

A Multi-Agent Reinforcement Learning Approach for Efficient Client Selection in Federated Learning

Sai Qian Zhang¹, Jieyu Lin², Qi Zhang³

¹Harvard University, ²University of Toronto, ³Microsoft

Abstract

Federated learning (FL) is a training technique that enables client devices to jointly learn a shared model by aggregating locally computed models without exposing their raw data. While most of the existing work focuses on improving the FL model accuracy, in this paper, we focus on the improving the training efficiency, which is often a hurdle for adopting FL in real world applications. Specifically, we design an efficient FL framework which jointly optimizes model accuracy, processing latency and communication efficiency, all of which are primary design considerations for real implementation of FL. Inspired by the recent success of Multi Agent Reinforcement Learning (MARL) in solving complex control problems, we present FedMarl, a federated learning framework that relies on trained MARL agents to perform efficient client selection. Experiments show that FedMarl can significantly improve model accuracy with much lower processing latency and communication cost.

Introduction

The rapid adoption of Internet of Things (IoT) in recent years has resulted in tremendous growth of data (e.g., text, image, audio) generated on client devices. With the success of deep neural networks (DNNs), there is a growing demand for efficiently training DNN models using the massive volume of data generated from client devices. The traditional centralized approach of gathering and training with all data at a central location not only raises scalability challenges, but also exposes data privacy concerns. To address these limitations, Federated Learning (FL) (McMahan et al. 2017) has been recently proposed to allow client devices to jointly learn a shared model by aggregating their local DNN weight updates without exposing their raw data. At the beginning of each training round, a central server selects a subset of client devices to receive the current global model. Next, each client device performs DNN training using its local dataset and sends its weight update to the central server. The central server then applies the weight updates received from client devices to the global model, which will be used as the initial model for the next training round. Unlike the centralized training scheme, FL naturally addresses the scalability and privacy issues, and hence is more applicable for real implementation.

Despite its advantages, training DNNs using FL still faces several challenges. First, the underlying statistical heterogeneity leads to non-independent, identically distributed (non-IID) training data, which can severely degrade FL convergence behavior (Zhao et al. 2018). Second, from the implementation perspective, the heterogeneous computing and networking resources on client devices can lead to the stragglers that will significantly slow down the FL training process. This problem is further exacerbated by the uneven distribution of client data size, as a client with more data generally incurs higher training latency. Lastly, FL training process often causes high communication cost due to the iterative model updates between client devices and the central server. From the practical perspective, an efficient FL framework that can mitigate all the three problems above is of paramount importance. While numerous solutions have been proposed for accelerating the FL convergence speed (Karimireddy et al. 2019; Li et al. 2020; Wang et al. 2020a; Yang, Fang, and Liu 2021), minimizing the processing latency (Wang, Wei, and Zhou 2020; Nishio and Yonetani 2019; Dhakal et al. 2019; Wang et al. 2020b), or reducing the communication overhead (Konečný et al. 2016; Singh et al. 2019; Luping, Wei, and Bo 2019; Sattler et al. 2019), these heuristic solutions do not tackle these problems jointly. Worse still, optimizing one of the objectives might deteriorate the others. For instance, Scaffold (Karimireddy et al. 2019) enables a better convergence behavior by computing additional model gradients, resulting in a large training latency.

To address this limitation, in this paper, we present a novel FL framework which jointly alleviates the three problems above, namely model accuracy, processing latency and communication overhead. Designing such an optimal client selection policy is challenging. This is due to the intractable convergence behavior of the FL training with the non-IID client data, the dynamic system performance on client devices and the complicated interaction between these objectives. More importantly, the significance of each optimization goal also varies across different scenarios. For example, communication cost may be the primary target in scenarios where network connection is costly, whereas minimizing total processing time may become the key objective for a delay-sensitive FL application. Therefore, the proposed solution must balance and offset different design goals. Developing such a hand-tuned heuristic solution for each combination of

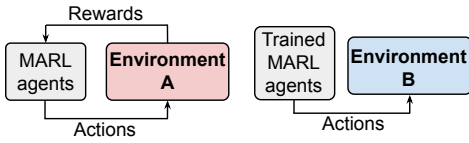


Figure 1: The MARL agents trained under one FL setting (left) can perform well under different FL settings (right).

these goals requires substantial time and effort.

In this work, we leverage the recent advances in Multi-Agent Reinforcement Learning (MARL), and propose *FedMarl*, an efficient FL framework that performs optimal client selection at run time. Although it is also possible to solve this problem with conventional reinforcement learning approach (i.e. single-agent RL), the high dimensionality on the action space for the RL agent will seriously degrade the convergence speed of RL training, leading to a suboptimal performance. FedMarl exploits the device performance statistics and training behaviors to produce efficient client selection decisions based on the designated objective imposed by the application designers. To closely simulate the real FL system operations, we perform extensive experiments to collect the real traces on DNN training latencies and transmission latencies over multiple mobile devices. We further build an MARL environment using the collected traces for the training of the MARL agents. Compared with the other algorithms, FedMarl achieves the best model accuracy while greatly reducing the processing latency and communication cost by $1.7\times$ and $2.2\times$, respectively. Furthermore, the evaluation results show that the superior performance of the MARL agents can translate to different FL settings without retraining (Figure 1), which significantly broadens the applicability of the FedMarl. A theoretical analysis is provided in the extended version of this work (Zhang, Lin, and Zhang 2022).

