Article

FedDdrl: Federated Double Deep Reinforcement Learning for Heterogeneous IoT with Adaptive Early Client Termination and Local Epoch Adjustment

Yi Jie Wong [1], Mau-Luen Tham [1,*], Ban-Hoe Kwan [2] and Yasunori Owada [3]

[1] Department of Electrical and Electronic Engineering, Lee Kong Chian Faculty of Engineering and Science, Universiti Tunku Abdul Rahman, Kajang 43000, Malaysia
[2] Department of Mechatronics and Biomedical Engineering, Lee Kong Chian Faculty of Engineering and Science, Universiti Tunku Abdul Rahman, Kajang 43000, Malaysia
[3] Resilient ICT Research Center, Network Research Institute, National Institute of Information and Communications Technology (NICT), Tokyo 184-8795, Japan
* Correspondence: thamml@utar.edu.my

Abstract: Federated learning (FL) is a technique that allows multiple clients to collaboratively train a global model without sharing their sensitive and bandwidth-hungry data. This paper presents a joint early client termination and local epoch adjustment for FL. We consider the challenges of heterogeneous Internet of Things (IoT) environments including non-independent and identically distributed (non-IID) data as well as diverse computing and communication capabilities. The goal is to strike the best tradeoff among three conflicting objectives, namely global model accuracy, training latency and communication cost. We first leverage the balanced-MixUp technique to mitigate the influence of non-IID data on the FL convergence rate. A weighted sum optimization problem is then formulated and solved via our proposed FL double deep reinforcement learning (FedDdrl) framework, which outputs a dual action. The former indicates whether a participating FL client is dropped, whereas the latter specifies how long each remaining client needs to complete its local training task. Simulation results show that FedDdrl outperforms the existing FL scheme in terms of overall tradeoff. Specifically, FedDdrl achieves higher model accuracy by about 4% while incurring 30% less latency and communication costs.

Keywords: federated learning; client selection; local epoch adjustment; deep reinforcement learning; Internet of Things

check for updates

Citation: Wong, Y.J.; Tham, M.-L.; Kwan, B.-H.; Owada, Y. FedDdrl: Federated Double Deep Reinforcement Learning for Heterogeneous IoT with Adaptive Early Client Termination and Local Epoch Adjustment. *Sensors* **2023**, *23*, 2494. https://doi.org/10.3390/s23052494

Academic Editors: Raffaele Montella, José Luis González Compeán and Sokol Kosta

Received: 30 December 2022
Revised: 3 February 2023
Accepted: 7 February 2023
Published: 23 February 2023

1. Introduction

In the Internet of Things (IoT), each device can collect massive amounts of data (i.e., measurements and location information) [1]. It is estimated that IoT devices will generate over 90 zettabytes of data globally by 2025 [2]. These data can be uploaded to a centralized server, where a new model can be retrained or fine-tuned using the collected dataset. This method is called centralized learning (CL) since the data must be centralized at one location for model training. However, privacy concerns make it inconvenient for devices to share potentially sensitive data with a centralized server (or any other party). For example, medical images may contain sensitive and private information about patients [3], which prohibits the collection of such data from multiple healthcare institutions for CL. Additionally, uploading bandwidth-hungry data requires a high communication cost, which is not feasible for most IoT devices with network resource constraints [1,4]. In fact, the data collected by IoT devices could be larger than the model size [4], especially when dealing with image data.

Federated learning (FL) has emerged as one of the promising candidates to address this challenge. FL is a technique that trains an algorithm across multiple edge devices holding individual local datasets without sharing or exchanging them. First, each IoT device (client) uses its locally collected data to train a local model. After training, each IoT device uploads its locally trained models to an FL server for aggregation. A new global model is generated, which is trained using data from all participating clients without actually sharing the sensitive and bandwidth-hungry data. Thus, FL addresses data privacy concerns by training a global model in distributed environments. FL has since been applied in various applications, ranging from mobile keyboard prediction [5] to natural disaster classification [6,7] and medical image segmentation [3]. Using FL, Google trained its mobile keyboard prediction using 600 million sentences from a surprising amount of 1.5 million clients [5]. In addition, Intel released its production-ready and open-source FL (OpenFL) framework [8]. OpenFL is used in the Federated Tumor Segmentation (FeTS) initiative, which is a program participated in by 56 clinical sites around the globe to train tumor segmentation models via FL. Experiments show that FL models can reach 99% of CL model without sharing the sensitive data [3]. These large-scale real-life applications proved the huge economic value of FL.

FL coupled with the IoT has huge potential for real-world application. For instance, research works [9–11] combined FL with industrial IoT (IIoT), creating an industrial-grade hierarchical FL framework. Hierarchical FL is a three-layer architecture FL framework composed of clients, edge servers and a cloud server. Regular FL is performed between the edge server and its corresponding client device. Upon model aggregation at the edge servers, the aggregated models are then uploaded to the cloud for global model aggregation. Experiments show hierarchical FL to be superior to a regular FL, with lower training latency and better convergence [12]. This is because the model aggregation at the client edge before global model aggregation can significantly reduce the training divergence. Due to the robustness of hierarchical FL, it has also been exploited to empower digital twins [10,13]. In this study, we only focus on regular FL, which is the fundamental building block for any sophisticated FL framework.

Despite huge potential, FL still faces several challenges from practical implementation: (1) model convergence in the presence of a non-independent and identically distributed (non-IID) dataset, (2) computing efficiency and (3) communication efficiency [14,15]. First, data is usually not uniformly distributed across IoT devices. Realistically, each IoT device has a unique data distribution and can be considered non-IID, whereas the global population (if the data is centralized) would be IID. According to [16], the earth mover's distance (EMD) between the local client data distribution and the global population is the main reason the FL model diverges from the global optima solution. This is also termed weight divergence between FL and CL models, which greatly reduces the convergence rate of FL models [16]. Additionally, the heterogeneity of computing and communication resources in IoT networks hinders resource utilization efficiency. In most studies, except [17,18], the local epoch number is set to be the same for all client devices disregarding their computing constraints. As a result, devices with stronger computing power often have to wait for the straggler devices to complete their training, which drastically increases the overall training latency. Moreover, some clients may not have access to high-speed networks, making local model uploading slow or unrealistic.

Many previous studies aimed to tackle the three challenges from different viewpoints. However, optimizing one of the objectives might deteriorate the other objectives [14]. Deep reinforcement learning (DRL) has recently been exploited for FL resource optimization. However, to the best of our knowledge, none of the DRL-based FL frameworks allows dynamic local epoch adjustment. Past studies [9,10,14,19–21] only exploited DRL to select clients that fulfil the resource constraints without introducing a tuning mechanism to adjust the local training epoch for clients with limited computing power.

To this end, we present federated double deep reinforcement learning (FedDdrl). FedDdrl exploits the double DRL (DDRL) framework, which uses two DRL agents to find the optimal client selection and local epoch adjustment policies. Our objective was to maximize the global model's accuracy while minimizing the FL system's training latency and communication cost. We first formulated the FL protocol as a Markov Decision Process (MDP). We then adopted two DRLs based on Value Decomposition Networks (VDNs) as the policy networks. To speed up the convergence speed, we adopted the recently proposed balanced-MixUp [22] augmentation technique to mitigate weight divergence. Simulation results showed that our FedDdrl algorithm improved model accuracy with lower training latency and communication cost. Note that FL facilitates edge computing, which is one of the goals of the ASEAN IVO project titled "Context-Aware Disaster Mitigation using Mobile Edge Computing and Wireless Mesh Network".

We summarize our contributions as follows.

1. We modeled the FL system as an MDP. Then, we proposed to use a DDRL framework for adaptive early client termination and local epoch adjustment, to maximize the global model accuracy while minimizing the training latency and communication costs.
2. We demonstrated our proposed algorithm in a non-IID setting on MNIST, CIFAR-10 and CrisisIBD datasets. We showed that our solution could outperform existing methods in terms of global model accuracy with shorter training latency and lower communication costs.
3. We explored the influence of balanced-MixUp in the FL system. In most settings, balanced-MixUp could mitigate weight divergence and improve convergence speed.

The rest of the paper is organized as follows. Section 2 describes the related work. Section 3 discusses the system model and problem formulation. Section 4 presents the proposed solution. Section 5 shows the experimental setup, followed by the simulation results and discussion. Section 6 concludes the paper and outlines future research directions.

2. Related Work

This section reviews existing works on FL and DRL-based FL to provide insights into the current trend in FL. Then, we discuss the limitation of each algorithm. Lastly, we also elaborate on the weight divergence problem in FL, which is a common problem faced by all FL algorithms.

2.1. Federated Learning and Deep Reinforcement Learning

Some of the commonly used FL algorithms include FedAvg [4], FedProx [23] and FedNova [17]. FedAvg was the first practical implementation of FL. In each communication round, the server sends the global model to N randomly selected clients. Each client trains the model using its local dataset. Then, each client uploads its locally trained model to the server, where the server averages the received local models' weights as the new global model. It has since become the de facto approach for FL and is widely used in various applications [3,5–7]. FedProx presents a reparameterization of FedAvg by introducing an additional L_2 regularization term in the local objective function. The regularization term limits the distance between the local and global models, preventing local updates from diverging from global optima. A hyperparameter μ controls the weight of the regularization term. Overall, the modification can be easily performed on the existing FedAvg algorithm while improving model accuracy on non-IID datasets. However, it introduces additional computing overhead, leading to longer training latency. On the other hand, FedNova improves FedAvg in the aggregation stage. It allows each $n \in N$ client to conduct a different number of local steps. This allows clients with higher computing resources to conduct more training while waiting for others to complete training. To ensure that the global updates are not biased, each local update is normalized and scaled according to the number of local steps conducted before they are averaged into the new global model. FedNova introduces negligible computation overhead compared to FedAvg while handling computing resources heterogeneity in FL systems. However, all the above algorithms are limited to handling statistical heterogeneity (non-IID dataset) and computing resources heterogeneity. Other heuristic algorithms have been proposed to optimize client selection in FL systems with heterogeneous network and/or energy resources [24,25]. However, heuristic solutions could only deliver sub-optimal performance since they often rely on qualitative analysis without exploring the optimal performance [14].

DRL has been widely applied to solve optimization problems involving complex sequential decision-making, such as playing Atari games [26], multiplayer games [27] and chess [28]. Since an FL procedure can be modelled as an MDP, it can also be optimized using DRL. FAVOR is one of the first research works to optimize FL using DRL [21]. They observed an implicit connection between the distribution of a local client dataset and the model weights trained on those data. Using the model weights collected from each participating client, a DRL agent can learn to select suitable clients for the next round of training. After proving DRL success in FL optimization, multiple studies [9,10,14,19,20,29] have exploited DRL in FL resource allocation problems. For instance, [9,10,14,19] used DRL to jointly optimize computing and network resources in an FL framework while retaining the global model's accuracy. These studies employed a DRL-based client selection policy or early client termination policy. Such a policy is responsible for selecting the best subset of clients for each round of training by optimizing the tradeoff between model accuracy and resource allocation. On the other hand, [29] optimized only the network resources by quantizing the model weights before uploading them to the FL server. However, none of the DRL-based FL frameworks described above allow dynamic local epoch adjustment. These frameworks fixed the same local epochs for all clients, disregarding their computing cost and training latency. With dynamic local epoch adjustment, clients with higher computing resources can conduct more training epochs. On the contrary, clients with limited computing power can train with fewer epochs.

Table 1 summarizes the key features of the aforementioned FL and DRL-based FL algorithms. In short, we noticed an ongoing trend of utilizing DRL to optimize the computing and network resources in the FL framework. Most of them relied only on client selection or early client termination techniques. As a result, such a method often rejects clients with limited computing power to prevent these devices from dragging the overall FL training latency. Even when such devices are selected, those with stronger computing power will finish training earlier and remain idle while waiting for the slower ones to complete training. However, these devices may contain crucial training data that is essential for FL convergence. Ideally, these devices should participate in FL training but with a lower local training epoch and vice versa. To the best of our knowledge, no DRL-based FL algorithms adopt DRL for automated local epoch adjustment. On the other hand, existing FL algorithms such as FedNova rely on manual adjustment to set devices with stronger computing power with a higher local epoch. Hence, an exciting potential exists for incorporating DRL-based dynamic local epoch adjustments for automated calibration.

Table 1. Features of existing FL and DRL-based FL algorithms.

