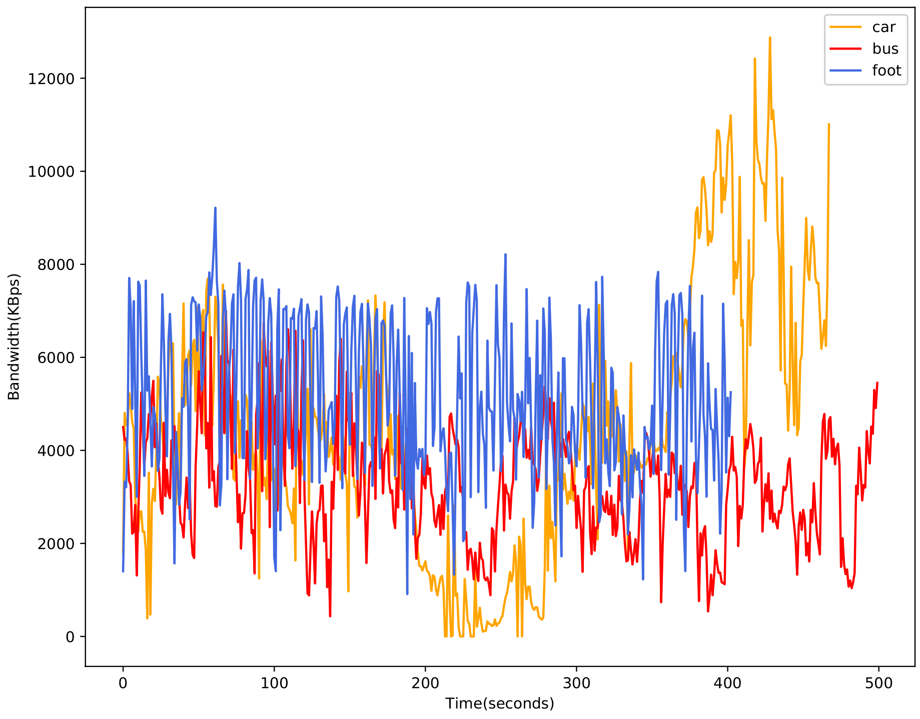

Fig. 7 Network bandwidth of different user

Table 1 Performance Comparison of Network Traffic Prediction Models

Model	MAE	MSE	MAPE
LSTM	0.5837	0.0032	0.0032
SVM	1.9859	7.2680	0.7444
RandomForest	1.6372	4.6821	0.6720
Autoencoder	3.0674	13.7687	0.704

Evaluation results

As shown in the method, we have designed sampling and dynamic deadline strategies. We have conducted comparative experiments between these two methods and the current state-of-art method, and the results are as follows.

The agent training process

Figure 8 shows the training process of the client selection agent and dynamic deadline agent on fmnist, where the number of candidate clients is fixed at 200, and the number of clients sampled in each round is fixed at 20. The experimental setup follows the description in the previous section. The reward refers to the cumulative discount reward of an entire episode. We can observe that after training 200 episodes, our reward can be stabilized at a relatively high value, which indicates that our agents can learn how to sample the best devices and set an appropriate deadline.

Performance comparison

We compare our sampling and dynamic deadline algorithms with the current state-of-art method. For user status, we simulated three scenarios with different ratios, in which the ratios of the high, medium and low-quality users are 1:7:2, 3:4:3, and 1:5:4, respectively.

Figure 9 shows the performance of our sampling algorithm and oort in fmnist. We can observe that our sampling algorithm is superior to other benchmarks for the three scenarios. In Fig. 9a, we can find that in this case, our sampling algorithm is obviously superior to oort. The larger the proportion of ordinary users, the greater the heterogeneity of the current system state. It can be seen that our sampling algorithm performs better when the state heterogeneity is greater. In the other two scenarios, due to the large proportion of high-quality users and low-quality users, and the small change in the state of these two users, the gain of our method relative to oort is not particularly high. Other algorithms only consider the device utility and data utility when sampling devices. If a device often fails to submit the model successfully, the probability of sampling the device will be reduced subsequently. This also leads to that the existing sampling algorithm may always sample the same batch of devices with stable state in the environment with heterogeneous states, this also affects the time-to-accuracy of the global model.