Background and Related Work

Federated Learning

The first FL framework, Federated Averaging (FedAvg) (McMahan et al. 2017), uses a central server to communicate with clients for decentralized training. Specifically, given a total of K client devices, during each training round t , a fixed number of N ($N \leq K$) devices are randomly picked by the central server. Each selected client n ($1 \leq n \leq N$) contains local training data of size D_n . During the FL operation, each client device receives a copy of the global DNN model W_{glb}^t from the central server, performs E epochs of local training using local surrogate of the global objective function $F_n(\cdot)$, and transmit the weight updates ΔW_n^t back to the central server. The central server then computes the weighted average $\frac{\sum_{n \in N} D_n \Delta W_n^t}{\sum_{n \in N} D_n}$ and applies the change to W_{glb}^t , generating the global weight W_{glb}^{t+1} for the next training round. Although FedAvg achieves superior performance on homogeneous clients with IID data, its performance degrades when data distribution among client devices is non-IID (Zhao et al. 2018; Wang et al. 2020a). While multiple FL frameworks were proposed to mitigate the impact of statistical diversity

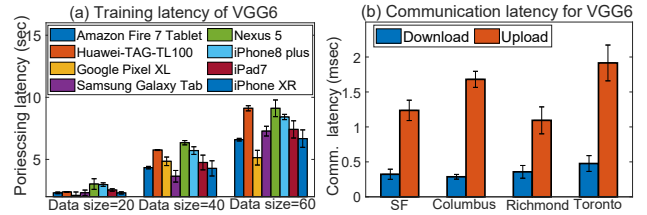


Figure 2: (a) Processing and (b) communication latencies for training VGG6 on CIFAR-10. SF in (b) means San Francisco.

by either controlling the divergence between the local model and global model (Li et al. 2020; Karimireddy et al. 2019), or by training a personalized model for each client (Smith et al. 2017; Deng, Kamani, and Mahdavi 2020; Zhang et al. 2020), none of these studies has considered system performance objectives in their design.

To achieve optimal model performance with minimized processing latency, in Wang, Wei, and Zhou (2020), the authors proposed a FL framework to dynamically control the optimal partitioning of the client data. In Nishio and Yonetani (2019), the authors propose to mitigate the impact of stragglers by assigning higher priorities to faster clients devices. Besides latency reduction, multiple solutions have been proposed for improving FL communication efficiency. Most of these studies reduce the communication cost by applying filtering algorithms or quantization techniques to eliminate the unimportant weight updates (Konečný et al. 2016; Luping, Wei, and Bo 2019; Lai et al. 2020; Fraboni et al. 2021; Sattler et al. 2019; Reisizadeh et al. 2020), or compressing the weight updates via sketching (Rothchild et al. 2020). While these studies have achieved significant performance improvement on their corresponding objectives, none of them can jointly optimize all the three objectives.

Multi-Agent Reinforcement Learning

In cooperative MARL, a set of N agents are trained to produce the optimal actions that lead to the maximum team reward. Specifically, at each timestamp t , each agent n ($1 \leq n \leq N$) observes its state s_n^t and selects an action a_n^t based on s_n^t . After all the agents complete their actions, the team receives a joint reward r_t and proceeds to the next state s_n^{t+1} . The goal is to maximize the total expected discounted reward $R = \sum_{t=1}^T \gamma^t r_t$ by selecting the optimal agent actions, where $\gamma \in [0, 1]$ is the discount factor. Recently, Value Decomposition Network (VDN) (Sunehag et al. 2017) has become a promising solution for jointly training agents in cooperative MARL (Jiang and Lu 2018; Das et al. 2019; Zhang, Zhang, and Lin 2019, 2020). In VDN, each agent n uses a DNN to infer its action. This DNN implements the Q-function $Q_n^\theta(s, a) = E[R_t | s_n^t = s, a_n^t = a]$, where θ is a parameter of the DNN and $R_t = \sum_{i=t}^T \gamma^i r_i$ is the total discounted team reward received at t . During MARL execution, every agent n selects the action a^* with the maximum Q-value (i.e., $a^* = \arg \max_a Q_n^\theta(s_n^t, a)$). To train the VDN, a replay buffer is used to save the transition tuples $\langle s_n^t, a_n^t, s_n^{t+1}, r_t \rangle$ for each agent n . A joint Q-

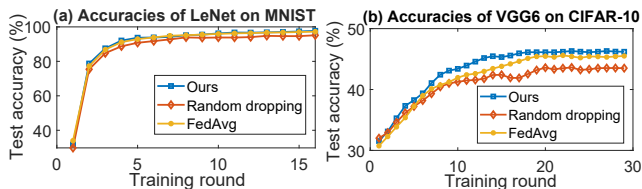


Figure 3: Test accuracies of the three algorithms on (a) LeNet and (b) VGG6.

function $Q_{tot}(\cdot)$ is represented as the elementwise summation of all the individual Q-functions (i.e., $Q_{tot}(\mathbf{s}_t, \mathbf{a}_t) = \sum_n Q_n^\theta(s_n^t, a_n^t)$), where $\mathbf{s}_t = \{s_n^t\}$ and $\mathbf{a}_t = \{a_n^t\}$ are the states and actions collected from all agents $n \in N$ at timestep t . The agent DNNs can be trained recursively by minimizing the loss $L = E_{s_t, \mathbf{a}_t, r_t, s_{t+1}} [y_t - Q_{tot}(\mathbf{s}_t, \mathbf{a}_t)]^2$, where $y_t = r_t + \gamma \sum_n \max_a Q_n^{\theta'}(s_n^{t+1}, a)$ and θ' represents the parameters of the *target network*, which are copied periodically from θ during the training phase.