Method	Resource Optimization	Client Selection	Local Epoch Adjustment
FedAvg [4]	-	Random	Fixed
FedProx [23]	-	Random	Fixed
FedNova [17]	Computing	Random	Flexible
FAVOR [21]	Computing	DRL Agent	Fixed
TP-DDPG [9]	Computing + Communication	DRL Agent	Fixed
Research work [10]	Computing + Communication	DRL Agent	Fixed
FedMarl [14]	Computing + Communication	DRL Agent	Fixed
Research work [19]	Computing + Communication	DRL Agent	Fixed
Research work [20]	Computing + Communication	Random	Fixed
Research work [29]	Communication	Random	Fixed
Proposed FedDdrl	Computing + Communication	DRL Agent	DRL Agent

2.2. Weight Divergence in Federated Learning

Weight divergence is the difference between FL model weights $\boldsymbol{w}_t^{FL}$ and CL model weights $\boldsymbol{w}_t^{CL}$. An ideal level of weight divergence in FL could exploit the rich decentralized data, resulting in a better performance. For instance, FL outperforms its CL counterpart in various applications, including drug discovery [30], disaster classification [7] and autonomous driving object detection [31]. However, in extreme non-IID cases where the local client data distribution p^k is far from the global data distribution p, the highly diverged local weight updates could lead to bad aggregated solutions which are far from the global optimum solution. This is especially true in IoT networks, where each IoT device has a unique data distribution and can be considered non-IID [1]. According to [16], the main source of weight divergence is the earth mover's distance (EMD) between p^k and p, denoted as $\sum_{i=1}^{n_c} \parallel p^k(y=i) - p(y=i) \parallel$, where n_c denotes the total number of classes. In general, weight divergence is inevitable since p^k and p are almost guaranteed to be different in a real-life setting. Figure 1 shows an example of weight divergence between FedAvg and CL models.

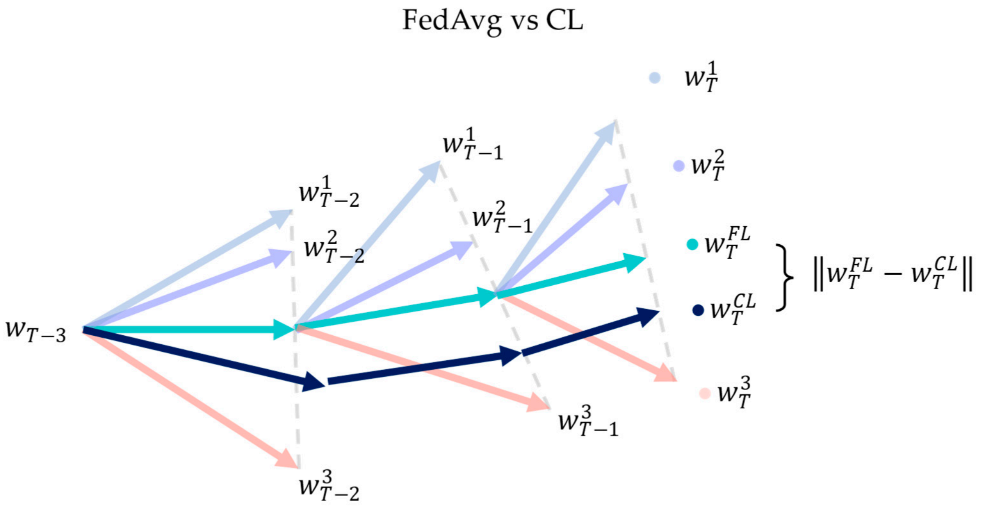

Figure 1. Weight divergence between $\boldsymbol{w}_t^{FL}$ and $\boldsymbol{w}_t^{CL}$ is inevitable even if both models have the same initialization weights.

Generally, only a fraction of the total clients is selected for training per communication round, t, to reduce the total communication cost in FL. Let C denote the total number of participating clients per round. When C is low, it is difficult to ensure the sampled data resemble the global data distribution. This also leads to high EMD between p^k and p, which again contributes to the divergence of $\boldsymbol{w}_t^{FL}$ from $\boldsymbol{w}_t^{CL}$. Let A_t be the accuracy of the global model at communication round $t \in T$. Figure 2 shows the accuracy curve of FedAvg models trained on a non-IID CIFAR-10 dataset using $C = 5$ and $C = 10$. First, the average accuracy of the global model after $t = 15$ communication round was 62.73% and 72.79% for $C = 5$ and $C = 10$, respectively. Additionally, the fluctuation and standard deviation of the accuracy curve were higher when $C = 5$ as compared to $C = 10$. It is shown that FedAvg (or FL in general) had inferior performance when the number of participating clients per round is low.

Recent studies have contributed various solutions to mitigate weight divergence. For instance, the FedProx [23] mentioned earlier adds a regularization term to the local subproblem to prevent the local updates from diverging away from the global FL model. This method, in turn, hopes to ensure the aggregated global FL model weights $\boldsymbol{w}_t^{FL}$ are close to $\boldsymbol{w}_t^{CL}$. Albeit effective, FedProx requires higher computing costs and a longer training time [32]. On the other hand, [16] proposed partial global sharing of local data to reduce EMD between client data distribution and the global populations. However, this induces high communication costs for data sharing and raises privacy concerns.

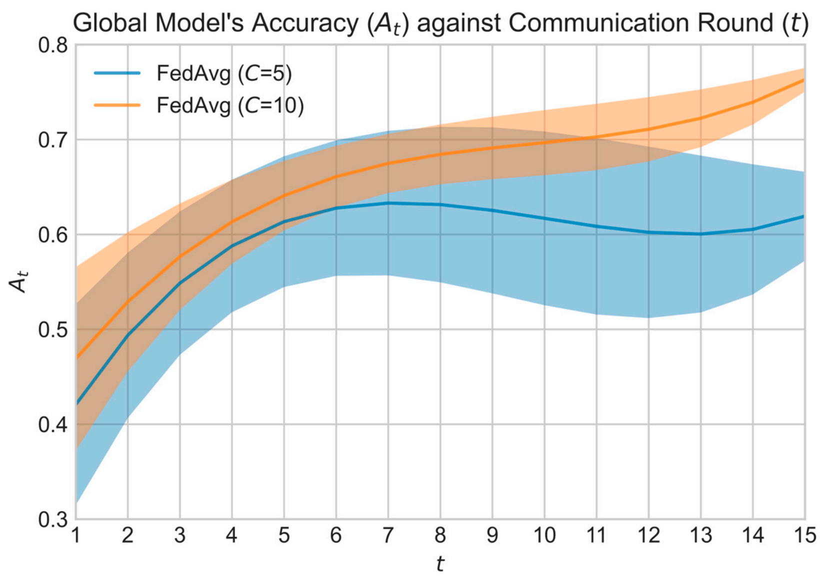

Figure 2. The global model's accuracy curve for $C = 5$ and $C = 10$.

Meanwhile, methods that employ adaptive client selection or early client termination (i.e., FedMarl) aim to tackle weight divergence via careful client selection. For each communication round, FedMarl will only select a subset of the C clients for training. Ideally, only the selected clients are useful for training, while the rest are not. Effectively, this means that C is not constant for each communication round t. However, a lower C may lead to less steady convergence based on Figure 2. Thus, FedMarl is expected to handle the careful dropping of clients considering the EMD between global and local populations while taking care of other optimizing objectives, such as the training latency and communication cost of each client. Dropping the wrong clients may lead to large weight divergence, as shown in Figure 3.

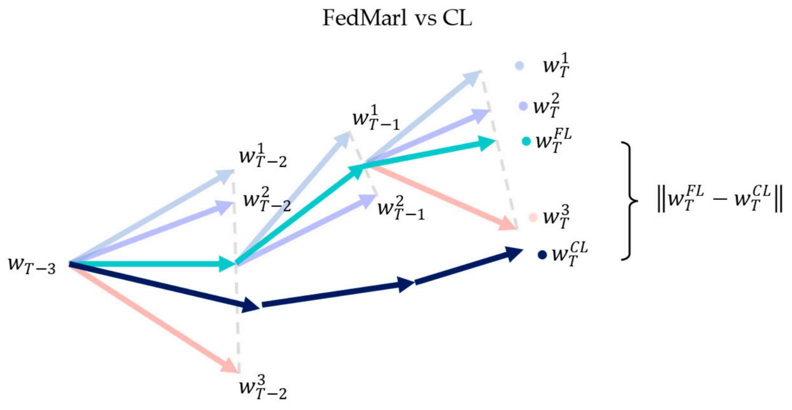

Figure 3. Ineffective client dropping by FedMarl will lead to large weight divergence. Client 3 (denoted by peach) is dropped from training at communication round $T-2$. This causes the aggregated FL weights w_t^{FL} (denoted by green) to converge toward Client 1 and 2 while diverging away from the w_t^{CL}.

3. System Model and Problem Formulation

In this section, we present the system model for our FL system and discuss the problem formulation. Our commonly used symbols are listed in Table 2 for ease of reference.

Table 2. List of key variables defined in the system model.

Notation	Definition
t	Index of communication round
K	The total number of client devices (IoT devices)
N	The total number of client devices selected at each communication round
n	Index of selected IoT devices at communication round t
$H^b_{t,n}$	Model broadcasting latency from server to client n
$H^p_{t,n}$	Probing training latency for client n
$H^m_{t,n}$	Metadata uploading latency from client n to server
$H^u_{t,n}$	Model uploading latency from client n to server
H_t	Complete training latency for communication round t
B^t_n	Communication cost of client n
B_t	Total communication cost for communication round t
A_t	Accuracy of the global model at communication round t
ΔA_t	Global model's accuracy improvement
$\boldsymbol{\phi}_t$	Client selection matrix at communication round t
$\boldsymbol{E}_t$	Local epoch count matrix at communication round t

3.1. System Model

We considered an FL system with K number of client devices. At communication round $t \in T$, N number of clients were randomly selected from the K number of clients. Each communication round consisted of four phases, which are: (1) model broadcasting, (2) probing training, (3) client dropping and (4) completion of training.

1. Model broadcasting: If $t = 1$, the FL server will initialize a global model, whereas at $t \geq 2$, the FL server will collect the client models trained at round $t - 1$ and aggregate them into a new global model. Then, the FL server will broadcast the global model to N randomly selected clients.
2. Probing training: Each selected client $n \in N$ will perform one epoch of local training called probing training. The purpose of probing training is to acquire the metadata of each client. The metadata consist of the client's states, which will be fed to the DRL agents for adaptive early client termination and local epoch adjustment. The details of the client states will be defined later together with the specification of the DRL agents. After probing training, each client will upload its metadata to the server and proceed to the next phase.
3. Early client termination: Based on the collected client states, the DRL agents at the FL server will drop non-essential clients to reduce total latency H_t and total communication cost B_t for round t. The decision made by DRL agents will be sent to each client.
4. Completion of training: Only the remaining C clients that are not dropped by the DRL agent will resume training. Each client n will complete the remaining local training until E^t_n epochs are reached. Each locally trained model will be uploaded to the FL server for model aggregation.

Let $H^p_{t,n}$ denote the probing training latency for client $n \in N$ at round $t \in T$. Let $H^c_{t,n}$ be the complete local training latency for client n at round t, while $H^u_{t,n}$ denotes the time taken for client n to upload its local model to the FL server. Let $\phi^t_n \in \{0, 1\}$ denote if client n is selected by the DRL agent to complete full local training. The total processing latency H_t and the total communication cost B_t at communication round t can be expressed by Equations (1) and (2):

$$H_t = \max_{1 \leq n \leq N} \left(H^c_{t,n} + H^u_{t,n}\right) a^t_n \tag{1}$$

$$B_t = \sum_{n=1}^{N} B^t_n \phi^t_n \tag{2}$$

Figure 4 depicts the system model for the proposed FL protocol. Note that the model broadcasting latency $H^b_{t,n}$ was not included into H_t since it is not part of the FL optimization problem. Additionally, the time latency to upload client metadata to the server, $H^m_{t,n}$, was ignored. This is because the metadata file size was only 278 bytes, while even a lightweight MobileNetV2 file is 24.5 megabytes. Thus, $H^m_{t,n}$ is negligible since $H^m_{t,n} \ll H^u_{t,n}$.