Figure 10 shows the comparison between our dynamic deadline algorithm and FedBalancer. From the figure, we can see that our algorithm has improved to a certain

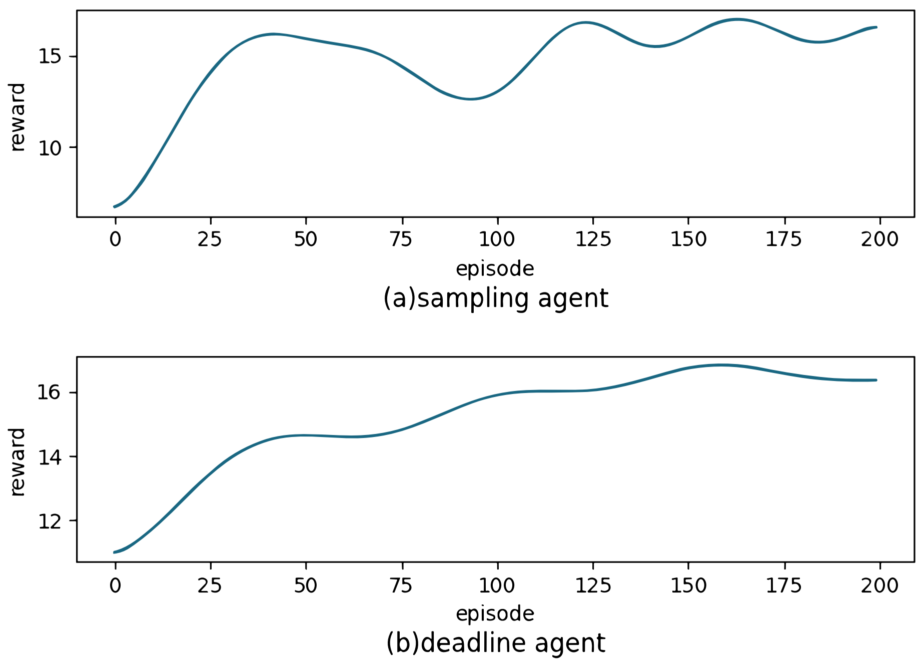

Fig. 8 The training process of the agents

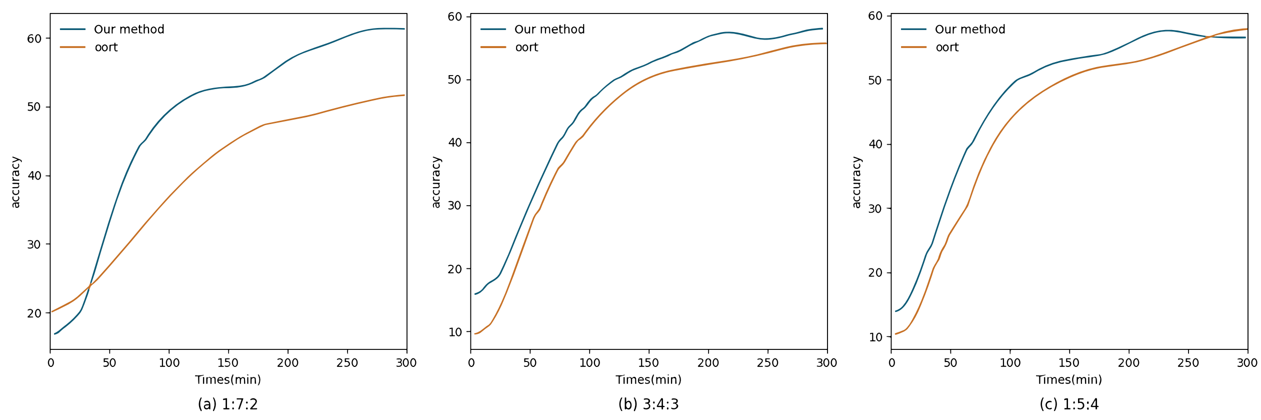