Motivation and Problem Definition

FL Processing Time Breakdown

To better understand the composition of the FL processing time, we trace the processing time for training VGG6 (Simonyan and Zisserman 2014) by using CIFAR-10 dataset. To simulate resource heterogeneity, we use 8 different client devices, including iPhone8, iPhone XR, iPad2, Huawei-TAG-TL100, Google Pixel XL, Nexus 5, Samsung Galaxy Tab A 8.0 and Amazon Fire 7 Tablet. For each device, we measure the time for performing 5 epochs of training with different data size ranging from 20 to 60 samples. We developed our own DNN training implementation on the mobile devices using the previous literature (TFT 2019; Cor 2020). For the Android devices, the DNN models are first built with Keras and then converted to TensorFlow Lite format before running on the client device. For iOS devices, we first build the DNN models using PyTorch and convert the resulting model to CoreML format with coremltools. We then perform the measurement 500 times and record the average latency. The measurement are shown in Figure 2 (a). To measure the communication latency between the central server and client devices, we use a single-core Amazon EC2 *p3.2xlarge* instance located at Portland (OR) to simulate the cloud server, and four virtual machines (VMs) located at Richmond (VA), San Francisco (CA), Columbus (OH), and Toronto (ON) to simulate the client devices. Figure 2 (b) shows the average latency for uploading and downloading VGG6 models over 500 measurements.

We make the following observations from Figure 2. First, the majority of processing time is spent on DNN training on the client devices. For example, the communication latency for uploading VGG6 is only 1.4s for the client device in Richmond, whereas training VGG6 for five epochs on Nexus 5 takes 8.9s with 60 training samples, which is 6.4× higher than the communication latency. Second, the training time varies significantly across client devices. For instance, training VGG6 with 60 samples for five epochs on Google

Pixel XL takes only 5.1s, which is 2× faster than on Nexus 5. Finally, uploading DNN models from client devices is much slower than downloading DNN models from the central server. For example, downloading VGG6 to the client device in Columbus only takes 361ms, whereas uploading the same DNN takes 1.68s (4.7× larger). Measurements on LeNet and ResNet-18 show a similar trend.

FL Convergence under Non-IID Client Data

In this section, we study the influence of Non-IID data on the FL convergence and possible solutions to mitigate its impact. The unstable FL convergence behavior is triggered by the inconsistency among the local objective functions $F_n(\cdot)$, which is caused by the non-IID-ness in the client training data. One promising approach to alleviate the disparity among local objective functions is to eliminate biased model updates, which are outliers that can hurt overall convergence rate. Previous literature has shown that excluding these outliers can significantly accelerate the training process (Luping, Wei, and Bo 2019; Duan, Li, and Lu 2021; Abay et al. 2020). For example, CMFL (Luping, Wei, and Bo 2019) utilizes the total number of sign differences between the local and global model parameters to measure the bias of the local model. However, this requires all the clients to first finish their local training to produce the local DNN weights, resulting in high processing latency. Moreover, counting the total sign differences will introduce additional computation overhead and further deteriorates the processing latency. To mitigate this issue, we measure the degree of bias using initial training loss, which is the training loss after the first epoch of local training process at each client. Using initial training loss as the indicator offers several advantages. First, the initial training loss naturally reflects the degree of inconsistency between the local client data and the global model, which can be further used to estimate the degree of bias on the local updates. Second, the initial training loss is produced as an intermediate result without any additional computational overhead. After the first epoch, each client device reports its initial training loss to the central server. The server collects the losses and performs early rejection by halting the remaining training process on the devices with high losses. Sending the initial training losses to central server will not impair the processing latency and communication cost, as the training loss (a single scalar) is tiny compared to the DNN model. Conversely, this approach will lower the processing latency and communication overhead, because only a subset of the clients need to complete their local training processes and send their weight updates. For simplicity, we call the first epoch of the local training process the *probing training*, and the initial training loss the *probing loss*. We compare our bias estimation approach with two baseline approaches. The first approach eliminates the outliers by randomly dropping 50% of the clients. The second approach implements FedAvg by keeping all the devices in the local training process. For our approach, at each training round, we reject half of the clients with their probing loss higher than the average probing loss. Figure 3 depicts the average test accuracies on LeNet and VGG6. Our method achieves a higher convergence speed. Specifically, our method obtains test accuracies of 98.7%

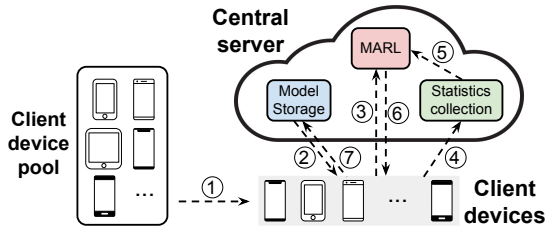


Figure 4: FedMarl workflow, steps are shown in circles.

and 45.2% on MNIST and CIFAR-10, which are 1.3% and 2.8% higher than the other methods on average. The above results demonstrate that the probing loss can be used as **an indicator of client selection** for better FL accuracy under non-IID setting.

Problem Formulation

We present the FL optimization problem formulation in this section. Let $H_{t,n}^c$ denote the local training time for client $n \in N$ at round $t \in T$. Also denote by $H_{t,n}^u$ the communication latency for uploading the local model from client n to the central server. During each training round t , a subset of clients will be early rejected by the central server, while the rest clients will continue to finish their local training process. Define B_n^t as the communication cost for sending updates from client n to the central server at round t . Let $a_n^t \in \{0, 1\}$ denote whether the client n is chosen to complete full local training. The total processing latency H_t of a training round t and the total communication cost B_t for round t can be expressed as:

$$H_t = \max_{1 \leq n \leq N} (H_{t,n}^u + H_{t,n}^c) a_n^t \text{ and } B_t = \sum_n B_n^t a_n^t \quad (1)$$

Finally, let $Acc(T)$ denote the test accuracy of the global model on a global test dataset after the last training round T . Our FL system optimization problem can be defined as:

$$\max_A E \left[w_1 Acc(T) - w_2 \sum_{t \in T} H_t - w_3 \sum_{t \in T} B_t \right] \quad (2)$$

where $A = [a_n^t]$ is a $T \times N$ matrix for client selection, w_1, w_2, w_3 are the importance of the objectives controlled by the FL application designers. Our objective is to maximize the accuracy of the global model while minimizing the total processing latency and communication cost. The expectation is taken over the stochasticity of DNN training.

