Learning Client Selection Strategy for Federated Learning across Heterogeneous Mobile Devices

***Abstract*—The rapid growth of Internet of Things have yielded a remarkable increase in the volume of the data generated on client devices. This technological trend coincides with the rise of machine learning applications, which leverage user-generated data for large scale model training. In this context, Federated Learning (FL) has become a popular model for facilitating model training across edge devices in a decentralized fashion. However, the statistical diversity presented in the client data and performance heterogeneity existed among the user mobile device can seriously impact the accuracy of the result model and system performance of FL. This article first illustrates the state-of-the-art FL algorithms and investigates the major issues presented in the FL implementation, and then presents a novel FL algorithm that jointly optimizes both the model performance and implementation efficiency for the FL systems. Specifically, we propose an intelligent FL client selection scheme by leveraging the recent advance of Reinforcement Learning (RL) in solving complex control problems. The proposed solution, termed *IntelliFL*, can greatly improve both the accuracy performance and system performance of FL under the training environment with heterogeneous client devices.**

I. Introduction

The emergence of Internet of Things (IoT) applications such as autonomous driving, smart homes, and augmented reality has led to a explosive growth of data generated on client devices. Subsequently, these applications leverage this tremendous volume of data to train their large-scale deep neural network (DNN) models, in order to provide better application performance and user experience. In order to train these large-scale DNN models, a naïve solution is to perform centralized DNN training by sending all client data to the cloud server. However, this is both inefficient and prone to privacy and security threats. To address these limitations, Federated learning (FL) has been proposed as new learning paradigm in which multiple client devices communicate with a coordinating server to jointly train a single global model without exposing the raw user data [1]. During the training process, each client device will first receive a copy of the global DNN model, and then perform local DNN training with its private user data. After the local training process is finished, the DNN model updates will be sent to the central server. The central server in the cloud will aggregate the DNN updates to produce an updated version of the global DNN model. This process repeats until the global DNN model is converged. Unlike the centralized training scheme, FL naturally addresses the scalability and privacy issues, and hence has gained significant attention in recent years.

Despite its benefits, however, the accuracy and performance (in terms of training time and resource cost) of FL is highly dependent on its operating environment. This is especially the case when running FL over mobile devices[1]. In practice, the heterogeneous computing and network resources on mobile devices often causes stragglers that can significantly slow down the FL training process, and severely impair the performance of the delay-sensitive FL applications. This issue is further exacerbated by the uneven distribution of client data size, as a client with more data would require higher local training time. Moreover, FL operation often incurs high communication cost, mainly due to sending the weight updates from mobile devices to the central server. On one hand, the client devices usually have limited bandwidth and high communication costs. On the other hand, the ever-growing DNN model complexity requires the client to deliver increasingly larger model updates, which further increases the network resource consumption. The limited bandwidth resource and heterogeneous processing powers of the mobile devices greatly constraints on the design space of the FL system. Finally, from the accuracy perspective, the underlying statistical heterogeneity on the client devices leads to non-independent and identically distributed (non-IID) local training data, which can severely slow down the convergence of the global DNN model and degrade the accuracy performance [2]–[4].

In order to expand practicality of FL for real-world applications, it is of paramount importance to design an efficient FL solution that can simultaneously mitigate all the aforementioned problems. However, designing such a solution is challenging because of (1) the dynamic FL system condition, (2) the intractable convergence behavior of the FL training, and the complicated interplay between these two factors. In this work, we leverage the recent advances in Reinforcement Learning (RL) and propose an FL framework called *IntelliFL* (Figure 1). IntelliFL explores the tangled interactions among the client devices and central server and exploits the recent statistics on the device performances and training behaviors, then produces the efficient client selection and DNN mapping decisions based on the designated objective functions imposed by the FL application designers. IntelliFL offers an elastic resource management framework to handle the resource heterogeneity among the client devices, leading to superior performance in terms of both system implementation and prediction accuracy.

The rest of the article is organized as follows. We first summarize the recent development in FL and illustrate the

[1]In this article, we will use the term "mobile device" and "client device" interchangeably.