Fig. 9 Performance comparison of sampling algorithm

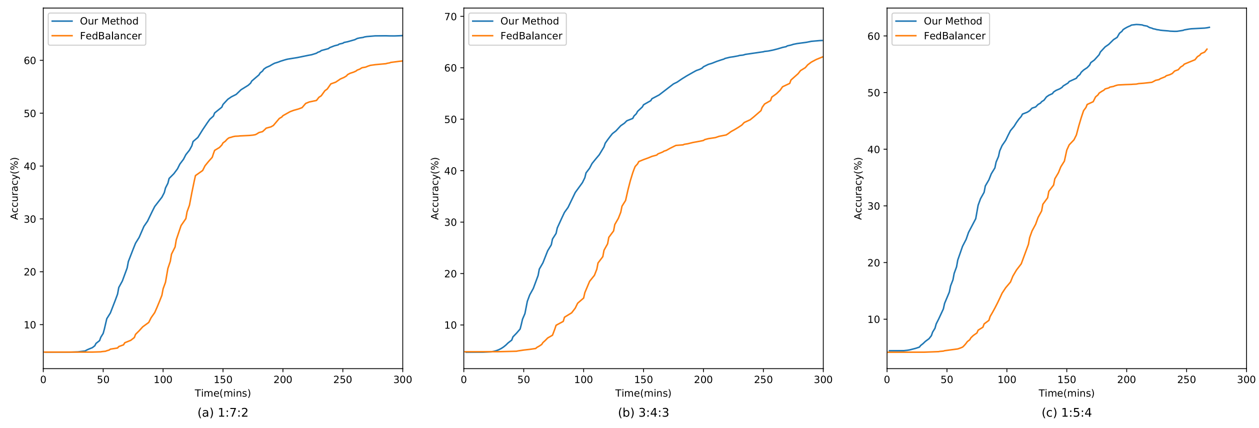

Fig. 10 Performance comparison of dynamic deadline algorithm

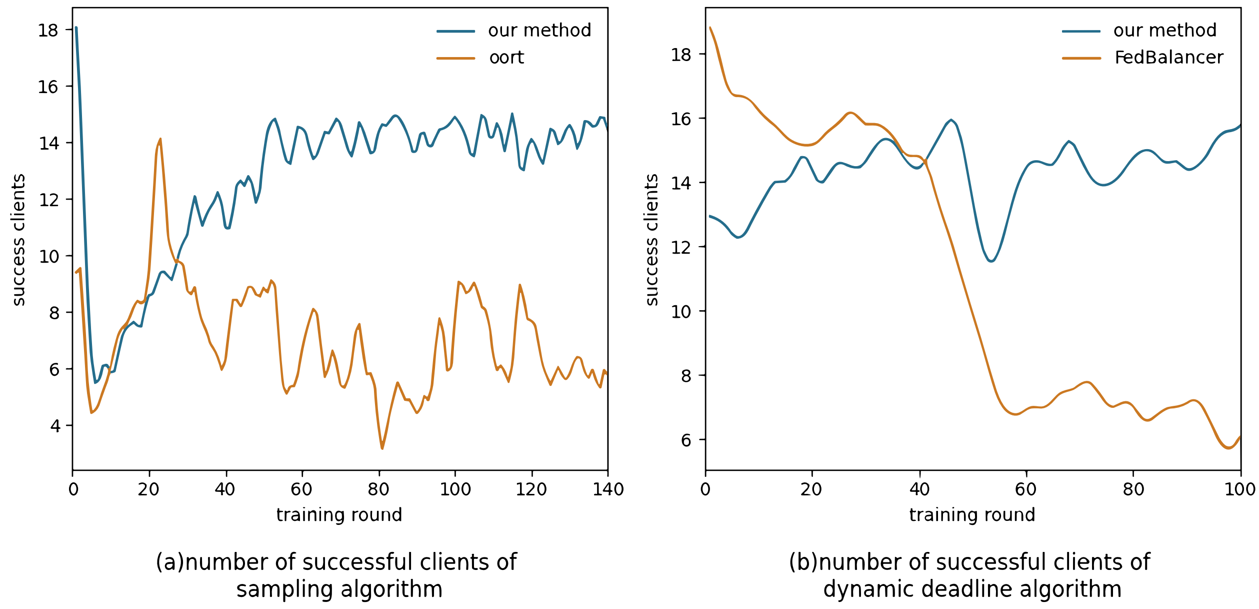

Fig. 11 Comparison of the number of local clients that successfully submitted models during training