FedMarl System Design

The FL optimization problem is difficult to solve directly. We instead model the problem as a MARL problem. In this section, we present our problem formulation and FedMarl system design.

MARL Agents Training Process

In FedMarl, each client device n relies on an MARL agent at the central server to make its participation decision. Each

Device type	iPhone8, iPhoneXR, iPad2, Huawei-TAG-TL100, Google Pixel XL, Nexus5, Samsung Galaxy Tab A, Amazon Fire 7 Tablet
Data size	20, 25, 30, 35, 40, 45, 50, 55, 60
Location	Richmond (Virginia), San Francisco (California), Columbus (Ohio), and Toronto (Ontario)

Table 1: Possible options for client device settings.

MARL agent contains a simple two-layer Multi-layer perceptron (MLP) that is cheap to implement. During the training phase of MARL, each MARL agent takes its current state and infers its action a_n^t . Based on the client selection pattern, the central server computes the team reward by considering the test accuracy improvement on the global model, total processing latency and the communication cost. The MARL agents are then trained with VDN to maximize the team reward.

Design of MARL Agent States The state of each MARL agent n consists of six components: the probing loss, the processing latency of probing training, historical values on communication latency, the communication cost from a client device to server, the size of the local training dataset and the current training round index. Let L_n^t denote the probing loss of agent n in round t . At each round t , each agent first performs the probing training and sends L_n^t to the central server. L_n^t is then examined by the corresponding MARL agent in the central server to infer its degree of bias on the local model updates. In addition, to infer the current training latency and communication latency at client device n , each MARL agent is provided with the historical probing training latencies $\mathbf{H}_{t,n}^p = [H_{t-\Delta T_p,n}^p, \dots, H_{t,n}^p]$ and communication latencies $\mathbf{H}_{t,n}^u = [H_{t-\Delta T_c,n}^u, \dots, H_{t-1,n}^u]$, where $H_{t,n}^p$ and $H_{t,n}^u$ denote the latencies for probing training and model uploading of agent n at training round t . ΔT_p and ΔT_c are the sizes of the historical information. Finally, each MARL agent n also involves the communication cost B_n^t , the local data size D_n and training round index t in its input state. This is because the individual communication cost contributes to the total communication cost, and the training data size affects training latency and model accuracy. The state vector \mathbf{s}_n^t of agent n at round t is defined as:

$$\mathbf{s}_n^t = [L_n^t, \mathbf{H}_{t,n}^p, \mathbf{H}_{t,n}^u, B_n^t, D_n, t] \quad (3)$$

To accelerate the training of VDN, we normalize each element in the state vector to make them under the same scale. In addition, this will improve the generalizability of the MARL agents under different system conditions, as shown in the evaluation section. Finally, all the MLPs in the MARL agents share their weights in order to reduce the storage overhead and prevent lazy agent problem (Jiang and Lu 2018).

Description on Agent Actions Given the input state shown in equation 3, each MARL agent n decides whether the client device n should be terminated earlier. In particular, the MARL agent produces a binary action $a_n^t \in [0, 1]$, where $a_n^t = 0$ indicates the client device will be terminated after the probing training, and vice versa.

	MNIST	CIFAR-10	F-MNIST	Shakespeare
FedMarl	96.91%	48.87%	96.14%	44.58%
FedAvg	94.40%	43.49%	94.70%	40.66%
HeteroFL	96.86%	47.74%	96.08%	43.98%
FedProx	95.85%	47.32%	95.73%	43.66%
FedProx-THS	94.43%	45.51%	94.80%	42.69%
FedNova	96.07%	47.97%	95.93%	44.10%
FedNova-THS	95.30%	44.88%	94.42%	42.41%
Oort	96.09%	48.11%	96.02%	44.05%
CS	95.89%	48.19%	96.14%	43.97%

Table 2: Accuracy performance of LeNet, VGG6, ResNet-18, LSTM on their datasets. FedMarl achieves the best accuracies across all the datasets.

Design of the Reward Function To optimize the FL performance described in Equation 2, the reward function should reflect the changes in the test accuracy, processing latency and communication cost after executing client selection decisions generated by the MARL agents. The reward r_t at training round t is defined as:

$$r_t = w_1 [U(\text{Acc}(t)) - U(\text{Acc}(t-1))] - w_2 H_t - w_3 B_t \quad (4)$$

H_t is the processing latency of round t and is defined as:

$$H_t = \max_{1 \leq n \leq N} (H_{t,n}^p) + \max_{n: 1 \leq n \leq N, a_n^t = 1} (H_{t,n}^{rest} + H_{t,n}^u) \quad (5)$$

Here, $\max_{1 \leq n \leq N} H_{t,n}^p$ represent the total time needed for generating all the probing losses. The MARL agents utilize these probing losses to select the client devices that will continue the local training and upload their model updates, which needs an additional time of $\max_{n: 1 \leq n \leq N, a_n^t = 1} (H_{t,n}^{rest} + H_{t,n}^u)$, where $H_{t,n}^{rest}$ is the time required for client device n to finish the local training process. Moreover, $U(\cdot)$ is a utility function that ensures $U(\text{Acc}(t))$ can still alter moderately even if $\text{Acc}(t)$ improvement is small near the end of the FL process. B_t is the total communication cost as defined in equation 1. The MARL agents are trained using VDN described in background section.