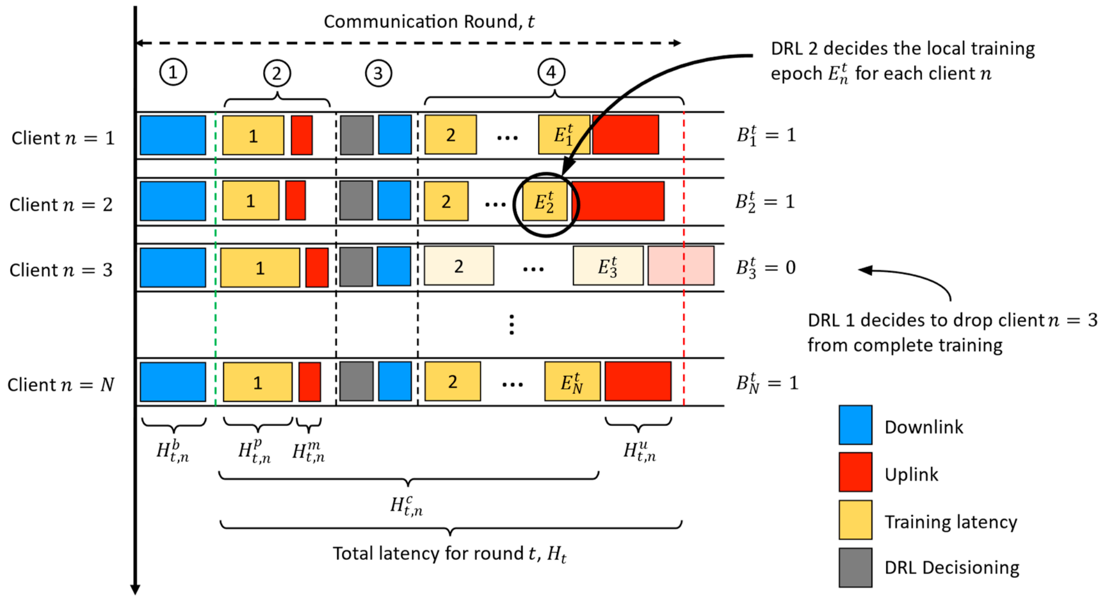

Figure 4. The proposed FL protocol's system model consists of four phases in each communication round: (1) model broadcasting, (2) probing training, (3) early client termination and (4) completion of training.

3.2. Problem Formulation

Our objective was to maximize the cumulative A_t improvement while minimizing the H_t and B_t. Let $\boldsymbol{\phi}_t = [\phi^t_n]$ and $\boldsymbol{E}_t = [E^t_n]$ be a $T \times N$ matrix for client termination and local epoch adjustment decided by the DRL agents, respectively. We formulated the problem as a weighted sum optimization problem, as formulated below:

$$\max_{\boldsymbol{\phi}_t, \boldsymbol{E}_t} \mathbb{E}\left[\sum_{t=1}^{T} w_1[U(A_t) - U(A_{t-1})] - (w_2 B_t + w_3 H_t)\right] \tag{3}$$

where w_1, w_2 and w_3 are the weights to control the importance of each objective. To ensure A_t can improve even if it is small near the end of the FL process, a utility function denoted as $U(\cdot)$ was used to reshape the A_t of the global model. In FedMarl, $U(A_t)$ is defined in Equation (4):

$$U(A_t) = \frac{20}{1 + e^{0.35(1-A_t)}} - 10 \tag{4}$$

One problem with the original $U(A_t)$ is that it only tells us the transformed value of A_t. The entire $w_1[U(A_t) - U(A_{t-1})]$ can be reparametrized into a single $w_1 U(\Delta A_t)$ expression, which could directly tell us the gain/penalty for ΔA_t. First, the $U(A_t)$ equation is simplified in the given range $0 \leq A_t \leq 1$ since A_t is bounded between 0 and 100%. In this range, $U(A_t)$ can be approximated as a straight line, as shown in Figure 5.

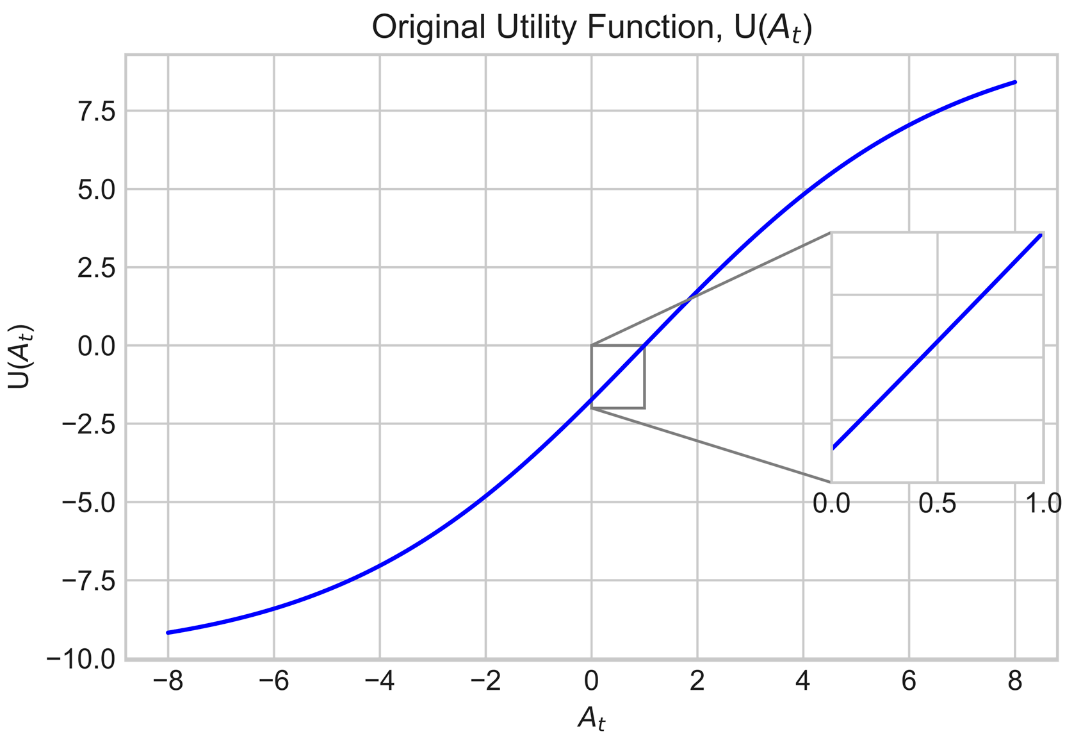

Figure 5. The original utility function $U(A_t)$ can be approximated as a straight line when A_t falls in the range [0, 1].

To approximate $U(A_t)$ as a straight line in the given range, the gradient and y-intercept of the graph are required. The gradient is denoted as $U'(A_t)$, which is the first derivative of the $U(A_t)$ function. $U'(A_t)$ can be written as in Equation (5):

$$U'(A_t) = \frac{7e^{0.35(1-A_t)}}{\left(1+e^{0.35(1-A_t)}\right)^2} \tag{5}$$

The mean of the gradient, $\overline{U'(A_t)}$, within the range can be formulated as in Equation (6):

$$\begin{aligned} &\overline{U'(A_t)} \\ &= \tfrac{1}{1-0}\int_0^1 U'(A_t)dt \\ &= \int_0^1 \tfrac{7e^{0.35(1-A_t)}}{\left(1+e^{0.35(1-A_t)}\right)^2}\,dt \\ &= 1.732 \end{aligned} \tag{6}$$

The y-intercept of $U(A_t)$, denoted as $U(A_t = 0)$, can be written as:

$$\begin{aligned} &U(A_t = 0) \\ &= \tfrac{20}{1+e^{0.35(1-0)}} - 10 \\ &= -1.732 \end{aligned} \tag{7}$$

Hence, $U(A_t)$ can be simplified into:

$$\begin{aligned} &(A_t) \\ &= \overline{U'(A_t)}\; A_t + U(A_t = 0) \\ &= 1.732\; A_t - 1.732 \end{aligned} \tag{8}$$

Thus, $U(\Delta A_t)$ can be defined as:

$$\begin{aligned}&U(\Delta A_t)\\&\cong U(A_t) - U(A_{t-1})\\&= (1.732\ A_t - 1.732) - (1.732\ A_{t-1} - 1.732)\\&= 1.732(\ A_t - A_{t-1})\\&= 1.732\ \Delta A_t\end{aligned} \tag{9}$$

$U(\Delta A_t)$ is more analyzable than $U(A_t) - U(A_{t-1})$ since the two expressions have been collapsed into one equation. At this end, we can define our optimization problem as:

$$\max_{\boldsymbol{\phi}_t, \boldsymbol{E}_t} \mathbb{E}\left[\sum_{t=1}^{T} w_1 U(\Delta A_t) - (w_2 B_t + w_3 H_t)\right] \tag{10a}$$

s.t.

$$\tag{10b}$$

$$\mathbb{E}\left(w_3 \sum_{t=1}^{T} H_t\right) > \Omega_1 \mathbb{E}\left(w_2 \sum_{t=1}^{T} B_t\right) \tag{10c}$$

$$w_1 U(\Delta A_t = 0.01) > \Omega_2 \mathbb{E}(w_2 B_t + w_3 H_t) \tag{10d}$$

$$\mathbb{E}\left(w_1 \sum_{t=1}^{T} U(\Delta A_t)\right) > \Omega_3 \left(w_2 \sum_{t=1}^{T} B_t + w_3 \sum_{t=1}^{T} H_t\right) \tag{10e}$$

where (10b–e) are the constraints for our optimization problem. Constraint (10b) is to make sure the sign of $U(\Delta A_t)$, B_t and H_t are not inverted. Meanwhile, constraint (10c) is to control the ratio of $\sum_{t=1}^{T} B_t$ to $\sum_{t=1}^{T} H_t$. Furthermore, constraint (10d) is to make sure $w_1 U(\Delta A_t)$ gain will not be outweighed $(w_2 B_t + w_3 H_t)$ penalties when ΔA_t is as small as 0.01. Lastly, constraint (10e) makes sure $\sum_{t=1}^{T} w_1 U(\Delta A_t)$ is at least Ω_3 greater than the penalty terms. Note that (10c-e) are additional constraints that are not imposed on the original FedMarl optimization problem.

In FedMarl, the w_1, w_2 and w_3 are treated as hyperparameters. FL engineers have to manually adjust the weightage of each objective until the desired outcome is achieved. However, the weightage w_1, w_2 and w_3 does not directly translate to the weightage of each objective $\sum_{t=1}^{T} U(\Delta A_t)$, $\sum_{t=1}^{T} B_t$ and $\sum_{t=1}^{T} H_t$. For instance, the ratio of $w_2 B_t$ to $w_3 H_t$ does not directly equate to the ratio of $w_2 \sum_{t=1}^{T} B_t$ to $w_3 \sum_{t=1}^{T} H_t$. This is because the values of H_t and B_t are instantaneous and stochastic, which means that the ratio of $w_3 H_t$ to $w_2 B_t$ at two different t is most likely different. On the other hand, $\mathbb{E}\left(\sum_{t=1}^{T} H_t\right)$ and $\mathbb{E}\left(w_2 \sum_{t=1}^{T} B_t\right)$ are more consistent. Taking the ratio of these two components is more reliable.

We can find the best w_1, w_2 and w_3 by setting the desired Ω_1, Ω_2 and Ω_3. We set $\Omega_1 = 0.2$, $\Omega_2 = 0.3$ and $\Omega_3 = 1.0$. We needed to run one iteration of FL using FedAvg to get the traces value of ΔA_t, B_t and H_t for $t \in T$ since these values are dependant on the target IoT environment setup. Based on the traces value, we could follow Algorithm 1 to acquire the suitable w_1, w_2 and w_3. In our experiment setup, we found the desired hyperparameters to be (w_1 = 2.9, w_2 = 0.1 and w_3 = 0.2).

Algorithm 1 Search for the best w_1, w_2 and w_3

```
1. Input:   Set $\Omega_1 = 0.2, \Omega_2 = 0.3, \Omega_3 = 1.0$
2. Output:  The best $w_1, w_2, w_3$
3: Run one complete iteration of FedAvg with $T = 15$ communication rounds and record the traces value of $\Delta A_t$, $B_t$ and $H_t$ for $t \in T$.
4: Initialize an empty set $cache = \{\}$ to store all $((w_1, w_2, w_3), R)$ that satisfied constraints (10c-e)$R$ is the weighted-sum optimization goal $w_1 U(\Delta A_t) - (w_2 B_t + w_3 H_t)$
5: for $w_1 = 0,\ 0.1,\ \ldots 3.0$ do
6:     for $w_2 = 0,\ 0.1,\ \ldots 1.0$ do
7:         for $w_3 = 0,\ 0.1,\ \ldots 1.0$ do
8:             Compute $\mathbb{E}\left[w_1 \sum_{t=1}^{T} U(\Delta A_t)\right], \mathbb{E}\left[w_2 \sum_{t=1}^{T} B_t\right], \mathbb{E}\left[w_3 \sum_{t=1}^{T} H_t\right]$, $\mathbb{E}(w_2 B_t + w_3 H_t)$ based on the recorded traces value, where we assume $\mathbb{E}(x) \triangleq x$
9:             if (10c-e) are satisfied:
10:                Compute $R = \sum_{t=1}^{T} w_1 U(\Delta A_t) - (w_2 B_t + w_3 H_t)$ from the traces value
11:                Record $((w_1, w_2, w_3), R)$ in cache
12:            end if
13:        end for
14:    end for
15. end for
16: From cache, find out which combination of $(w_1, w_2, w_3)$ results in the smallest $R$. This can be treated as finding the worse-case max $\mathbb{E}[R]$.
17: return$w_1, w_2, w_3$
```

4. Proposed Method

We propose FedDdrl, which exploits two DRL policy networks for FL optimization. Specifically, we adopted VDNs as the DRL policy networks for FedDdrl. We elaborate in detail on how we formulated the problem as MDP, including the design of state space, action space and reward of the algorithm. In addition, we also exploited the recently proposed balanced-MixUp to mitigate the impact of weight divergence and speed up the FL convergence speed.