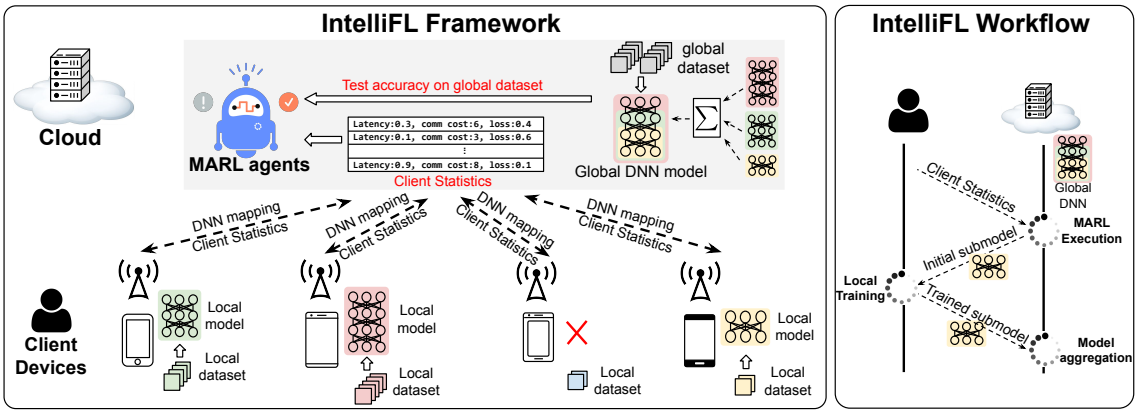


Fig. 1: The left part of the figure shows the overall architecture of IntelliFL system, the right part of the figure depicts the IntelliFL workflow during each training round.

major research challenges of the FL design, we also give a background introduction on Multi-agent Reinforcement Learning (MARL) and Hierarchical DNN in Section II. We then describe the overall problem in detail together with the proposed IntelliFL framework in Section III. After that, we will present the evaluation results of IntelliFL in section IV. We conclude the paper with a discussion on the open issues for future FL research in section V.

II. Major Challenges in Federated Learning

A. Statistical Heterogeneity on Local Training Data

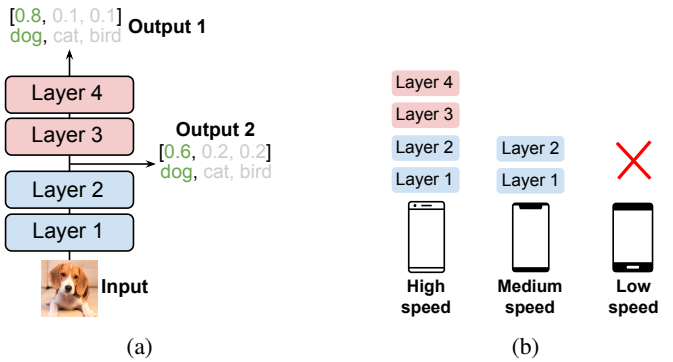
As an extension of distributed learning, FL allows the DNN models to be trained in a distributed fashion with client privacy protection. The first FL framework, Federated Averaging (FedAvg) [1], uses a central server to send the global DNN model to a set of selected clients, which then perform local Stochastic Gradient Descent (SGD) and return their model updates to the central server.

However, the heterogeneity on the local training data will induce non-IID data distribution among client devices, this underlying statistical diversity will further cause inconsistency in the client weight updates and severely undermines the accuracy of the global DNN model after applying the aggregated weight updates on the global DNN model [2], [4]–[6]. To improve the accuracy performance of FL in the non-IID data scenario, FedProx [4] restrains the divergence between the local model and global model by introducing an extra regularization term in the FL training objective. Scaffold [3] adopts the variance reduction technique to correct the local updates so that the local model will not deviate too much from the global model. Although these two approaches achieve decent improvement on the global model accuracy, the result accuracy is still much lower than that from the DNN trained using the IID dataset.

Another promising approach to alleviate the disparity among local objective functions is to detect and eliminate the biased model updates during the aggregation, because these outliers can hurt global model accuracy. To detect these biased model updates, CMFL [7] counts the total number of local weight values whose sign is opposite from the corresponding weight value in the global model, and uses this total count as an indicator of the bias degree for the local model. However, this requires additional operations in the client devices in order to count the differences on the weight signs and further increases the workload of the resource-limited client devices. FedMarl [8] utilizes the training loss after the first round of local training (termed *probing loss*) to measure the degree of bias on the local DNN weights. The training loss naturally reflects the degree of inconsistency between the local DNN data and the global model weights. Furthermore, the training loss is generated as an intermediate result during the local DNN training process, therefore no additional operation is required. In this work, we utilize the training loss to measure the degree of the bias.