extent compared with FedBalancer in different scenarios. The proportion of different types of users has no great impact on the effect of our algorithm. We can achieve higher accuracy at the same time. Fedbalancer does not consider the user's state heterogeneity when adjusting the deadline, so its adjustment strategy will affect the real-time state of users in the system, thus affecting the total number of devices that successfully uploaded the model.

Figure 11 shows the number of models successfully submitted by our algorithm and other algorithms in each round in a heterogeneous environment. From the figure, we can see that our two algorithms can increase the probability that clients can successfully upload the model. However, in different training stages, a large part of the clients of oort and FedBalancer cannot successfully upload the local model, which affects the accuracy of the global model.

Conclusion

Due to the influence of state heterogeneity, traditional algorithms can not run efficiently in heterogeneous environments. In this paper, we propose a federated learning framework that considers time-to-accuracy in heterogeneous environments. We designed an experience-driven method based on DRL to dynamically sample devices according to the current system state information and adjust the deadline. The final control experiment further proves the superiority of our method compared with the most advanced solution.

Authors' contributions

Deyu Zhang and Shiwen He wrote the main manuscript test. Wang Sun and Zi-Ang Zheng set up the evaluation and run the experiments. Wenxin Chen prepared the figures. All authors reviewed the manuscript.

Funding

This work was supported in part by the National Key R &D Program of China (2022YFF0604504), National Science Foundation of China (62172439, 62171474, 62177047), Major Project of Natural Science Foundation of Hunan Province (2021JC0004), National Natural Science Foundation of Hunan Province (2023JJ20076), Central South University Research Programme of Advanced Interdisciplinary Studies (No.2023QYJC020), and Central South University InnovationDriven Research Programme (2023CXQD061).

Availability of data and materials

Some or all data, models, or codes that support the findings of this study are available from the corresponding author upon resonable request.

Declarations

Ethics approval and consent to participate

Not applicable.

Competing interests

The authors declare no competing interests.

Received: 30 December 2022 Accepted: 4 September 2023
Published online: 03 November 2023