4.1. Deep Reinforcement Learning for Federated Learning Optimization

The proposed optimization problem in Equation (10a–e) is a 0–1 Multidimensional Knapsack Problem (MKP). The items to be put in knapsacks are the client devices n with complete training latency $H^c_{t,n}$, model uploading latency $H^u_{t,n}$, communication cost B^t_n and data size D_n. The total capacity of the knapsack equals the total communication cost $B_t = \sum_n^N B^t_n a^t_n$, where a^t_n is the binary indicator of item (client) n. When a^t_n is set to 1, item (client) n is selected. Otherwise, a^t_n is set to 0. The total weight of the knapsack has a lower bound which has to fulfil the minimum requirement of accuracy constraint(10d). Our goal is to select a subset of clients C $(1 < C \le N)$ for complete training in each communication round to maximize the total accuracy gain $\sum_{t=1}^T \Delta A_t$ while minimizing the total latency $\sum_{t=1}^T H_t$ and total communication cost $\sum_{t=1}^T B_t$ of the entire FL training. Thus, the proposed optimization is NP-hard.

To solve problem (10), our FedDdrl algorithm adopted a double DRL framework for our optimization problem. Specifically, we formulated the DRL policy network for both tasks using a multi-agent reinforcement learning (MARL) approach. In particular, VDN has proven itself in the recent literature [33] to be a promising candidate for MARL problems. A VDN network consists of N agents, in which each agent $n \in N$ uses a deep neural network (DNN) parametrized with θ to implement the Q-function $Q^\theta_n(s, a) = \mathbb{E}[R_t | s = s^t_n, \text{a} = a^t_n]$. At each timestep t, each agent n observes its states s^t_n and selects the optimal action a^t_n with the maximum Q-value. Let $\boldsymbol{s}_t = \{\boldsymbol{s}^t_n\}$ and $\boldsymbol{a}_t = \{a^t_n\}$ represent the states and actions collected from all agents $n \in N$ at timestep t, respectively. The joint Q-function $Q_{tot}(\cdot)$ for the multi-agent VDN system can be represented by the elementwise summation of all the individual Q-functions, where $Q_{tot}(\boldsymbol{s}_t, \boldsymbol{a}_t) = \sum_n Q_n^{\theta}(s_n^t, a_n^t)$. In FedDdrl, we set each agent in both VDN as a simple two-layer multi-layer perceptron (MLP), which is cheap to implement. All MLPs in each VDN share their weights to prevent the lazy agent problem [33].

As illustrated in Figure 6, the first VDN network takes the client states $\boldsymbol{s}_t$ (which will be detailed later) to obtain the optimal client termination matrix $\boldsymbol{\phi}_t$. The second VDN network takes the same client states $\boldsymbol{s}_t$ to obtain the optimal local training epoch per client $\boldsymbol{E}_t$.

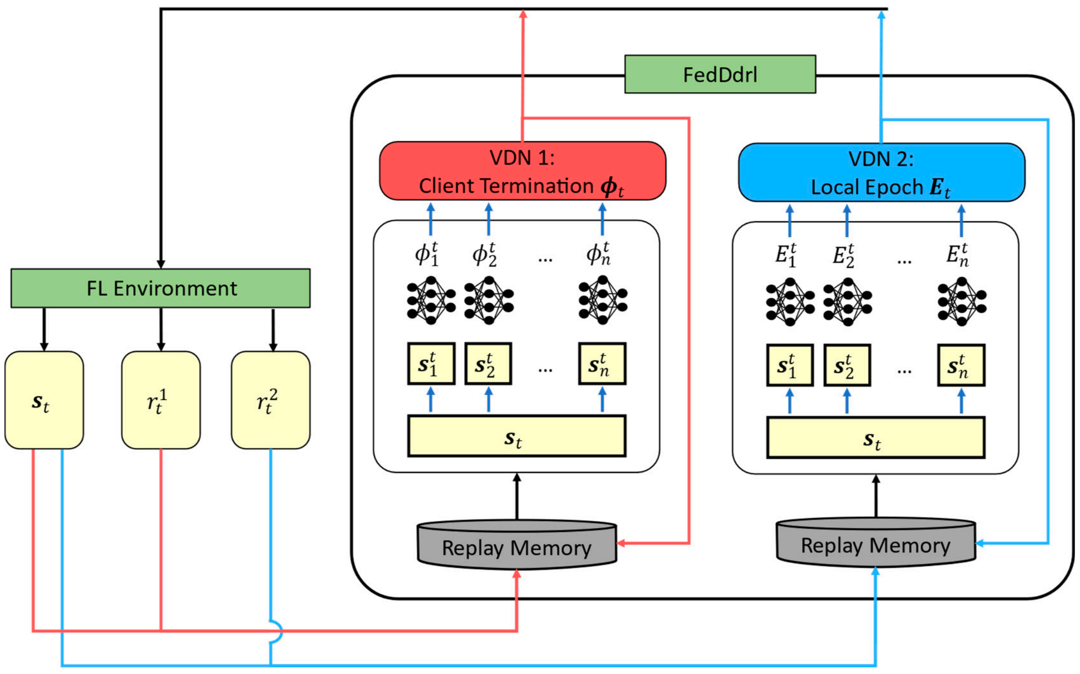
FedDdrl
VDN 1: Client Termination $\boldsymbol{\phi}_t$
VDN 2: Local Epoch $\boldsymbol{E}_t$
ϕ_1^t ϕ_2^t ... ϕ_n^t
E_1^t E_2^t ... E_n^t
s_1^t s_2^t ... s_n^t
$\boldsymbol{s}_t$
Replay Memory
Replay Memory
FL Environment
$\boldsymbol{s}_t$
r_t^1
r_t^2

Figure 6. Structure of the proposed FedDdrl algorithm.

4.1.1. Early Client Termination

Inspired by the work of FedMarl [14], the first VDN network was employed to learn the optimal policy for early client termination matrix $\boldsymbol{\phi}_t$ at round t. We reformulated the problem as an MDP with the following state, action and reward to train a VDN network with $N = 10$ agents.

1. **State $\boldsymbol{s}_t$** State $\boldsymbol{s}_t = \{\boldsymbol{s}_n^t\}$ consisted of the client states for each VDN agent. Each agent n consisted of six components: (i) the probing loss L_n^t, (ii) probing training latencies $\boldsymbol{H}_{t,n}^p$, (iii) model uploading latencies $\boldsymbol{H}_{t,n}^u$, (iv) communication cost from client to server B_n^t, (v) local training dataset size $|D_n|$ and (vi) current communication round index t. The state vector for agent c can be written as Equation (11):

$$\boldsymbol{s}_n^t = \left[L_n^t, \boldsymbol{H}_{t,n}^p, \boldsymbol{H}_{t,n}^u, B_n^t, |D_n|, t\right] \tag{11}$$

It is noteworthy that since each agent in the VDN only has access to its own local observation instead of the full observed environment, the policy has to incorporate past agent observations from history [33]. Thus, the historical values of probing latencies $\boldsymbol{H}_{t,n}^p = \left[H_{t-\Delta T_p,n}^p, \ldots, H_{t,n}^p\right]$ and model uploading latencies $\boldsymbol{H}_{t,n}^u = \left[H_{t-\Delta T_u-1,n}^u, \ldots, H_{t-1,n}^u\right]$ were included in the state vector to mitigate the limitation of local observation. Note that ΔT_p and ΔT_u are the sizes of the historical information of probing latencies and model uploading latencies, respectively.

2. **Action $\boldsymbol{\phi}_t$**: Action $\boldsymbol{\phi}_t = \{\phi_n^t\}$ comprised the client termination decision for each VDN agent. The action space for client termination was $\boldsymbol{\phi}_n^t = \{0,\ 1\}$, where 0 indicates the termination of the client and 1 indicates the client is selected for complete training.
3. **Reward r_t^1**: A vanilla reward for VDN 1, denoted as r_t^1, can be adopted from the FL optimization problem as described in Equation (12):

$$r_t^1 = w_1 U(\Delta A_t) - (w_2 B_t + w_3 H_t) \tag{12}$$

where the system is rewarded with accuracy improvement ΔA_t and penalties for B_t and H_t. However, Equation (12) has one obvious limitation. When $\Delta A_t \to 0$, the $\lim_{\Delta A_t \to 0} w_1 U(\Delta A_t) = 0$, regardless of the magnitude of w_1. If $w_1 U(\Delta A_t) \to 0$, the reward $r_t^1 \approx -\left(\overline{w_2 B_t + w_3 H_t}\right)$. This causes the optimization problem to diverge from improving accuracy with the constraint of B_t and H_t to merely the reduction of B_t and H_t. To show the severeness of this problem, we trained the VDN agents using the r_t^1 as defined by Equation (12). Let $\mathbb{E}[w_1 U(\Delta A_t)]$ and $\mathbb{E}[w_2 B_t + w_3 H_t]$ denote the expected values of accuracy improvement ΔA_t and penalties $(w_2 B_t + w_3 H_t)$, respectively. For MNIST dataset, the expected values of both components for the last $\mathrm{R} = 5$ communication rounds can be computed in Equations (13) and (14):

$$\begin{aligned} &\mathbb{E}[w_1 U(\Delta A_t)]|_{t=\{T-R,\ T-R-1,\ \ldots T\}} \\ &= \tfrac{1}{5} \sum_{t=T-5}^{T} w_1 U(\Delta A_t) \\ &= 0.0273 \end{aligned} \tag{13}$$

$$\begin{aligned} &\mathbb{E}[w_2 B_t + w_3 H_t]|_{t=\{T-R,\ T-R-1,\ \ldots T\}} \\ &= \tfrac{1}{5} \sum_{t=T-5}^{T} (w_2 B_t + w_3 H_t) \\ &= 0.279 \end{aligned} \tag{14}$$

It is observed that $\mathbb{E}[w_1 U(\Delta A_t)] \ll \mathbb{E}[w_2 B_t + w_3 H_t]$ for the last five communication rounds. This is because as training approach the end, the accuracy improvement is often smaller compared to the earlier stage. Consequently, the VDN agents start to terminate more clients from complete training, giving way to the reduction of B_t and H_t. To make sure the agents are motivated to learn even when $\Delta A_t \to 0$, we can introduce a bias term b to r_t^1. Let $b = \frac{3}{10}\ \mathbb{E}[w_2 B_t + w_3 H_t]$. Hence, the reward function r_t^1 can be reformulated as shown in Equation (15):

$$r_t^1 = \begin{cases} r_t + b, & \Delta A_t > 0 \\ r_t, & \Delta A_t \le 0 \end{cases},\ r_t = w_1 U(\Delta A_t) - (w_2 B_t + w_3 H_t) \tag{15}$$

Note that we only added the bias term b to the reward r_t when $\Delta A_t > 0$ since it is intended to encourage accuracy improvement. We did not subtract the bias term b from the reward r_t when $\Delta A_t \le 0$ since the penalty terms are sufficient to penalize the inferior actions.

4.1.2. Local Epoch Adjustments

The second DRL network was employed to learn the optimal policy for local epoch adjustments $\boldsymbol{E}_t$ at round t. A VDN algorithm with $N = 10$ agents can be formulated by defining the state, action and reward as follows:

1. **State $\boldsymbol{s}_t$**: The second VDN shared the same state in Equation (11) since both VDNs required the same local observation for decision making.
2. **Action $\boldsymbol{E}_t$**: Action $\boldsymbol{E}_t = \{E_n^t\}$ comprises the local epoch counts for each VDN agent. The action space is $E_n^t = \{3, 5,\ 7\}$. This action aims to exploit client devices with stronger computation power for more training epochs and vice versa.

3. **Reward r_t^2**: We adopted Equation (15) as the starting point for the reward function for VDN 2. However, the communication cost B_t was not part of the optimizing objectives of VDN 2 since local epoch adjustment is only bounded by the H_t constraint. Hence, the reward function r_t^2 for this VDN networks can ignore the B_t penalty. As such, the r_t^2 can be defined in Equation (16):

$$r_t^2 = \begin{cases} r_t + b, & \Delta A_t > 0 \\ r_t, & \Delta A_t \le 0 \end{cases}, \ where\ r_t = w_1 U(\Delta A_t) - w_3 H_t \tag{16}$$

where we used the same bias term b from (15) for the simplicity's sake.