B. System Heterogeneity on Local Client Devices

Aside from the diverse local training data, the heterogeneous computing resources of the client devices can also degrade the overall FL system performance [9]. During the FL operation, a straggler device can significantly increase FL training latency, making FL unsuitable for delay-sensitive applications. To tackle this problem, Oort [10] prioritizes the use of the clients who can offer the greatest improvement on the global model accuracy and the capability to run training quickly. A heuristic utility function is proposed to evaluate the clients, and only the clients with a high utility score will be selected for local training. Although Oort outperforms some baseline algorithms in terms of accuracy and processing latency, its utility function is still not optimal and more sophisticated solutions are left to be developed. To reduce the training latency, another solution is to partition the DNN model into multiple parts, and assigning one or more parts of DNN for local training based on the computing capability of each client device. For example, HeteroFL [11] partitions the global DNN model and assigns the submodel to the clients based on their computing capability. The trained submodels are then sent from each client and are aggregated into a single global DNN



(a) (b)

Fig. 2: (a) Layerwise partition is applied on a four-layer DNN model, which produces two exit points. Both exit points can generate inference results. (b) During the FL operation, a submodel will be assigned to each client devices according to their computing capabilities. A device with higher processing power will be given the entire DNN for local training, while a slow device will be eliminated from the local training process.

model in the cloud. HeteroFL provides a solution to jointly optimize both the model accuracy and FL system performance.

To partition the DNN model, one effective partition scheme is *layerwise partition* [12]. Figure 2 shows an example of applying layerwise partition on a DNN with four layers. An early exit is introduced at the end of the second layer, as shown in Figure 2a. During the FL execution, the faster device will be assigned with the entire DNN model to train, whereas the slower devices will only be assigned with a two-layer submodel or even be eliminated from the local training process, as shown in Figure 2b. This enables the load balancing on the computational workloads across the client devices, which further reduces the processing latency and communication cost of FL.

III. IntelliFL System for Resource-Efficient Federated Learning Implementation

A. Problem Statement

In this section, we present the FL optimization problem in detail. Assume layerwise partition is applied to the global DNN model D with $m(1 \leq m \leq M)$ exit points (including the original DNN output), and denote D_m the submodel with the exit point m. Let $A(m)$ represent the final test accuracy of D_m over the global test dataset. In addition, let $T_{m,n}$ denote the processing time needed for training D_m locally at client $n(1 \leq n \leq N)$, and denote $B_{m,n}$ the total communication cost of sending D_m from client n to the central server. Our FL optimization problem can be described as follows:

$$\max_{Q} \ \mathbf{E}\Big[w_1 \sum_{m=1}^{M} A(m) - w_2 \sum_{r=1}^{R} T_{proc,r} - w_3 \sum_{r=1}^{R} B_{comm,r}\Big]$$
$$\text{s.t.} \ \sum_{m=1}^{M} q_{r,m,n} \leq 1, \quad \forall 1 \leq n \leq N, 1 \leq m \leq M \tag{1}$$

where $T_{proc,r} = \max_{n,m} T_{m,n} q_{r,m,n}$ and $B_{comm,r} = \sum_{m=1}^{M} \sum_{n=1}^{N} B_{m,n} q_{r,m,n}$ are the total processing time and total communication cost of training round r, respectively. The expectation is taken over the stochasticity of the local training data at each client. $q_{r,m,n} \in \{0, 1\}$ is a binary variable, where $q_{r,m,n} = 1$ if client device n is assigned with submodel D_m at the training round r. Also, let $Q = \{q_{r,m,n}\}$ represent an $R \times M \times N$ matrix of $q_{r,m,n}$. At training round r, a client n can not be assigned with more than one submodel. Finally, w_1, w_2 and w_3 indicate the relative importance of the objectives, which are determined by the FL application designer.

Due to the intractable convergence behavior of the FL training over the non-IID client data, the dynamic variation on the mobile device performance and the complicated interaction between the above objectives, it is challenging to solve this problem using hand-tuned heuristic solutions directly. We instead model the problem as a MARL problem and reply on the MARL agents to generate the optimal control decisions.