References

1. Chen Z, Cao Y, Liu Y, Wang H, Xie T, Liu X (2020) A comprehensive study on challenges in deploying deep learning based software. ESEC/FSE 2020. Association for Computing Machinery, New York, p 750–762. https://doi.org/10.1145/3368089.3409759
2. Chen Y, Xing H, Ma Z, Chen X, Huang J (2022) Cost-efficient edge caching for noma-enabled IoT services. Chin Commun. https://doi.org/10.1155/2022/8072493
3. Huang J, Gao H, Wan S et al (2023) Aoi-aware energy control and computation offloading for industrial IoT. Futur Gener Comput Syst 139:29–37
4. Van Der Hooft J, Petrangeli S, Wauters T, Huysegems R, Alface PR, Bostoen T, De Turck F (2016) Http/2-based adaptive streaming of HEVC video over 4g/LTE networks. IEEE Commun Lett 20(11):2177–2180
5. Shen X, Gao J, Wu W, Li M, Zhou C, Zhuang W, (1st. Quart. (2022) Holistic network virtualization and pervasive network intelligence for 6G. IEEE Commun Surv Tuts 24(1):1–30
6. Wu W, Zhou C, Li M, Wu H, Zhou H, Zhang N, Xuemin S, Zhuang W (Feb. 2022,) AI-native network slicing for 6G networks. IEEE Wirel Commun 29(1):96–103
7. Xu X, Duan S, Zhang J, Luo Y, Zhang D (2021) Optimizing federated learning on device heterogeneity with a sampling strategy. In: 2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS), IEEE, pp 1–10
8. CHEN Y, HU J, ZHAO J, MIN G (2023) Qos-aware computation offloading in leo satellite edge computing for iot: A game-theoretical approach. Chin J Electron 33:1–12
9. Chen Y, Zhao J, Zhou X, Qi L, Xu X, Huang J (2023) A distributed game theoretical approach for credibility-guaranteed multimedia data offloading in mec. Inf Sci 119306
10. McMahan B, Moore E, Ramage D, Hampson S, y Arcas BA (2017) Communication-efficient learning of deep networks from decentralized data. In: Artificial intelligence and statistics, PMLR, pp 1273–1282
11. Chen Y, Zhao J, Hu J, Wan S, Huang J (2023) Distributed task offloading and resource purchasing in noma-enabled mobile edge computing: Hierarchical game theoretical approaches. ACM Trans Embed Comput Syst. https://doi.org/10.1145/3597023. Just Accepted
12. Chen Y, Zhao J, Wu Y et al (2022) Qoe-aware decentralized task offloading and resource allocation for end-edge-cloud systems: A game-theoretical approach. IEEE Trans Mob Comput. https://doi.org/10.1109/TMC.2022.3223119
13. Mell P, Grance T et al (2011) The nist definition of cloud computing
14. Li T, Sahu AK, Talwalkar A, Smith V (2020) Federated learning: Challenges, methods, and future directions. IEEE Signal Proc Mag 37(3):50–60
15. Wang J, Liu Q, Liang H, Joshi G, Poor HV (2020) Tackling the objective inconsistency problem in heterogeneous federated optimization. Adv Neural Inf Process Syst 33:7611–7623
16. Luo B, Li X, Wang S, Huang J, Tassiulas L (2021) Cost-effective federated learning design. In: IEEE INFOCOM 2021-IEEE Conference on Computer Communications, IEEE, pp 1–10
17. 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. Proc Mach Learn Syst 1:374–388
18. Zhan Y, Li P, Guo S (2020) Experience-driven computational resource allocation of federated learning by deep reinforcement learning. In: 2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS), IEEE, pp 234–243
19. Shin J, Li Y, Liu Y, Lee SJ (2022) Fedbalancer: Data and pace control for efficient federated learning on heterogeneous clients. In: Proceedings of the 20th Annual International Conference on Mobile Systems, Applications and Services. MobiSys '22, Association for Computing Machinery, New York, p 436–449. https://doi.org/10.1145/3498361.3538917
20. Wang H, Kaplan Z, Niu D, Li B (2020) Optimizing federated learning on non-iid data with reinforcement learning. In: IEEE INFOCOM 2020-IEEE Conference on Computer Communications, IEEE, pp 1698–1707
21. Yang C, Wang Q, Xu M, Chen Z, Bian K, Liu Y, Liu X (2021) Characterizing impacts of heterogeneity in federated learning upon large-scale smartphone data. Proceedings of the Web Conference 2021:935–946
22. Li Z, Zhang J, Liu L, Liu J (2022) Auditing privacy defenses in federated learning via generative gradient leakage. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, p 10132–10142