As the training converges, VDN 1 will deliver the optimal client selection, and VDN 2 will impart the optimal local epoch number for each client. The overall algorithm for solving the problem in Equation (10a–e) is summarized in Algorithm 2.

Algorithm 2 FedDdrl Algorithm

```
1:  Input:   Initialize VDN 1 $Q_{tot}^1$ and its target network $Q_{tot}^{1\,\prime}$ for client selection $\boldsymbol{\phi}_t$ policy
             Initialize VDN 2 $Q_{tot}^2$ and its target network $Q_{tot}^{2\,\prime}$ for local epoch adjustment $\boldsymbol{E}_t$ policy
2:  Output:  Trained $Q_{tot}^1$ and $Q_{tot}^2$ networks
3:  Set $\varepsilon = 1.0$
4:  for Episode $n_{ep} = 1, 2, \ldots, N_{ep}$ do
5:      Reset the FL environment
6:      Initialize a global model $\boldsymbol{w}_0$
7:      for communication round $t = 1, 2, \ldots, T$ do
8:          Randomly select $N$ clients from all $K$ clients
9:          Broadcast the global model $\boldsymbol{w}_t$ to each selected client
10:         for each client $n \in N$ in parallel do
11:             $\boldsymbol{w}_t^n \leftarrow \boldsymbol{w}_t$ ; Copy the global model as each client model
12:             Update the client model $\boldsymbol{w}_t^n$ using the local training dataset $D_n$
13:             Upload client states $\boldsymbol{s}_n^t$ to the FL server
14:         end for
15:         Each agent $n$ in VDN 1 selects the optimal action $\phi_{t,n}^*$= argmax $Q_n^1(\boldsymbol{s}_n^t, \phi_n^t)$ with a $(1-\varepsilon)\times$ 100% probability, else randomly output actions
16:         Each agent $n$ in VDN 2 selects the optimal action $E_{t,n}^*$= argmax $Q_n^2(\boldsymbol{s}_n^t, E_n^t)$ with a $(1-\varepsilon)\times$ 100% probability, else randomly output actions
17:         Send action $\phi_{t,n}^*$ and $E_{t,n}^*$ to each client $n \in N$
18:         for each client $n \in N$ in parallel do
19:             if $\phi_{t,n}^* = 1$:
20:                 Continue updating $\boldsymbol{w}_t^n$ using $D_n$ until $E_{t,n}^*$ is reached
21:                 Return updated $\boldsymbol{w}_t^n$
22:             end if
23:         end for
24:         Aggregate global model $\boldsymbol{w}_{t+1} \leftarrow \sum_{i=1}^{N} \frac{|D_i|}{\sum_{i=1}^{N}|D_i|} \boldsymbol{w}_t^n$ where $i \in \{n|\phi_{t,n}^* = 1\}$
25:         Reward $r_t^1$ and $r_t^2$ are given to $Q_{tot}^1$ and $Q_{tot}^2$ based on $\Delta A_t, B_t, H_t$
26:         $s_t = \{s_n^t\}, \phi_t = \{\phi_n^t\}, \boldsymbol{E}_t = \{E_n^t\}$
27:         Store transitions 1 $[s_t, \phi_t, r_t^1]$ for $Q_{tot}^1$ into memory buffer 1
28:         Store transitions 2 $[s_t, \boldsymbol{E}_t, r_t^2]$ for $Q_{tot}^2$ into memory buffer 2
29:         Sample mini-batches with size $n_b$ from memory buffer to train $Q_{tot}^1$, $Q_{tot}^2$ and $Q_{tot}^{1\,\prime}$, $Q_{tot}^{2\,\prime}$
30:         Decay $\varepsilon$ gradually from 1.0 to 0.1
31:     end for
32: end for
```

4.2. Balanced-MixUp to Mitigate Weight Divergence

In this study, we focused on the non-IID label shift, where the client dataset is heavily skewed to one of the label classes. The huge EMD between the client data distribution p^k and the global distribution p will contribute to weight divergence, deteriorating the training efficiency of FL. Thus, the highly imbalanced client dataset has to be handled wisely. MixUp [34] is a simple yet effective data augmentation technique that could shed some light on this problem.

MixUp extends the training data distribution by linearly interpolating between existing data points, filling the underpopulated areas in the data space. It generates synthetic training data $\left(\hat{x}, \hat{y}\right)$ by simply taking the weighted combination of two random data pairs, (x_i, y_i) and (x_j, y_j), as shown in Equations (17) and (18):

$$\hat{x} = \lambda x_i + (1 - \lambda)x_j \tag{17}$$

$$\hat{y} = \lambda y_i + (1 - \lambda)y_j \tag{18}$$

where $\lambda \sim \text{Beta}(\alpha, \alpha)$, with $\alpha > 0$. Despite its simplicity, MixUp has been proven to improve model calibration and better generalization [35]. Thus, its application has expanded from image and speech classification tasks [34] to other domains, including image segmentation [36] and natural language processing [37,38]. However, a vanilla MixUp works poorly in highly imbalanced datasets [39]. In highly imbalanced datasets, MixUp would end up sampling the data pairs (x_i, y_i) and (x_j, y_j) from the same class for most of the time since the sampling is done randomly.

Some recent studies focused on solving the data imbalance problem for MixUp. In Remix by [39], x_i and x_j are mixed in the same fashion as MixUp, but y_i and y_j are mixed such that the minority class is assigned a higher weight. This method pushes the decision boundaries away from the minority class, balancing the generalization error between the majority and minority classes. Balanced-MixUp is another variation of MixUp, where MixUp is combined with a data-resampling technique to achieve a balanced class distribution [22]. Specifically, balanced-MixUp combines instance-based sampling and class-based sampling for the majority and minority classes, respectively. This ensures that each data pair (x_i, y_i) and (x_j, y_j) always consists of instances from both the majority and minority classes.

We adopted balanced-MixUp as the augmentation into the formulation of our solution to address the class imbalanced problem. To the best of our knowledge, we are the first to integrate balanced-MixUp into FL for weight divergence mitigation. Let (x_M, y_M) and (x_m, y_m) denote the instance pair sampled from the majority and minority classes, respectively. Balanced-MixUp can be expressed as shown in Equations (19) and (20):

$$\hat{x} = \lambda x_M + (1 - \lambda)x_m \tag{19}$$

$$\hat{y} = \lambda y_M + (1 - \lambda)y_m \tag{20}$$

Balanced-MixUp guarantees that each data pair mixing consists of instances from both the majority and minority class. Unlike the original balanced-MixUp where $\lambda \sim \text{Beta}(1, \alpha)$, we adopted $\lambda \sim \text{Beta}(\alpha, \alpha)$ and found it to work better in our study. The best α value may be different depending on the datasets, which will be detailed in the results section.

5. Simulation Results

This study adopted TensorFlow as the deep learning platform. We adopted three datasets for FL benchmarking: MNIST, CIFAR-10 and CrisisIBD [40]. First, MNIST is a relatively simple task under most non-IID settings. It is mainly used to prove that a novel FL algorithm is at least working. In contrast, CIFAR-10 is a challenging dataset in non-IID settings, which is strongly encouraged to be included in FL benchmarking experiments [32]. On the other hand, the CrisisIBD dataset is the benchmark dataset for various real-world disaster-related image classification tasks. In this study, we adopted the disaster classification dataset from the dataset (hereinafter referred to as CrisisIBD) as one of our benchmark datasets. We adopted balanced-MixUp to augment all three datasets. We found the best α value used by balanced-MixUp was 0.05, 0.4 and 0.2 for MNIST, CIFAR-10 and CrisisIBD, respectively. We used all three datasets to train a lightweight

MobileNetV2 [41], which aligned with our goal of developing the FL framework for low-powered IoT devices. All clients adopted $B_n^t = 1 \forall n, t$, similar to the setting in [14].

In this study, we focused on the non-IID label shift as demonstrated in [14,21]. Similarly, we divided each dataset into K clients. For each client, a fraction $\sigma = 0.8$ of the local training dataset was sampled from one random label (which is the majority label), while the rest of the training data were sampled uniformly from the remaining labels (which are the minority labels). We compared our proposed method with FedAvg, FedProx and FedMarl. The first two algorithms were shown to be robust baselines in non-IID label shift [32] and are prebuilt in many existing FL frameworks, including Tensorflow Federated [42] and Intel OpenFL [8]. On the other hand, FedMarl is one of the state-of-the-art FL algorithms [14]. Our FedDdrl aims to outperform all three of the algorithms. The hyperparameters are listed in Table 3.

Table 3. List of hyperparameters.

Parameters	Values
Number of agents in each VDN network, N	10
Total number of clients, K	100
Local training dataset distribution, σ	0.8
Learning rate for VDN network	1×10^{-3}
Target network update interval	5
Number of episodes, N_{ep}	40
Number of clients selected for training in each round, C	10
Default number of local epochs (before adjustment by FedDdrl), E_t	5
Number of communication rounds, T	15
Batch size to update VDN agents, N_b	32
Initial ε-greedy exploration value	1
Final ε-greedy exploration value	0.1
Replay memory size	300
VDN 1 agent (MLP) size	$10 \times 256 \times 256 \times 2$
VDN 2 agent (MLP) size	$10 \times 256 \times 256 \times 3$

Due to limited resources, we only had two hardware devices: (i) an Intel NUC with an Intel core i7-10710U processor with 4.70 GHz and (ii) a workstation equipped with an Intel core i7-10875H processor with 2.30 GHz and NVIDIA RTX 2070 SUPER. In total, this yielded two TensorFlow CPU operators and one TensorFlow GPU operator. However, this was far from enough to simulate a heterogeneous FL environment with $K = 100$ clients if each operator only represents one client. Thus, we carefully devised our experimental setup, as shown in Figure 7.

To simulate an FL environment with $K = 100$ clients (IoT devices) and $C = 10$ selected clients (before early termination), we set up the experiment as shown below:

1. We created $K = 100$ client configurations, each consisting of the (i) client's computing latency per data, (ii) model upload latency and (iii) local dataset identity (ID) number. To closely simulate the heterogeneity of resources in an IoT network as in [14], the computing latency per data in each client configuration can be any of $\{0.25, 0.50, 0.75\}$ seconds, while the model upload latency can be any of $\{1.00, 1.25, 1.75, 2.00\}$ seconds.
2. CPU 1, CPU 2 and GPU simulated three, three and four clients, respectively. The simulated clients represent the $C = 10$ randomly selected clients from the total $K = 100$ clients in each communication round t.
3. In each communication round t, 10 client configurations were randomly sampled out from the configuration pools. The 10 simulated clients (in CPU 1, CPU 2 and GPU) were configured according to the selected client configuration. This entire process (3) is equivalent to the FL process of randomly selected 10 clients with unique local datasets and resources.
4. After step (3), each simulated client proceeded with its training. If the FL algorithm was FedAvg or FedProx, all 10 simulated clients underwent complete training of

$E_t = 5$ local epochs. On the contrary, if the FL algorithm was FedMarl or FedDdrl, only the simulated clients that were not terminated by the FedMarl/FedDdrl completed their local training based on $E^*_{t,n}$= argmax $Q^2_n\left(s^t_n, E^t_n\right)$ by VDN 2.

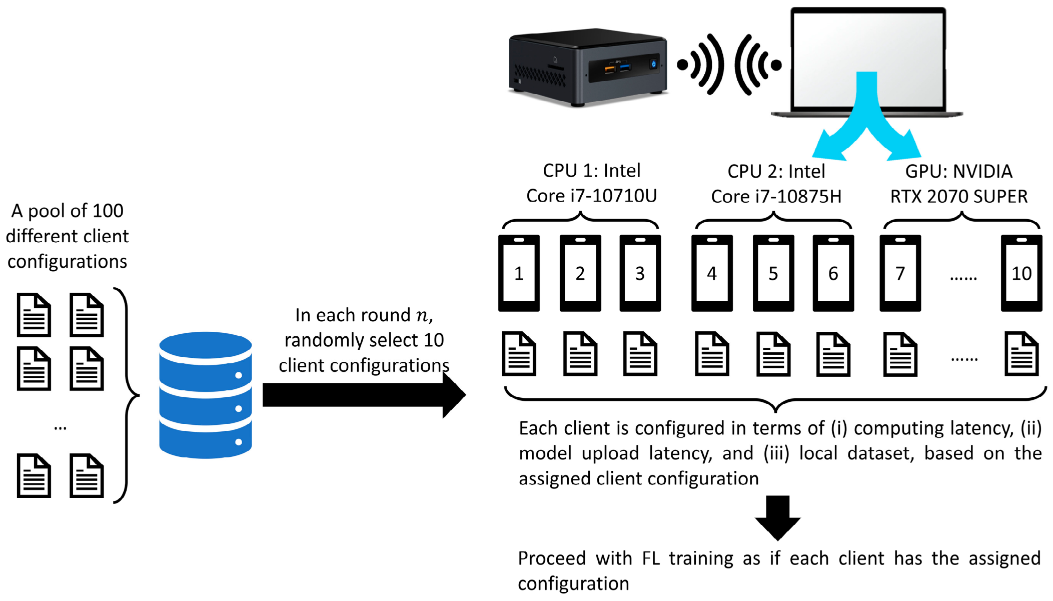
A pool of 100 different client configurations
...
In each round n, randomly select 10 client configurations
CPU 1: Intel Core i7-10710U
CPU 2: Intel Core i7-10875H
GPU: NVIDIA RTX 2070 SUPER
1
2
3
4
5
6
7
......
10
......
Each client is configured in terms of (i) computing latency, (ii) model upload latency, and (iii) local dataset, based on the assigned client configuration
Proceed with FL training as if each client has the assigned configuration

Figure 7. Experiment setup.