B. An Overview on Multi-agent Reinforcement Learning

In recent years, Multi-agent Reinforcement Learning (MARL) has achieved success in many distributed learning tasks such as autonomous driving, swarm robotics, traffic control etc. In cooperative MARL, a group of agents shares the common objective function, they are trained jointly to collaborate and act in a proper way to generate the maximum team reward. In this work, we consider a standard MARL setting. At each timestamp, each MARL agent receives an input state and produces an action based on the input state. After all the agents in the team have performed their actions, a global reward will be assigned to them based on the team performance. The MARL agents are usually trained using a simulated training environment, in which the goal training is to shape the DNN presented in each MARL agent in order to maximize team reward during the MARL execution. Next, we will describe the detailed FL system design.

C. IntelliFL System Design

The architecture of IntelliFL is shown in the left part of Figure 1. The right part of Figure 1 provides an overview of the IntelliFL workflow. The MARL agents are pretrained offline and deployed on the central cloud server to make online decisions. The global DNN model is also stored in the central cloud. During each FL training round, each client device first delivers the client statistics (e.g., training loss, processing latency, etc) to the cloud, which will be used as the input of the MARL agents. The size of client statistics is tiny compared to the DNN model, and therefore will not impair the total communication cost. After receiving all the client statistics, the MARL agents will then generate the DNN mapping decision, and the central server will then deliver the corresponding DNN submodels to the client devices. After receiving the initial submodel, the client device will perform the local training using their private data. After the local training processes are finished, the resulting DNN submodels will be uploaded to the central server.

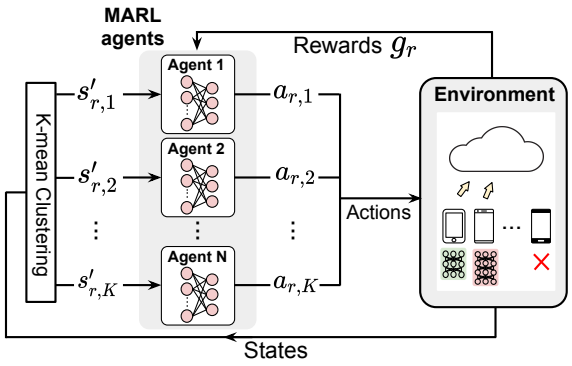


Fig. 3: The training process for the MARL agents.

D. MARL Agents Training Process

In IntelliFL, each mobile device n relies on the MARL agent at the central server to decide its local DNN architecture. To train the MARL, each MARL agent takes its input state $s_{r,k}$ and infers its DNN mapping decision $a_{r,k}$. Based on the decision, the MARL agents then receive a team reward g_r based on the test accuracy improvement on the global model and changes on the total processing latency and communication cost. The MARL agents are then trained to maximize the team reward. The MARL training process is highlighted in Figure 3.

1) Design of the Input States: The input state of the MARL agents involves the historical information on the client device performance and local training statistics. Specifically, it contains the following components:

1) Historical information on the training loss with the local client dataset.
2) Historical information on processing latency for training the DNN model.
3) Historical information on DNN mapping decisions for the prior training rounds.
4) Size of the local DNN model.
5) The size of the local training dataset.
6) The current training round index.

The training loss is used to estimate the degree of bias on the local DNN weights, the processing latency is used to evaluate the computing capability of a mobile device, and local DNN model size is utilized to assess the amount of the transmitted information from a mobile device to the central cloud, which is related to the communication cost. Additionally, the information on the past mapping decision, local training dataset and current round number will also impact the FL system performance and model model accuracy.

2) Design of the Agent Actions: After receiving the input states from the clients, the MARL agent k will generate a one-hot action vector $\mathbf{a}_{r,k}$ with a length of $M+1$, where M is total number of exit points in the global DNN model. Setting the i-th element of $\mathbf{a}_{r,k}$ to one indicates a submodel with the i-th exit point. The additional element in the action vector is used to indicate the situation that the MARL agent is eliminated from the local DNN training.

3) Design of the MARL Reward: The reward function needs to reflect the impact of the current DNN mapping decision on the test accuracy of each submodel, processing latency and communication cost of the entire FL system. During the MARL training stage, we record the changes on the above information between the consecutive training rounds r and $r-1$. The MARL reward can then be generated by computing the weighted sum of the changes with the weights w_1, w_2 and w_3 described in equation 1.

4) Scalable IntelliFL Implementation: As the number of client devices increases, it is not scalable to associate each client device with a dedicated MARL agent for generating the mapping decision. To reduce the number of MARL agents implemented in the cloud, we categorize the input states $s_{r,n}$ from the clients n into K clusters using the k-means clustering algorithm. The center of each cluster $s'_{r,k}$ will be used as the input state of the K MARL agents. All the clients in the same cluster will share the same DNN mapping produced by the associated MARL agents.