FedMarl System Workflow

Figure 4 provides an overview of the FedMarl workflow. The central server contains three building blocks: *Model storage block* for storing and updating the global DNN model, *MARL block* for executing the trained MARL agents and generating the client selection decision, and the *Statistics collection block* for gathering the client device statistics such as processing latency and communication latency. At each training round, N client devices are picked from the client device pool with the criteria described in (Bonawitz et al. 2019) (step 1). The selected devices then receive a copy of global DNN model from the Model storage block (step 2). Next, the client devices perform the probing training and send their probing losses to the MARL block (step 3). Meanwhile, the client devices also transmit their probing training latencies to the Statistics collection block (step 4). After receiving all the probing losses from the clients, the MARL agents take these losses together with the historical information from the Statistics collection block (step 5) and produce the client

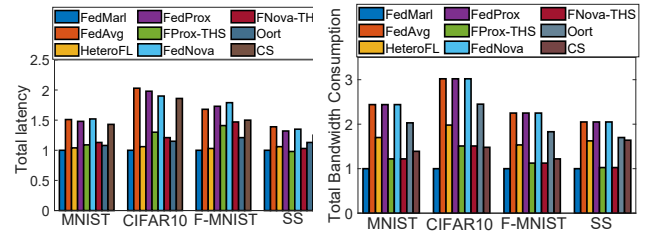


Figure 5: Comparison on (a) normalized latency and (b) normalized bandwidth cost. "SS" denotes Shakespeare dataset.

selection decisions a_n^t (step 6). The selected client devices then perform the rest local training and deliver their model updates to the Model storage block (step 7), which will apply the weight updates to the global DNN model. Meanwhile, the Statistics collection block also updates the existing statistics on communication latency.

Evaluation

We evaluate FedMarl using several popular DNN models, including LeNet (LeCun et al. 1998) on MNIST, VGG6 (Simonyan and Zisserman 2014) on CIFAR-10 and ResNet-18 (He et al. 2016) on Fashion MNIST for image classification, LSTM (Hochreiter and Schmidhuber 1997) on Shakespeare dataset for text prediction. To simulate device heterogeneity, each client device is randomly assigned a device type, a location and a training set size, as summarized in Table 1. We collect training latency data over the above DNNs for each device type under each data size, and measure the time for sending each DNN model from the client devices at each location to the central server, as described in the motivation section. To simulate the non-IID training data at each client device, we sort the training data by its label. For each client device, 80% of its training data are from one random label, the rest of the training data are sampled uniformly from the remaining labels. The number of training data per device is generated by the power law (Li et al. 2020). All client devices adopt a uniform communication cost $B_n^t = 1 \forall t, n$. During each training round, $N = 10$ client devices are randomly selected from a pool of $K = 100$ devices. The training latency and communication latency for each device are sampled from the collected traces based on its device type, data size and location. Each client device first performs the probing training and reports its probing loss to the central server. Based on the feedback from the MARL agents, a subset of them will continue their local training process for $E = 5$ epochs. The number of training round T is set to $T = 20$ for VGG6, and $T = 15$ for LeNet, ResNet-18 and LSTM, respectively. Each MARL agent consists of a MLP of two layers with a hidden layer of 256 neurons. For the reward function (equation 4), we make $w_1 = 1.0$, $w_2 = 0.2$ and $w_3 = 0.1$. The utility function is defined to be $U(x) = 10 - \frac{20}{1 + e^{0.35(1-x)}}$ for shaping the test accuracy. The sizes of the historical information ΔT_p and ΔT_c are set to 3 and 5, respectively. We train the VDN with 300, 200, 300, 200 episodes for LeNet, VGG6, ResNet-18 and LSTM until convergence. To demonstrate the advantage of the MARL over the single-agent RL, we also implement

Source/Target	LeNet	VGG6	ResNet-18
LeNet	1.0 (1.0)	0.720 (0.798)	0.896 (0.930)
VGG6	0.906 (0.974)	0.803 (0.803)	0.859 (0.927)
ResNet-18	0.965 (0.989)	0.731 (0.792)	0.933 (0.933)

Table 3: Performance of the trained MARL agents across different DNNs. All the results are normalized by the performance of agents trained with LeNet and evaluated on itself. Numbers in brackets show the performance after finetuning.

the FedMarl with single-agent RL and compare their convergence behaviors. We observe that MARL approach converges at a much fast speed than single-RL approach under the same training environment.

FedMarl Performance

We compare the performance of FedMarl with multiple advanced benchmark algorithms, including: FedNova (Wang et al. 2020b), HeteroFL (Diao, Ding, and Tarokh 2020), Oort (Lai et al. 2020), FedProx (Li et al. 2020), Clustered Sampling (CS) (Fraboni et al. 2021) and FedAvg (McMahan et al. 2017). The clients perform local training for $E = 6$ epochs in each training round. For FedProx, we adopt the optimal proximal term μ for each DNN, which gives $\mu = 1, 1, 0.1, 0.001$ for LeNet, VGG6, ResNet-18 and LSTM, respectively. For FedNova, the fast client will perform more local training steps until the slowest client finishes its training. The weight updates are then normalized based on the total number of local training steps. For HeteroFL, we adopt five computing complexity levels with a hidden channel shrinkage ratio of 0.5. For Oort, we set the exploitation factor step window and straggler penalty to 0.1, 5, 2 for all the tasks. For CS, we group the clients devices in the pool into 10 clusters, and one representative device is selected from a single cluster per training round. In order to reduce the total processing latency, we further consider a simple client selection. Instead of performing the local training process on all the N client devices, the new selection algorithm, called *Top-half Speed* (THS), selects the bottom 50% of the client devices with the lowest probing training latency during each training round. By removing the straggler client devices, THS can greatly lower the total processing latency and communication cost. We further apply THS on FedProx and FedNova, producing two additional benchmark algorithms: FedProx-THS and FedNova-THS. We perform the evaluation for 100 times and record the average performance for each algorithm. All the local training are performed with SGD.