5.1. Results and Ablation Study

We compared the performance of FedDdrl with other baselines in all three objectives of the optimization problem, which are the (i) model accuracy, (ii) training latency and (iii) communication efficiency.

5.1.1. Model Accuracy

Table 4 shows the model accuracy trained using each FL setting after $T = 15$ communication rounds. We also conducted an ablation study showing how FedDdrl improves beyond FedMarl.

Table 4. Model accuracy for each FL setting. Bolded indicates the best score, while underlined indicates the second-best score.

Method		MNIST (K=100)	CIFAR-10 (K=100)	CrisisIBD (K=98)
FedAvg		94.6% ± 2.1%	72.8% ± 3.9%	43.2% ± 5.5%
FedAvg with Balanced-MixUp		93.2% ± 2.0%	76.5% ± 1.7%	60.2% ± 1.5%
FedProx (μ = 0.01)		95.6% ± 0.5%	74.5% ± 0.2%	48.1% ± 2.9%
FedProx (μ = 0.01) with Balanced-MixUp		<u>95.4% ± 0.7%</u>	<u>77.8% ± 0.5%</u>	60.7% ± 2.0%
A:	FedMarl (w_1=1.0, w_2=0.1, w_3=0.2)	91.5% ± 1.1%	65.5% ± 2.3%	42.4% ± 3.6%
B:	A + Optimized (w_1=2.9, w_2=0.1, w_3=0.2)	93.2% ± 1.4%	71.7% ± 2.9%	44.4% ± 3.9%
C:	B + Balanced-MixUp	93.3% ± 1.2%	75.0% ± 2.6%	<u>63.3% ± 2.0%</u>
D:	C + Local Epoch Adjustment (FedDdrl)	94.9% ± 1.1%	**78.2% ± 2.4%**	**64.2% ± 1.4%**

Setting A in Table 4 was our implementation of FedMarl with the original hyperparameters (w_1 = 1.0, w_2 = 0.1, w_3 = 0.2). In Setting B, we proved that FedMarl performance could be improved using our hyperparameters (w_1 = 2.9, w_2 = 0.1, w_3 = 0.2). However, we found that its accuracy was still far behind both FedAvg and FedProx. This is because MobileNetV2 is a lightweight model which is easy to overfit [41]. In FedAvg and FedProx, the total number of clients selected for training is always $C = 10$ in each communication round t. During aggregation, there is a sufficient amount of client models overfitted for different classes, which, when aggregated, can generate a regularization effect, thus mitigating the weight divergence caused by overfitting. This is not the case for FedMarl, which does not have a fixed C for each round. We argue that since the original FedMarl was not tested on MobileNetV2, it was able to perform better than FedAvg and FedProx. Applying balanced-MixUp could significantly mitigate this problem, as shown in Setting C. The reasoning on how balanced-MixUp helps weight divergence mitigation is detailed in Section 5.5.

Setting D was our FedDdrl, where we added another VDN for local epoch adjustment. FedDdrl allows client devices to train for more epochs when required and vice versa. This allows FedDdrl to converge faster than FedAvg and FedProx for most cases, even when it is not utilizing all clients at each communication round. FedDdrl outperformed other FL algorithms in both the challenging CIFAR-10 and CrisisIBD datasets, and it was the second-best for MNIST. We suspect FedDdrl is slightly overengineered for an easy task like MNIST. Nevertheless, it was still very robust considering that real-world data is often not as simple as MNIST and is instead more challenging like the CIFAR-10 and CrisisIBD datasets.

5.1.2. Training Latency

Figure 8 shows the normalized training latency for each FL algorithm (with balanced-MixUp) on all three datasets. Our FedDdrl outperformed all three other algorithms in all datasets. This is promising since FedDdrl allows dynamic local adjustment. The FedDdrl will sometimes increase the local epoch from five to seven. However, the extra training latency is balanced when FedDdrl decreases the local epoch from five to three, especially in the early communication round when the MobileNetV2 is still learning lower-level features. FedMarl followed closely behind FedDdrl. This is mainly because both FedDdrl and FedMarl can terminate clients with longer probing latencies. On the other hand, FedAvg had a moderate performance in terms of training latency. It was not as fast as FedMarl and FedDdrl, but it was still significantly faster than FedProx. As expected, FedProx had the longest training latency compared to other FL algorithms, which is aligned with the observation by [32]. This is mainly due to the extra computing cost required to compute the L2 distance between the client and the global model. Hence, applying FedProx in low-powered devices (i.e., IoT devices) with limited computation power is not feasible.

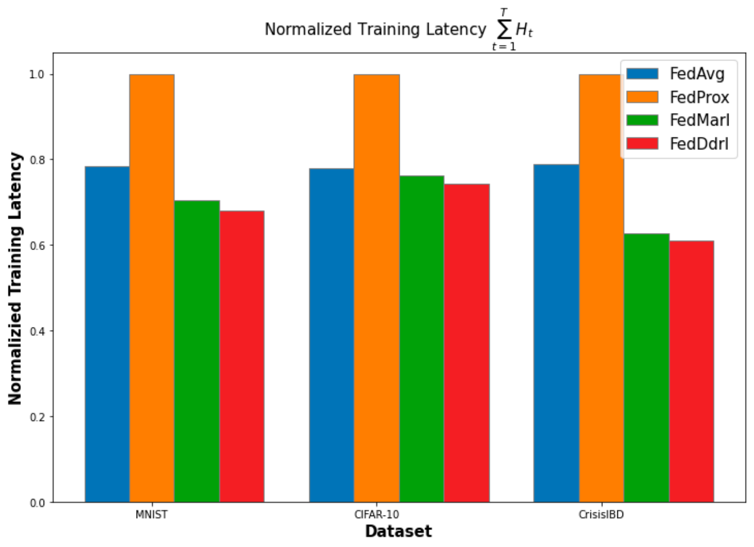

Figure 8. Normalized training latency of each FL algorithm.

5.1.3. Communication Efficiency

Figure 9 shows the normalized communication costs of all FL algorithms in the three datasets. FedDdrl and FedMarl were significantly more efficient than FedAvg and FedProx in total communication costs. There was no clear winner between FedDdrl and FedMarl regarding communication efficiency. However, FedDdrl outperformed FedMarl in CIFAR-10 and CrisisIBD datasets, which are significantly harder tasks compared to MNIST. Thus, we argue that FedDdrl is the best algorithm. On the other hand, FedAvg and FedProx have a fixed number of clients selected in each round. Since we assume $B_n^t = 1 \forall n, t$, both algorithms have the same total communication costs.

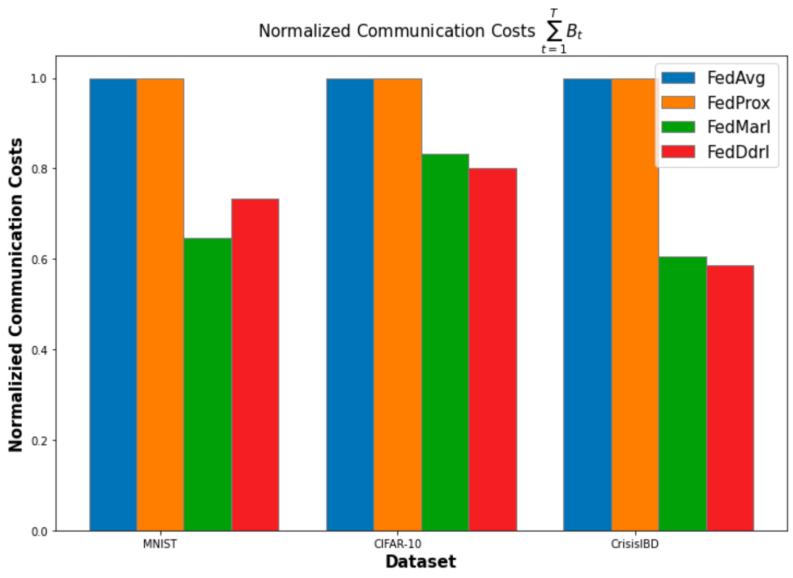

Figure 9. Normalized communication cost of each FL algorithm.

5.2. Strategy Learned by FedDdrl

In this section, we analyze the strategies learned by the FedDdrl algorithm to fully utilized its computing resources while reducing communication costs. Figures 10 and 11 show the early client termination strategy learned by FedDdrl. Blue dots indicate that the client was chosen for complete training, while red dots indicate early client termination.

First, FedDdrl generally picked lesser clients for complete training in the early phase of FL training. From Figures 10 and 11, it is noticed that only three clients were selected for complete training in the first two communication rounds $t = \{1, 2\}$. This is because DNNs usually learn the low-complexity features before learning the higher-complexity features. The former is more robust to noises [43] and can be learned with fewer data [14]. This allows FedDdrl to reduce communication costs by terminating most clients from training in the early phase, where the MobileNetV2 is still learning low-level features. Starting from round $t = 3$ to $t = 6$, the number of clients that underwent complete training increased from 6 to 10. This indicates that MobileNetV2 was beginning to learn higher-level features that require more training data. For the remaining rounds, the number of selected clients was roughly five. Second, FedDdrl preferred clients with a lower probing loss for complete training, which is aligned with the findings in FedMarl [14]. Third, FedDdrl tended to pick clients with shorter probing latency for complete training to reduce the total latency of FL training.

As mentioned earlier, conventional FL training sets the same local epoch E_t^n for all clients, disregarding their computing resources. Hence, one of the contributions of FedDdrl is to learn the optimal strategy to adjust the local epoch count for each client dynamically. In Figure 12, we plotted the local epoch count corresponding to each selected client from the scenario in Figure 11. Bigger dots indicate that a higher local epoch count was assigned for the corresponding clients. It was found that FedDdrl tended to set a lower epoch count (smaller dots) for clients with higher probing latency. This method could reduce the total training latency since clients with limited computing power did not have to participate in long training epochs. On the other hand, clients with lower probing latency tended to have a higher epoch count. This strategy can fully utilize the computing resource of clients with stronger computing power since they can continue training while waiting for other clients to finish. However, this was not always the case, as shown in Figure 13. On certain occasions, FedDdrl set a high epoch count for clients with long probing latency if the data in these clients were crucial for FL convergence. In any case, FedDdrl was superior to FedMarl, where the FedMarl could either select or terminate a client without the third option of selecting the clients and dynamically tuning the local epoch.

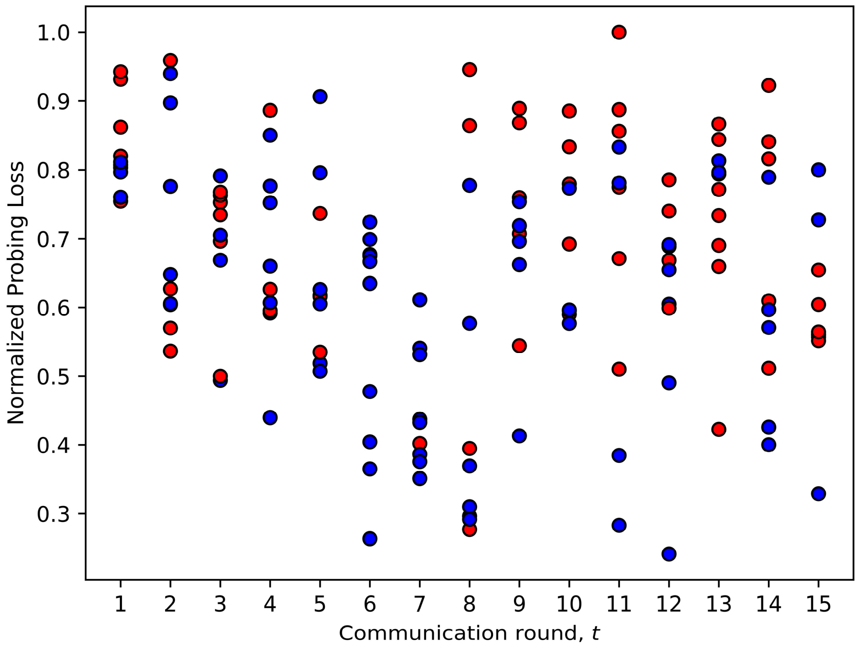

Figure 10. Decisions made by Fiddly based on probing loss.