IV. Evaluation

To train the MARL agents, we perform extensive experiments to collect the real traces on DNN training latency over multiple mobile devices. We further build a training environment using the collected traces to simulate the FL system operations. The MARL agents are first trained offline with the simulated training environment, and the trained MARL agents are then adopted by the IntelliFL system to make online decision.

A. Building the Simulated Training Environment

We collect the training latencies on eight different mobile devices with different processing powers, including Huawei-TAG-TL100, iPhone8, Google Pixel XL, iPhone XR, iPad2, Nexus 5, Amazon Fire 7 Tablet, and Samsung Galaxy Tab A 8.0. For each device, we measure the time for performing 5 epochs of training with different data sizes ranging from 20 to 60 samples. We developed our own DNN training platform on the mobile devices based on the previous literature [13]. We use three popular Convolutional Network Networks (CNNs), including LeNet on MNIST dataset, VGG6 on CIFAR-10 dataset and ResNet-18 on Fashion-MNIST dataset. The batch size is fixed at 20 for all the measurements. For each setup, we perform the measurement for 200 runs and report an average training time. For the devices with Android system, the DNN models are first built with Keras and then converted to TensorFlow Lite format before running on the client devices. For iOS devices, we first build their DNN models using PyTorch and convert the resulting model to CoreML format with coremltools. We collect training latency data over the above DNNs for each device type under each data size. We observe that, even under the same training dataset size, the training time varies significantly across different client devices. For example, training VGG6 with 60 samples for five epochs on Google Pixel XL takes only $5.2s$, which is $2.2\times$ faster than Huawei-TAG-TL100. The difference in the local training set

size further exacerbates the diversity of the training time. For instance, training VGG6 with 20 samples for five epochs on iPad7 takes $1.1s$, which is $8.1\times$ faster than training VGG6 with 60 samples on Nexus5.

Based on the measured results, we then build a simulated FL environment with 100 client devices to train the MARL agents. To simulate device heterogeneity, each client device is randomly assigned a device type and a training set size. 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 according to the power law [4]. We assume the communication cost $B_{m,n}$ is directly proportional to the size of the local DNN model D_m uploaded from the client to the cloud. After receiving the corresponding DNN models from the central server, each client device will perform local training process for $E = 5$ epochs (as shown in the right part of Figure 1). The number of training round T is set to $T = 20, 15, 15$ for VGG6, LeNet and ResNet-18, respectively.

B. Training the MARL Agents

Each MARL agent consists of an MLP of two layers with a hidden layer of 256 neurons. The MARL agents are trained under two different settings on the relative importance defined in equation 1. In setting A and setting B, we make $[w_1, w_2, w_3]$ to $[1.0, 0.2, 0.1]$ and $[1.0, 0.4, 0.2]$, respectively. Note that IntelliFL can work with any selection of the relative importance w. During each training round r, the client devices first deliver their statistics to the central server, the statistics are then classified into $K = 10$ clusters using k-means clustering algorithm. The MARL agents will then produce the DNN mapping decisions for each client cluster. We apply two early exit points on the global DNN model, so each MARL agent has four possible actions to select:

1) Assign the entire DNN to the client device.
2) Assign the DNN with the first early exit to the client device.
3) Assign the DNN with the second early exit to the client device.
4) Eliminate the client device from the local DNN training process.

We train the MARL agents with 300, 200 and 300 episodes for LeNet, VGG6 and ResNet-18 until convergence. We notice that all the MARL training processes converge with a high reward, which indicates that the trained MARL agents will generate a better overall performance on prediction accuracy, processing latency and communication cost.

C. Performance Evaluation

We first evaluate the performance of IntelliFL in terms of test accuracy, processing latency and communication cost. In particular, we compare IntelliFL with several popular FL algorithms, including: FedAvg [1], Oort [10], FedNova [14], HeteroFL [11], FedProx [4], and FedMarl [8]. We train the MARL agents using the simulated environment described in Section IV-A, and evaluate the performance under the same environment. 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 FedProx, we utilize the optimal proximal term μ for each DNN, which gives $\mu = 1, 1, 0.1$ for LeNet, VGG6 and ResNet-18, respectively. For HeteroFL, we use five computing complexity levels, the hidden channel shrinkage ratio is set to 0.5. For FedMarl, we set the weight of the objective function to $1.0, 0.3, 0.3$, respectively. For Oort, the exploitation factor, step window and straggler penalty are set to $0.1, 5, 2$ for all the tasks.