23. Gong X, Sharma A, Karanam S, Wu Z, Chen T, Doermann D, Innanje A (2021) Ensemble attention distillation for privacy-preserving federated learning. In: Proceedings of the IEEE/CVF International Conference on Computer Vision, p 15076–15086
24. Wang J, Liu Q, Liang H, Joshi G, Poor HV (2021) A novel framework for the analysis and design of heterogeneous federated learning. IEEE Trans Signal Process 69:5234–5249
25. Palihawadana C, Wiratunga N, Wijekoon A, Kalutarage H (2022) Fedsim: Similarity guided model aggregation for federated learning. Neurocomputing 483:432–445
26. Chen HY, Chao WL (2020) Fedbe: Making bayesian model ensemble applicable to federated learning. arXiv preprint arXiv:2009.01974
27. Liu Z, Chen Y, Yu H, Liu Y, Cui L (2022) Gtg-shapley: Efficient and accurate participant contribution evaluation in federated learning. ACM Trans Intell Syst Technol (TIST) 13(4):1–21
28. Wang T, Rausch J, Zhang C, Jia R, Song D (2020) A principled approach to data valuation for federated learning. In: Federated Learning, Springer, pp 153–167
29. Paulik M, Seigel M, Mason H, Telaar D, Kluivers J, van Dalen R, Lau CW, Carlson L, Granqvist F, Vandevelde C, et al (2021) Federated evaluation and tuning for on-device personalization: System design & applications. arXiv preprint arXiv:2102.08503
30. Wang S, Lee M, Hosseinalipour S, Morabito R, Chiang M, Brinton CG (2021) Device sampling for heterogeneous federated learning: Theory, algorithms, and implementation. In: IEEE INFOCOM 2021-IEEE Conference on Computer Communications, IEEE, pp 1–10
31. Balakrishnan R, Li T, Zhou T, Himayat N, Smith V, Bilmes J (2022) Diverse client selection for federated learning via submodular maximization. In: International Conference on Learning Representations. https://openreview.net/forum?id=nwKXyFvaUm
32. Zhang SQ, Lin J, Zhang Q (2022) A multi-agent reinforcement learning approach for efficient client selection in federated learning. arXiv preprint arXiv:2201.02932
33. Cho YJ, Wang J, Joshi G (2022) Towards understanding biased client selection in federated learning. In: International Conference on Artificial Intelligence and Statistics, PMLR, pp 10351–10375
34. Lai F, Zhu X, Madhyastha H, Chowdhury M (2021) Oort: Efficient federated learning via guided participant selection. In: Proceedings of the 15th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2021, p 19–35
35. Li L, Xiong H, Guo Z, Wang J, Xu CZ (2019) Smartpc: Hierarchical pace control in real-time federated learning system. In: 2019 IEEE Real-Time Systems Symposium (RTSS), IEEE, pp 406–418
36. Konečnỳ J, McMahan HB, Yu FX, Richtárik P, Suresh AT, Bacon D (2016) Federated learning: Strategies for improving communication efficiency. arXiv preprint arXiv:1610.05492
37. Li T, Sahu AK, Zaheer M, Sanjabi M, Talwalkar A, Smith V (2020) Federated optimization in heterogeneous networks. Proc Mach Learn Syst 2:429–450
38. Ignatov A, Timofte R, Chou W, Wang K, Wu M, Hartley T, Van Gool L (2018) Ai benchmark: Running deep neural networks on android smartphones. In: Proceedings of the European Conference on Computer Vision (ECCV) Workshops
39. Lai F, Dai Y, Singapuram S, Liu J, Zhu X, Madhyastha H, Chowdhury M (2022) FedScale: Benchmarking model and system performance of federated learning at scale. In: Chaudhuri K, Jegelka S, Song L, Szepesvari C, Niu G, Sabato S (eds.) Proceedings of the 39th International Conference on Machine Learning, vol. 162. Proceedings of Machine Learning Research, p 11814–11827
40. Deng Y, Lyu F, Ren J, Wu H, Zhou Y, Zhang Y, Shen X (2021) Auction: Automated and quality-aware client selection framework for efficient federated learning. IEEE Trans Parallel Distrib Syst 33(8):1996–2009
41. Chen Y, Gu W, Xu J, Zhang Y, Min G (2023) Dynamic task offloading for digital twin-empowered mobile edge computing via deep reinforcement learning. Chin Commun 1–12 https://doi.org/10.23919/JCC.ea.2022-0372.202302
42. Huang J, Wan J, Lv B, Ye Q et al (2023) Joint computation offloading and resource allocation for edge-cloud collaboration in internet of vehicles via deep reinforcement learning. IEEE Syst J. https://doi.org/10.1109/JSYST.2023.3249217
43. Wu W, Chen N, Zhou C, Li M, Shen X, Zhuang W, Li X (2021) Dynamic RAN slicing for service-oriented vehicular networks via constrained learning. IEEE J Sel Areas Commun 39(7):2076–2089
44. Van Hasselt H, Guez A, Silver D (2016) Deep reinforcement learning with double q-learning. In: Proceedings of the AAAI conference on artificial intelligence, vol 30. https://doi.org/10.1609/aaai.v30i1.10295. https://ojs.aaai.org/index.php/AAAI/article/view/1029

Publisher's Note