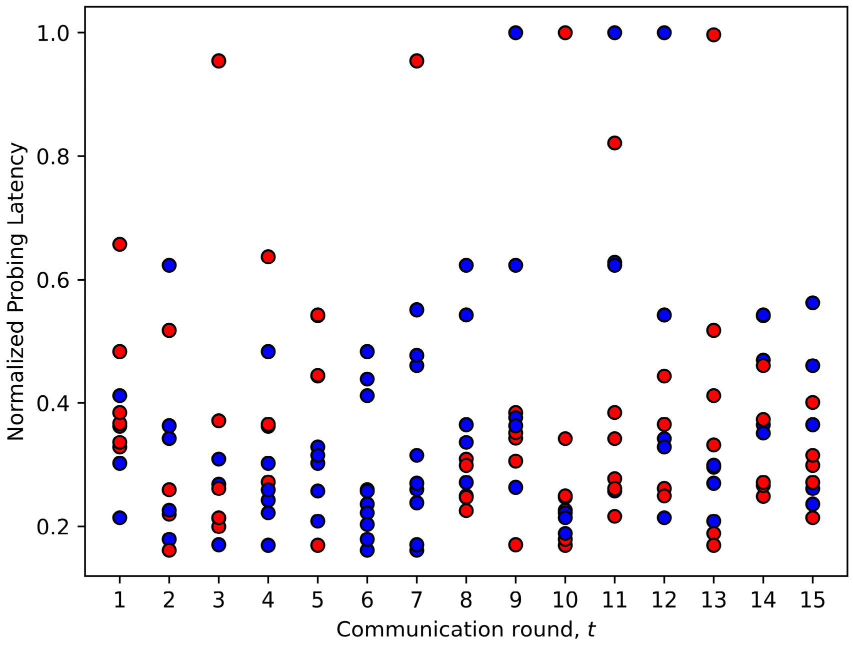

Figure 11. Decisions made by FedDdrl based on probing latency.

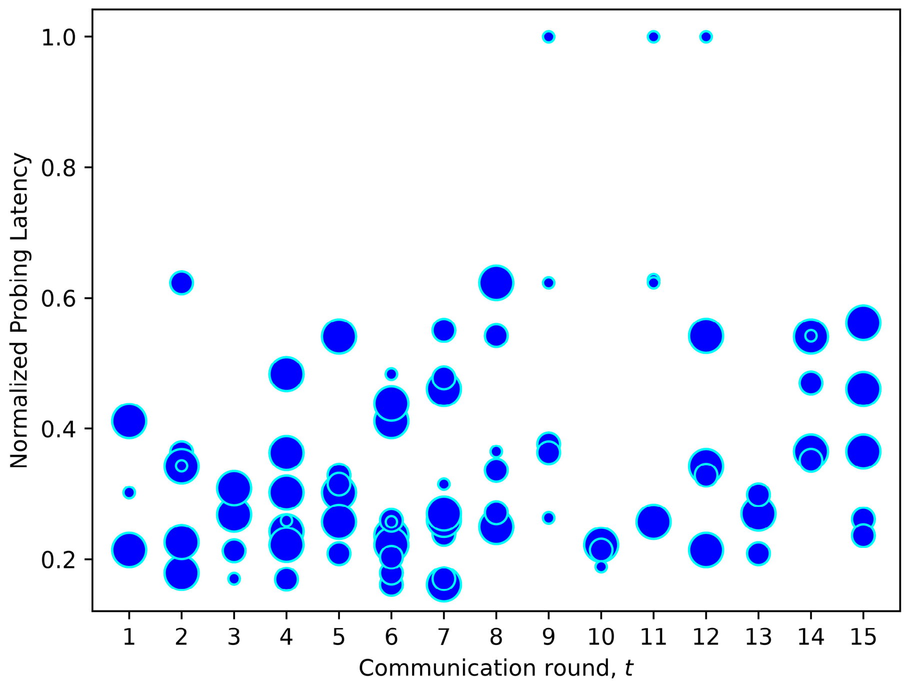

Figure 12. Local epoch count for each selected client in Figure 10. Clients with high probing latency were assigned smaller epoch counts so that the clients could finish local training earlier.

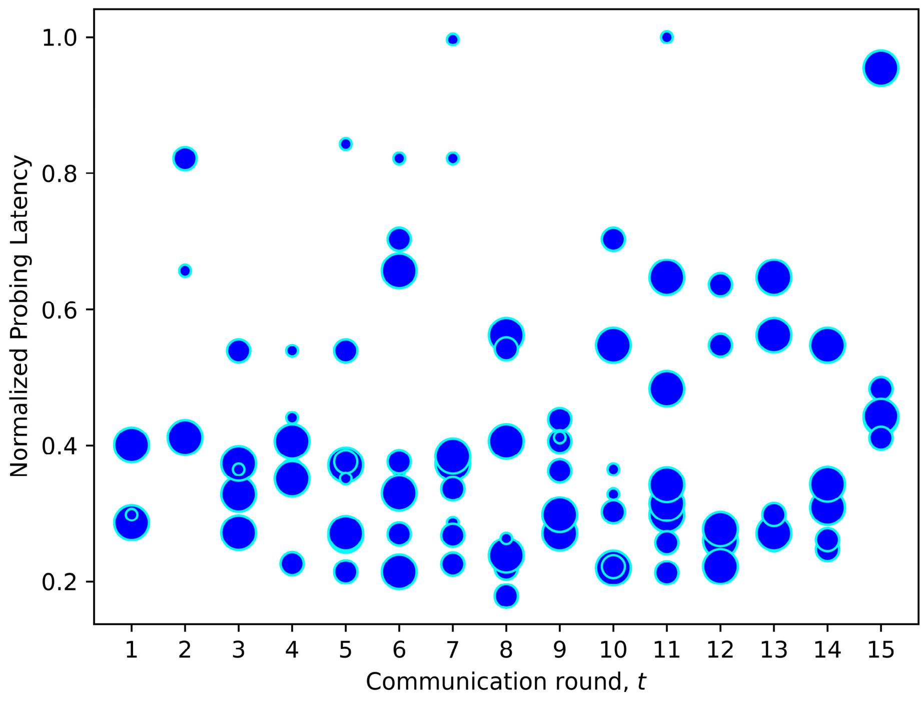

Figure 13. Another example of local epoch adjustment strategy learned by FedDdrl. Selected clients with high probing latency were occasionally assigned large epoch counts to assist the FL convergence.

5.3. How FedDdrl Optimizes the Three Objectives Simultaneously

The objectives of FedDdrl are to (i) maximize the global model's accuracy while minimizing the (ii) FL system's training latency and (iii) communication cost. First, VDN 1 will perform early client termination to terminate clients who are not essential for training. By doing so, we can reduce the total communication cost. Additionally, VDN 1 prefers clients with lower probing latency (which also translates to lower training latency). Thus, VDN 1 plays a huge role in reducing both communication costs and training latency. Second, VDN 2 will dynamically adjust the local epoch count. Clients with limited computing power only have to train with a lesser epoch, so they can finish training earlier. Meanwhile, VDN 2 assigns a higher epoch count to clients with stronger computing power, so they can continue training while waiting for the slower clients. Hence, VDN 2 reduces the training latency and fully utilizes clients with stronger computing power.

Lastly, the global model's accuracy must be retained. When VDN 1 performs client termination, it is essentially reducing C. Intuitively, reducing C seems counter-productive since it reduces the total number of clients participating in training for each communication round t. Fewer clients translate to fewer training data. However, [44] showed that in a non-IID setup, the convergence rate of FedAvg had a weak dependence on C. This makes sense, as some clients may have local datasets with a huge EMD distance from the global distribution. Training the FL model using these clients may hinder the convergence rate. Additionally, [14] showed that using DRL in client selection (or early client termination) can positively affect the convergence rate. This is because selecting useful clients (with useful data) can improve the quality of the overall FL data, which is more crucial than increasing the quantity of data.

In short, FedDdrl can reduce communication cost and training latency without sacrificing model accuracy via early client termination due to the weak correlation between convergence rate and C.

5.4. Computational Complexity Analysis

FedDdrl is composed of a finite number of MLPs. In MLP, let L, n_0 and n_i denote the layer numbers, the size of the input layer (which corresponds to the client state's size) and the number of neurons in i-th layer, respectively. During training mode, the computational complexity for an MLP to update its weight in each step can be expressed as $O(N_b(n_0 n_1 + \sum_{i=1}^{L-1} n_i n_{i+1}))$ [45]. In total, it takes $N_{ep} \times T$ steps for the FedDdrl algorithm to finish training. Hence, the total training computational complexity of FedDdrl is $O(N_{ep} T N_b(n_0 n_1 + \sum_{i=1}^{L-1} n_i n_{i+1}))$. The high computation complexity of the MLP can be performed offline using a powerful device (i.e., the FL server). In the online deployment mode, the computational complexity in each step is dramatically reduced to $O(n_0 n_1 + \sum_{i=1}^{L-1} n_i n_{i+1})$. This is done by cutting off the training procedure, which requires feedforward and backpropagation of N_b data points. Thus, the computational complexity is retained at a favorable level.

5.5. Why Balanced-MixUp Helps in Federated Learning

Without loss of generality, we explored how balanced-MixUp mitigates weight divergence in FL assuming the amount of training data for each class is uniform in the global population. Under this setting, we can express the global distribution $p(y = i)$ for all labels $i = \{1, 2, 3, \ldots, n_c\}$ as shown in Equation (21):

$$p(y = i) = \frac{1}{n_c} \tag{21}$$

In this study, a fraction σ of the local training dataset is sampled from one random label, while the remaining $1 - \sigma$ fraction is sampled uniformly from the remaining labels. Following this assumption, let $i = 1$ be the majority class in each client and $i = \{2, 3, \ldots, n_c\}$ be the minority classes (whichever i can be the majority class since the ordering does not affect the approximation of client distribution). Without balanced-MixUp, we can express the client dataset distribution $p^k(y = i)$ as Equation (22):

$$p^k(y = i) = \begin{cases} \sigma, & i = 1 \\ \frac{1-\sigma}{n_c - 1}, & i = \{2, 3, \ldots, n_c\} \end{cases} \tag{22}$$

Based on Equations (21) and (22), the EMD between local and global distribution for FedAvg without balanced-MixUp, denoted as EMD$_{ori}$, can be written as Equation (23):

$$\begin{aligned} &\text{EMD}_{ori} \\ &= \textstyle\sum_{i=1}^{n_c} \parallel p^k(y=i) - p(y=i) \parallel \\ &=\parallel \sigma - \tfrac{1}{n_c} \parallel + (n_c - 1) \parallel \tfrac{1-\sigma}{n_c-1} - \tfrac{1}{n_c} \parallel \end{aligned} \tag{23}$$

On the other hand, the client dataset distribution with balanced-MixUp $p^k_{MixUp}(y=i)$ can be expressed as shown in Equation (24):

$$p^k_{MixUp}(y=i) = \begin{cases} \mathbb{E}(\lambda), & i = 1 \\ \frac{1-\mathbb{E}(\lambda)}{n_c-1}, & i = \{2,\, 3,\, \ldots,\, n_c\,\} \end{cases} \tag{24}$$

where $\mathbb{E}(\lambda)$ is the expected value of $\lambda \sim$ Beta(α, β). $\mathbb{E}(\lambda)$ can be written as Equation (25):

$$\mathbb{E}(\lambda) = \frac{\alpha}{\alpha + \beta} \tag{25}$$

Based on Equations (21) and (24), the EMD between FedAvg with balanced-MixUp denoted as EMD$_{\text{MixUp}}$ can be written as Equation (26):

$$\begin{aligned} &\text{EMD}_{\text{MixUp}} \\ &=\parallel \textstyle\sum_{i=1}^{n_c} p^k_{MixUp}(y=i) - p(y=i) \parallel \\ &=\parallel \mathbb{E}(\lambda) - \tfrac{1}{n_c} \parallel + (n_c - 1) \parallel \tfrac{1-\mathbb{E}(\lambda)}{n_c-1} - \tfrac{1}{n_c} \parallel \end{aligned} \tag{26}$$

Take our experiments using CIFAR-10 as example, where $\sigma = 0.8$, $n_c = 10$, $\lambda \sim$ Beta(0.4, 0.4) and $\mathbb{E}(\lambda) = 0.5$. Based on Equations (23) and (26), the EMD$_{ori}$ and EMD$_{\text{MixUp}}$ can be computed as 0.777 and 0.444, respectively. This shows that balanced-MixUp could significantly reduce the weight divergence caused by the EMD between p^k and p.