Table I lists the final test accuracies of the global model, total processing latencies and communication costs for all the algorithms. For IntelliFL, we report the test accuracy of the entire DNN, since the rest algorithms do not support the training of the hierarchical DNN model. As shown in Table I, FedAvg, FedProx and FedNova make all the clients transmit their local updates, and therefore there is no reduction in processing latency and communication cost. By comparison, in Oort, FedMarl, HeteroFL, and IntelliFL, only partial clients are allowed to deliver their local model to the cloud server, which effectively reduces the FL processing latency and communication overhead. We have the following observation. First, IntelliFL, FedMarl and HeteroFL achieve a much better performance on test accuracy, training latency and communication cost than the rest algorithms on all the three DNN models. For example, IntelliFL (setting A) obtains an average of 22% and 12% savings on the processing latency and communication cost, while achieving very high test accuracies across all the three DNN architectures. Second, we notice that the relative importance w_1, w_2, w_3 plays an important role on the performance of the IntelliFL system. Specifically, by increasing the relative importance w_2 and w_3, the overall processing latency and communication cost will further decrease accordingly. This enables the FL application designer to customize the IntelliFL system performance based on their preferences.

D. Performance of MARL Agents under Dynamic Environment

The practical environment for FL implementation may differ from the simulated environment that is utilized to train the MARL agents. For example, in a mobile edge environment, the mobile devices may join and leave the FL system over time, leading to a dynamic FL system configuration. It is impractical to train a separate set of MARL agents for each scenario. In this section, we evaluate the generalizability of the MARL agents. We train the MARL agents under the simulated environment, and we demonstrate that the trained MARL agents can still achieve a great performance under an FL system configuration that is different from the simulated environment.

We first evaluate the impact of the client devices in the FL system on the IntelliFL performance. In particular, we first train the MARL agents in a simulated environment with 100

heightModel	FedAvg	HeteroFL	FedProx	FedNova	Oort	FedMarl	IntelliFL (setting A)	IntelliFL (setting B)
LeNet	A: 94.40% L: 1.0× B: 1.0×	A: 96.86% L:0.76× B:0.80×	A: 95.85% L:0.96× B:1.0×	A: 96.07% L: 0.98× B: 1.0×	A: 96.09% L:0.82× B:0.84×	A: 96.91% L:0.73× B:0.85×	A: 96.90% L:0.68× B:0.81×	A: 96.84% L:0.60× B:0.72×
VGG6	A: 43.49% L: 1.0× B: 1.0×	A: 47.74% L:0.57× B:0.62×	A: 47.32% L:0.93× B:1.0×	A: 47.97% L: 0.91× B: 1.0×	A: 48.11% L:0.60× B:0.78×	A: 48.87% L:0.54× B:0.44×	A: 49.33% L: 0.44× B: 0.47×	A: 48.72% L:0.40× B:0.40×
ResNet-18	A: 94.70% L: 1.0× B: 1.0×	A: 96.08% L:0.70× B:0.73×	A: 95.73% L:0.96× B:1.0×	A: 95.93% L: 0.98× B: 1.0×	A: 96.02% L:0.71× B:0.79×	A: 96.14% L:0.66× B:0.77×	A: 96.23% L: 0.63× B: 0.69×	A: 96.15% L:0.57× B:0.61×

TABLE I: Performance evaluation of IntelliFL. "A","L","B" denote the test accuracy, latency and communication cost, respectively. The latencies and communication costs are normalized by the performance of FedAvg. We adopt two different settings on the objective function to train the MARL agents of IntelliFL. In setting A and setting B, we make $[w_1, w_2, w_3] = [1.0, 0.2, 0.1]$ and $[w_1, w_2, w_3] = [1.0, 0.4, 0.2]$, respectively.