Table 2 and Figure 5 show the final test accuracy, total processing latencies and communication costs for all the algorithms. In Figure 5, all values are normalized by the values of the FedMarl. FedAvg, FedProx and FedNova make all the clients transmit their local updates, therefore there is no reduction on processing latency and communication cost. In contrast, Oort, CS, HeteroFL, FedProx-THS and FedNova-THS enables only partial client to report their local updates, which reduces the communication cost and processing latency. We notice that FedMarl, HeteroFL and FedNova outperform the rest algorithms on test accuracy, but FedMarl

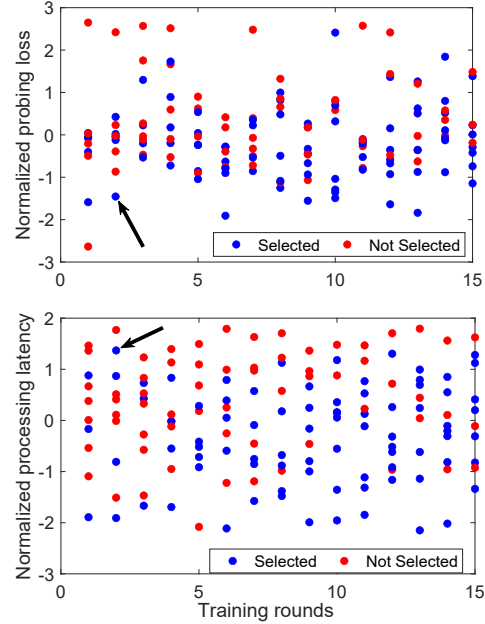


Figure 6: Decisions made by FedMarl. Each dot represent a client device. The blue dot means the client is selected for local training, the red dot means the client is early rejected.

achieves the optimal accuracy, processing latency and communication cost at the same time.

Generalizability of the MARL Agents

In practice, FL system configuration can vary over time. For example, client devices may join and quit the FL application over time, leading to a varying client pool size K . This may require separate MARL agents to be trained for each system configuration, leading to a significant MARL training cost. Next, we employ the MARL agents trained with the default settings described in the beginning of the evaluation section, and evaluate their performances under different conditions.

Evaluation under Different System Conditions We first evaluate the FedMarl performance by varying the pool size K . Specifically, we utilize the MARL agents trained with $K = 100$ and evaluate the performance under $K = 400$ and $K = 800$. Figure 7(a) depicts the total reward (defined in equation 4) of the algorithms on CIFAR-10. Remember that a higher reward indicates a better overall performance in terms of prediction accuracy, processing latency and bandwidth consumption. It can be seen that FedMarl outperforms the rest algorithms for each K . In particular, FedMarl achieves a $1.36\times$ and $1.32\times$ higher total reward than the other algorithms on average under $K = 400$ and $K = 800$, respectively. Similarly, Figure 7(b) shows the performance of the algorithms under different amount of epochs E for local training. FedMarl also obtains the best overall performance under different E . Finally, we modify the FL system configuration by introducing new types of client devices. Specifically, in addition to the eight devices shown in Table 1, we introduce another six mobile devices to the device pool including

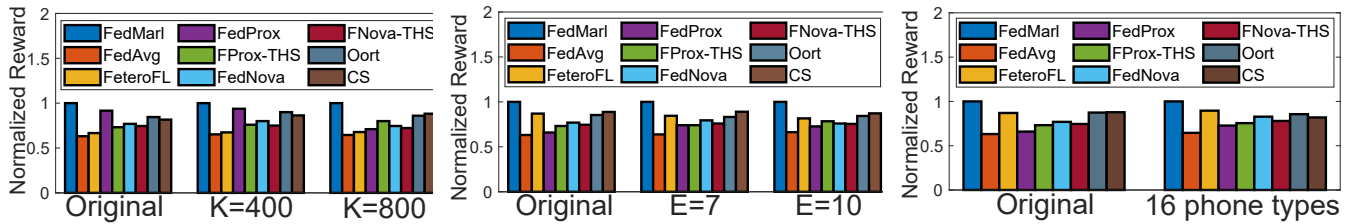


Figure 7: Performance under different system configurations. All the results are normalized with the performance of FedMarl.

w_1, w_2, w_3	LeNet	VGG6	ResNet-18
[1.0, 0.2, 0.1]	A: 98.15%	A: 46.05%	A: 98.43%
	L: 1.0×	L: 1.0×	L: 1.0×
	B: 1.0×	B: 1.0×	B: 1.0×
[1.0, 0.5, 0.3]	A: 97.88%	A: 45.28%	98.03%
	L: 0.82×	L: 0.80×	L: 0.80×
	B: 0.83×	B: 0.78×	B: 0.86×

Table 4: ‘A’, ‘L’, ‘B’ denote the test accuracy, latency and communication cost. The latencies and communication costs are normalized by the performance of the default setting.

iPhone 12, iPhone 7, Samsung Galaxy S21, Google Pixel 5, Samsung A11, Huawei P20 pro. We collect the latency traces for these new devices and evaluate the performances of the algorithms using the new device pool on CIFAR-10 (Figure 7(c)). The results show that the superior performance of the MARL agents can translate across different system settings. This is because the MARL agents only take normalized version of the inputs (equation 3) for generating the decisions, making them independent of a specific system configuration.

Evaluation across Different DNN Architectures and Datasets Next, we evaluate the generalizability of MARL agents over different DNN architectures and datasets. In particular, we train the MARL agents with a source DNN architecture (e.g., VGG6 on CIFAR-10), and evaluate their performance using a target DNN (e.g., ResNet-18 on Fashion MNIST). Table 3 shows the normalized rewards of the MARL agents. We notice that the MARL agents achieve a superior performance across different DNNs and datasets in general. For example, the MARL agents trained on ResNet-18 with Fashion MNIST can obtain a reward of 0.965 on LeNet with MNIST, which is comparable with the performance of the MARL agents trained with LeNet from scratch (1.0). Additionally, by finetuning the MARL agents with 10 episodes on the target DNN (Numbers in the brackets in Table 3), we notice further improvements on the reward. This demonstrates that the trained MARL agents can generalize to different DNN architectures and datasets.

Learned Strategy by the MARL Agents

In this section, we take a closer look at the strategies learnt by the MARL agents. Figure 6 shows the decisions made by the MARL agents during the FL process for LSTM. In particular, we investigate how the client selection decisions are affected by the probing losses (Figure 6 (a)) and the probing training latencies (Figure 6 (b)) of the clients. We make

the following observations. First, the MARL agents prefer to select the clients with both low probing loss and training latency. Second, the MARL agents occasionally select the clients with low probing loss for the better FL convergence, even though their latencies are high. For instance, the two blue dots pointed by the arrows in Figure 6 represent a client device with low probing loss and high processing latency. The MARL agents apply the optimal criteria to achieve a trade off between the training accuracy and total processing latency objectives. Finally, relatively less number of clients are picked in the early stage of the FL training than the later stage. In Figure 6, 3 and 7 (out of 10) clients are selected at the first and last round, respectively. One reason is that in the early stage of training, DNNs usually learn low-complexity (lower-frequency) functional components before learning more advanced features, with the former being more robust to noises and perturbations (Xu et al. 2019; Rahaman et al. 2019). Therefore it is possible to use less data to train in the early stages, which further reduces processing latency and communication cost.

Performance under Different Rewards

In this section, we investigate the impact of relative importance w_1, w_2, w_3 in the reward function (equation 4) on the performance of FedMarl. Specifically, besides the original setting with $[w_1, w_2, w_3] = [1.0, 0.2, 0.1]$, we increase w_2 from 0.2 to 0.5 and w_3 from 0.1 to 0.3. This will force the MARL agents to learn a client selection algorithm with a lower processing latency and communication cost. As shown in Table 4, we observe that increasing w_2 and w_3 will lead to a lower total processing latency and communication cost at the price of lower accuracy. This indicates that FedMarl can adjust its behavior based on the relative importance of the objectives, which enables the application designers to customize the FedMarl based on their preferences.

Conclusion

In this work, we present FedMarl, an MARL-based FL framework that makes intelligent client selection at run time. The MARL agents are pretrained with the simulated training environment which is built using the real traces on DNN training latencies and transmission latencies. The trained MARL agents are then deployed in the FedMarl system to select the client participate at run time. The evaluation results show that FedMarl outperforms the benchmark algorithms in terms of model accuracy, total processing latency and communication cost.