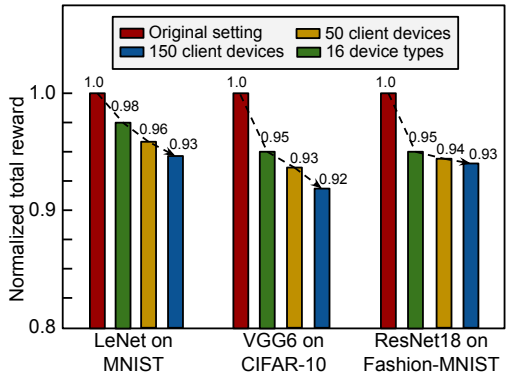


Fig. 4: Performance of IntelliFL under different system configurations. The results are normalized with the reward obtained under the original system settings.

clients, and evaluate the performance of IntelliFL under an FL configuration with 50 and 150 clients. We adjust the cluster size of the K-mean algorithm to keep the total number of MARL agents the same (K=10). Figure 4 (yellow and blue bars) shows the total reward of the IntelliFL under different numbers of client devices. Note that a high reward indicates better overall performance in terms of test accuracy, processing latency and communication cost. We observe that the MARL agents can still achieve a great performance with different numbers of clients. We then measure the performance of IntelliFL by increasing the diversity on the client devices. In addition to the eight devices adopted in the simulated environment described in Section IV-A, we introduce eight more synthetic device types, where each device type has a synthetic processing time for local DNN training. This leads to a total of 16 device types, and each of the 100 client devices in the FL system is assigned with one of the 16 device types. The evaluation results presented in Figure 4 (red bars) demonstrate that the superior performance of the MARL agents can also translate across the device types. In summary, the trained MARL agents can generalize well under different system configurations without training the MARL agents for each particular setting.

V. Conclusions

IntelliFL describes a reinforcement learning based FL framework that aims at jointly solving multiple problems across machine learning algorithm and FL system performance. The evaluation results show that IntelliFL can outperform the benchmark algorithms in terms of model accuracy, total processing latency and communication cost. Furthermore, IntelliFL can also operate under dynamic system configurations while maintaining superior overall performance.

References

[1] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial intelligence and statistics*. PMLR, 2017, pp. 1273–1282.

[2] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra, "Federated learning with non-iid data," *arXiv preprint arXiv:1806.00582*, 2018.

[3] S. P. Karimireddy, S. Kale, M. Mohri, S. J. Reddi, S. U. Stich, and A. T. Suresh, "Scaffold: Stochastic controlled averaging for federated learning," *arXiv preprint arXiv:1910.06378*, 2019.

[4] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated optimization in heterogeneous networks," *Proceedings of Machine Learning and Systems*, vol. 2, pp. 429–450, 2020.

[5] Y. Deng, M. M. Kamani, and M. Mahdavi, "Adaptive personalized federated learning," *arXiv preprint arXiv:2003.13461*, 2020.

[6] H. Wang, M. Yurochkin, Y. Sun, D. Papailiopoulos, and Y. Khazaeni, "Federated learning with matched averaging," *arXiv preprint arXiv:2002.06440*, 2020.

[7] W. Luping, W. Wei, and L. Bo, "Cmfl: Mitigating communication overhead for federated learning," in *IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2019, pp. 954–964.

[8] S. Q. Zhang, J. Lin, and Q. Zhang, "A multi-agent reinforcement learning approach for efficient client selection in federated learning," *arXiv preprint arXiv:2201.02932*, 2022.

[9] T. Nishio and R. Yonetani, "Client selection for federated learning with heterogeneous resources in mobile edge," in *ICC IEEE International Conference on Communications (ICC)*. IEEE, 2019.

[10] F. Lai, X. Zhu, H. V. Madhyastha, and M. Chowdhury, "Oort: Efficient federated learning via guided participant selection," *arxiv.org/abs/2010.06081*, 2020.

[11] E. Diao, J. Ding, and V. Tarokh, "Heterofl: Computation and communication efficient federated learning for heterogeneous clients," *arXiv preprint arXiv:2010.01264*, 2020.

[12] Y. Kang *et al.*, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," in *ACM SIGARCH Computer Architecture News*, vol. 45, no. 1. ACM, 2017, pp. 615–629.

[13] P. Senchanka, "Example on-device model personalization with TensorFlow Lite," https://blog.tensorflow.org/2019/12/example-on-device-model-personalization.html, 2019, [Accessed 01-May-2022].

[14] J. Wang, Q. Liu, H. Liang, G. Joshi, and H. V. Poor, "Tackling the objective inconsistency problem in heterogeneous federated optimization," *arXiv preprint arXiv:2007.07481*, 2020.