References

2019. DNN Training with Tensorflow Lite. <https://blog.tensorflow.org/2019/12/example-on-device-model-personalization.html>.
2020. DNN Training with CoreML. <https://github.com/hollance/coreml-training>.
- Abay, A.; Zhou, Y.; Baracaldo, N.; Rajamoni, S.; Chuba, E.; and Ludwig, H. 2020. Mitigating bias in federated learning. *arXiv preprint arXiv:2012.02447*.
- Bonawitz, K.; Eichner, H.; Grieskamp, W.; Huba, D.; Ingerman, A.; Ivanov, V.; Kiddon, C.; Konečný, J.; Mazzocchi, S.; McMahan, H. B.; et al. 2019. Towards federated learning at scale: System design. *arXiv preprint arXiv:1902.01046*.
- Das, A.; Gervet, T.; Romoff, J.; Batra, D.; Parikh, D.; Rabbat, M.; and Pineau, J. 2019. Tarmac: Targeted multi-agent communication. In *International Conference on Machine Learning*, 1538–1546. PMLR.
- Deng, Y.; Kamani, M. M.; and Mahdavi, M. 2020. Adaptive personalized federated learning. *arXiv preprint arXiv:2003.13461*.
- Dhokal, S.; Prakash, S.; Yona, Y.; Talwar, S.; and Himayat, N. 2019. Coded federated learning. In *2019 IEEE Globecom Workshops (GC Wkshps)*, 1–6. IEEE.
- Diao, E.; Ding, J.; and Tarokh, V. 2020. HeteroFL: Computation and communication efficient federated learning for heterogeneous clients. *arXiv preprint arXiv:2010.01264*.
- Duan, J.-H.; Li, W.; and Lu, S. 2021. FedDNA: Federated Learning with Decoupled Normalization-Layer Aggregation for Non-IID Data. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 722–737. Springer.
- Fraboni, Y.; Vidal, R.; Kameni, L.; and Lorenzi, M. 2021. Clustered Sampling: Low-Variance and Improved Representativity for Clients Selection in Federated Learning. *arXiv preprint arXiv:2105.05883*.
- He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.
- Hochreiter, S.; and Schmidhuber, J. 1997. Long short-term memory. *Neural computation*, 9(8): 1735–1780.
- Jiang, J.; and Lu, Z. 2018. Learning attentional communication for multi-agent cooperation. In *Advances in Neural Information Processing Systems*, 7254–7264.
- Karimireddy, S. P.; Kale, S.; Mohri, M.; Reddi, S. J.; Stich, S. U.; and Suresh, A. T. 2019. SCAFFOLD: Stochastic Controlled Averaging for Federated Learning. *arXiv preprint arXiv:1910.06378*.
- Konečný, J.; McMahan, H. B.; Yu, F. X.; Richtárik, P.; Suresh, A. T.; and Bacon, D. 2016. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*.
- Lai, F.; Zhu, X.; Madhyastha, H. V.; and Chowdhury, M. 2020. Oort: Efficient federated learning via guided participant selection. *arxiv.org/abs/2010.06081*.
- LeCun, Y.; Bottou, L.; Bengio, Y.; and Haffner, P. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11): 2278–2324.
- Li, T.; Sahu, A. K.; Zaheer, M.; Sanjabi, M.; Talwalkar, A.; and Smith, V. 2020. Federated optimization in heterogeneous networks. *Proceedings of Machine Learning and Systems*, 2: 429–450.
- Luping, W.; Wei, W.; and Bo, L. 2019. Cmfl: Mitigating communication overhead for federated learning. In *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, 954–964. IEEE.
- McMahan, B.; Moore, E.; Ramage, D.; Hampson, S.; and y Arcas, B. A. 2017. Communication-efficient learning of deep networks from decentralized data. In *Artificial Intelligence and Statistics*, 1273–1282. PMLR.
- Nishio, T.; and Yonetani, R. 2019. Client selection for federated learning with heterogeneous resources in mobile edge. In *ICC 2019-2019 IEEE International Conference on Communications (ICC)*, 1–7. IEEE.
- Rahaman, N.; Baratin, A.; Arpit, D.; Draxler, F.; Lin, M.; Hamprecht, F.; Bengio, Y.; and Courville, A. 2019. On the spectral bias of neural networks. In *International Conference on Machine Learning*, 5301–5310. PMLR.
- Reisizadeh, A.; Mokhtari, A.; Hassani, H.; Jadbabaie, A.; and Pedarsani, R. 2020. Fedpaq: A communication-efficient federated learning method with periodic averaging and quantization. In *International Conference on Artificial Intelligence and Statistics*, 2021–2031.
- Rothchild, D.; Panda, A.; Ullah, E.; Ivkin, N.; Stoica, I.; Braverman, V.; Gonzalez, J.; and Arora, R. 2020. Fetchsgd: Communication-efficient federated learning with sketching. *arXiv preprint arXiv:2007.07682*.
- Sattler, F.; Wiedemann, S.; Müller, K.-R.; and Samek, W. 2019. Robust and communication-efficient federated learning from non-iid data. *IEEE transactions on neural networks and learning systems*.
- Simonyan, K.; and Zisserman, A. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.
- Singh, A.; Vepakomma, P.; Gupta, O.; and Raskar, R. 2019. Detailed comparison of communication efficiency of split learning and federated learning. *arXiv preprint arXiv:1909.09145*.
- Smith, V.; Chiang, C.-K.; Sanjabi, M.; and Talwalkar, A. S. 2017. Federated multi-task learning. In *Advances in Neural Information Processing Systems*, 4424–4434.
- Sunehag, P.; Lever, G.; Gruslys, A.; Czarnecki, W. M.; Zambaldi, V.; Jaderberg, M.; Lanctot, M.; Sonnerat, N.; Leibo, J. Z.; Tuyls, K.; et al. 2017. Value-decomposition networks for cooperative multi-agent learning. *arXiv preprint arXiv:1706.05296*.
- Wang, C.; Wei, X.; and Zhou, P. 2020. Optimize scheduling of federated learning on battery-powered mobile devices. In *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 212–221. IEEE.

- Wang, H.; Kaplan, Z.; Niu, D.; and Li, B. 2020a. Optimizing federated learning on non-iid data with reinforcement learning. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*, 1698–1707. IEEE.
- Wang, J.; Liu, Q.; Liang, H.; Joshi, G.; and Poor, H. V. 2020b. Tackling the objective inconsistency problem in heterogeneous federated optimization. *arXiv preprint arXiv:2007.07481*.
- Xu, Z.-Q. J.; Zhang, Y.; Luo, T.; Xiao, Y.; and Ma, Z. 2019. Frequency principle: Fourier analysis sheds light on deep neural networks. *arXiv preprint arXiv:1901.06523*.
- Yang, H.; Fang, M.; and Liu, J. 2021. Achieving Linear Speedup with Partial Worker Participation in Non-IID Federated Learning. *arXiv preprint arXiv:2101.11203*.
- Zhang, M.; Sapra, K.; Fidler, S.; Yeung, S.; and Alvarez, J. M. 2020. Personalized Federated Learning with First Order Model Optimization. *arXiv preprint arXiv:2012.08565*.
- Zhang, S. Q.; Lin, J.; and Zhang, Q. 2022. A Multi-agent Reinforcement Learning Approach for Efficient Client Selection in Federated Learning. *arXiv preprint arXiv:2201.02932*.
- Zhang, S. Q.; Zhang, Q.; and Lin, J. 2019. Efficient Communication in Multi-Agent Reinforcement Learning via Variance Based Control. *arXiv preprint arXiv:1909.02682*.
- Zhang, S. Q.; Zhang, Q.; and Lin, J. 2020. Succinct and robust multi-agent communication with temporal message control. *Advances in Neural Information Processing Systems*, 33: 17271–17282.
- Zhao, Y.; Li, M.; Lai, L.; Suda, N.; Civin, D.; and Chandra, V. 2018. Federated learning with non-iid data. *arXiv preprint arXiv:1806.00582*.