The above proposition is aligned with our experiments. Table 5 shows the performance of FedAvg with and without balanced-MixUp in different C. Balanced-MixUp provided a drastic accuracy boost to FedAvg in all datasets. The accuracy improvement was as high as 17.0% for the CrisisIBD dataset when $C = 10$. This shows that balanced-MixUp can effectively mitigate the weight divergence caused by the EMD, especially when a low number of clients participate in a communication round t. Note that for the MNIST dataset ($C = 10$), the accuracy of FedAvg with balanced-MixUp was slightly poorer than its counterpart, lagging behind by merely 1.4%. This is reasonable, as MNIST is considered a simple task [32] in which FedAvg could perform similarly in certain non-IID settings. Another notable observation is that FedAvg with balanced-MixUp significantly outperformed its counterpart in both the challenging CIFAR-10 and CrisisIBD datasets. The observation is consistent in both $C = 5$ and $C = 10$ experiments. This is encouraging because it proves that balanced-MixUp is useful in mitigating non-IID label shifts, especially for algorithms like FedMarl and FedDdrl which do not have a fixed value of C.

Table 5. Performance of FedAvg with and without Balanced-MixUp.

Method		MNIST (K=100)	CIFAR-10 (K=100)	CrisisIBD (K=98)
$C = 5$	FedAvg	78.9% ± 9.3%	62.7% ± 2.9%	42.0% ± 3.4%
	FedAvg with Balanced-MixUp	88.1% ± 3.6%	69.4% ± 2.4%	52.9% ± 4.2%
$C = 10$	FedAvg	94.6% ± 2.1%	72.8% ± 3.9%	43.2% ± 5.5%
	FedAvg with Balanced-MixUp	93.2% ± 2.0%	76.5% ± 1.7%	60.2% ± 1.5%

6. Conclusions

In this paper, we proposed a DDRL-based FL framework (FedDdrl) for adaptive early client termination and local epoch adjustment. FedDdrl can terminate clients with high probing latency to reduce total training latency and communication costs, and it can automatically adjust the local epoch to fully utilize clients' computing resources. We also showed that balanced-MixUp is a useful augmentation technique to mitigate the impact of weight divergence arising from non-IID label shifts in FL. The simulation results on MNIST, CIFAR-10 and CrisisIBD confirmed that FedDdrl outperformed the comparison schemes in terms of the model's accuracy, training latency and communication costs of FL under extreme non-IID settings. As a future work, we would explore the performance of FedDdrl on other types of non-IID settings, such as feature distribution skew and quantity skew.

Author Contributions: Conceptualization, Y.J.W. and M.-L.T.; methodology, Y.J.W.; software, Y.J.W.; validation, Y.J.W., M.-L.T., B.-H.K. and Y.O.; formal analysis, Y.J.W. and M.-L.T.; resources, M.-L.T.; writing—original draft preparation, Y.J.W.; writing—review and editing, M.-L.T., B.-H.K. and Y.O.; visualization, Y.J.W. and M.-L.T.; supervision, M.-L.T. and B.-H.K.; project administration, M.-L.T.; funding acquisition, M.-L.T. All authors have read and agreed to the published version of the manuscript.

Funding: National Institute of Information and Communications Technology (NICT): ICT Virtual Organization of ASEAN Institutes and NICT (ASEAN IVO).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Acknowledgments: The ASEAN IVO (http://www.nict.go.jp/en/asean_ivo/index.html) (accessed on 22 December 2022) project, Context-Aware Disaster Mitigation using Mobile Edge Computing and Wireless Mesh Network, was involved in the production of the contents of this work and financially supported by NICT (http://www.nict.go.jp/en/ index.html) (accessed on 22 December 2022). Also, we gratefully appreciate the anonymous reviewers' valuable reviews and comments.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Al-Maslamani, N.; Abdallah, M.; Ciftler, B.S. Secure Federated Learning for IoT Using DRL-Based Trust Mechanism. In Proceedings of the 2022 International Wireless Communications and Mobile Computing, IWCMC 2022, Dubrovnik, Croatia, 30 May–3 June 2022; pp. 1101–1106. [CrossRef]
2. Reinsel, D.; Gantz, J.; Rydning, J. The Digitization of the World from Edge to Core. *Fram. Int. Data Corp.* **2018**, *16*, 16–44.
3. Sheller, M.J.; Edwards, B.; Reina, G.A.; Martin, J.; Pati, S.; Kotrotsou, A.; Milchenko, M.; Xu, W.; Marcus, D.; Colen, R.R.; et al. Federated Learning in Medicine: Facilitating Multi-Institutional Collaborations without Sharing Patient Data. *Sci. Rep.* **2020**, *10*, 12598. [CrossRef] [PubMed]
4. McMahan, B.H.; Moore, E.; Ramage, D.; Hampson, S.; Agüera y Arcas, B. Communication-Efficient Learning of Deep Networks from Decentralized Data. In Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, AISTATS 2017, Fort Lauderdale, FL, USA, 20–22 April 2017. [CrossRef]
5. Hard, A.; Rao, K.; Mathews, R.; Ramaswamy, S.; Beaufays, F.; Augenstein, S.; Eichner, H.; Kiddon, C.; Ramage, D. Federated Learning for Mobile Keyboard Prediction. *arXiv* **2018**, arXiv:1811.03604. [CrossRef]
6. Ahmed, L.; Ahmad, K.; Said, N.; Qolomany, B.; Qadir, J.; Al-Fuqaha, A. Active Learning Based Federated Learning for Waste and Natural Disaster Image Classification. *IEEE Access* **2020**, *8*, 208518–208531. [CrossRef]
7. Wong, Y.J.; Tham, M.-L.; Kwan, B.-H.; Gnanamuthu, E.M.A.; Owada, Y. An Optimized Multi-Task Learning Model for Disaster Classification and Victim Detection in Federated Learning Environments. *IEEE Access* **2022**, *10*, 115930–115944. [CrossRef]
8. Reina, G.A.; Gruzdev, A.; Foley, P.; Perepelkina, O.; Sharma, M.; Davidyuk, I.; Trushkin, I.; Radionov, M.; Mokrov, A.; Agapov, D.; et al. OpenFL: An Open-Source Framework for Federated Learning. *arXiv* **2021**, arXiv:2105.06413. [CrossRef]
9. Chen, X.; Li, Z.; Ni, W.; Wang, X.; Zhang, S.; Xu, S.; Pei, Q. Two-Phase Deep Reinforcement Learning of Dynamic Resource Allocation and Client Selection for Hierarchical Federated Learning. In Proceedings of the 2022 IEEE/CIC International Conference on Communications in China, ICCC 2022, Foshan, China, 11–13 August 2022; pp. 518–523. [CrossRef]
10. Yang, W.; Xiang, W.; Yang, Y.; Cheng, P. Optimizing Federated Learning with Deep Reinforcement Learning for Digital Twin Empowered Industrial IoT. *IEEE Trans. Industr. Inform.* **2022**, *19*, 1884–1893. [CrossRef]

11. Zhang, W.; Yang, D.; Wu, W.; Peng, H.; Zhang, N.; Zhang, H.; Shen, X. Optimizing Federated Learning in Distributed Industrial IoT: A Multi-Agent Approach. *IEEE J. Sel. Areas Commun.* **2021**, *39*, 3688–3703. [CrossRef]
12. Liu, L.; Zhang, J.; Song, S.H.; Letaief, K.B. Client-Edge-Cloud Hierarchical Federated Learning. In Proceedings of the IEEE International Conference on Communications 2020, Dublin, Ireland, 7–11 June 2020. [CrossRef]
13. Song, Q.; Lei, S.; Sun, W.; Zhang, Y. Adaptive Federated Learning for Digital Twin Driven Industrial Internet of Things; Adaptive Federated Learning for Digital Twin Driven Industrial Internet of Things. In Proceedings of the 2021 IEEE Wireless Communications and Networking Conference (WCNC), Nanjing, China, 29 March–1 April 2021. [CrossRef]
14. Zhang, S.Q.; Lin, J.; Zhang, Q. A Multi-Agent Reinforcement Learning Approach for Efficient Client Selection in Federated Learning. *Proc. AAAI Conf. Artif. Intell.* **2022**, *36*, 9091–9099. [CrossRef]
15. Abdulrahman, S.; Tout, H.; Ould-Slimane, H.; Mourad, A.; Talhi, C.; Guizani, M. A Survey on Federated Learning: The Journey from Centralized to Distributed on-Site Learning and Beyond. *IEEE Internet Things J.* **2021**, *8*, 5476–5497. [CrossRef]
16. Zhao, Y.; Li, M.; Lai, L.; Suda, N.; Civin, D.; Chandra, V. Federated Learning with Non-IID Data. *arXiv* **2018**, arXiv:1806.00582. [CrossRef]
17. Wang, J.; Liu, Q.; Liang, H.; Joshi, G.; Vincent Poor, H. Tackling the Objective Inconsistency Problem in Heterogeneous Federated Optimization. In Proceedings of the 34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, BC, Canada, 6–12 December 2020. [CrossRef]
18. Park, J.; Yoon, D.; Yeo, S.; Oh, S. AMBLE: Adjusting Mini-Batch and Local Epoch for Federated Learning with Heterogeneous Devices. *J. Parallel. Distrib. Comput.* **2022**, *170*, 13–23. [CrossRef]
19. Zhang, H.; Xie, Z.; Zarei, R.; Wu, T.; Chen, K. Adaptive Client Selection in Resource Constrained Federated Learning Systems: A Deep Reinforcement Learning Approach. *IEEE Access* **2021**, *9*, 98423–98432. [CrossRef]
20. Zhang, P.; Wang, C.; Jiang, C.; Han, Z. Deep Reinforcement Learning Assisted Federated Learning Algorithm for Data Management of IIoT. *IEEE Trans. Industr. Inform.* **2021**, *17*, 8475–8484. [CrossRef]
21. Wang, H.; Kaplan, Z.; Niu, D.; Li, B. Optimizing Federated Learning on Non-IID Data with Reinforcement Learning. In Proceedings of the IEEE INFOCOM 2020-IEEE Conference on Computer Communications, Toronto, ON, Canada, 6–9 July 2020; pp. 1698–1707. [CrossRef]
22. Galdran, A.; Carneiro, G.; González Ballester, M.A. Balanced-MixUp for Highly Imbalanced Medical Image Classification. In *Medical Image Computing and Computer Assisted Intervention–MICCAI 2021, Proceedings of the 24th International Conference, Strasbourg, France, 27 September–1 October 2021*; Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics); Springer: Cham, Switzerland, 2021; Volume 12905, pp. 323–333. [CrossRef]
23. Li, T.; Sahu, A.K.; Zaheer, M.; Sanjabi, M.; Talwalkar, A.; Smith, V. Federated Optimization in Heterogeneous Networks. *arXiv* **2018**, arXiv:1812.06127. [CrossRef]
24. Nishio, T.; Yonetani, R. Client Selection for Federated Learning with Heterogeneous Resources in Mobile Edge. In Proceedings of the ICC 2019—2019 IEEE International Conference on Communications (ICC), Shanghai, China, 20–24 May 2019. [CrossRef]
25. Zheng, J.; Li, K.; Tovar, E.; Guizani, M. Federated Learning for Energy-Balanced Client Selection in Mobile Edge Computing. In Proceedings of the 2021 International Wireless Communications and Mobile Computing, IWCMC 2021, Harbin, China, 28 June–2 July 2021; pp. 1942–1947. [CrossRef]
26. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; Riedmiller, M. Playing Atari with Deep Reinforcement Learning. *arXiv* **2013**, arXiv:1312.5602. [CrossRef]
27. Vinyals, O.; Babuschkin, I.; Czarnecki, W.M.; Mathieu, M.; Dudzik, A.; Chung, J.; Choi, D.H.; Powell, R.; Ewalds, T.; Georgiev, P.; et al. Grandmaster Level in StarCraft II Using Multi-Agent Reinforcement Learning. *Nature* **2019**, *575*, 350–354. [CrossRef]
28. Silver, D.; Hubert, T.; Schrittwieser, J.; Antonoglou, I.; Lai, M.; Guez, A.; Lanctot, M.; Sifre, L.; Kumaran, D.; Graepel, T.; et al. Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm. *arXiv* **2017**, arXiv:1712.01815. [CrossRef]
29. Han, M.; Sun, X.; Zheng, S.; Wang, X.; Tan, H. Resource Rationing for Federated Learning with Reinforcement Learning. In Proceedings of the 2021 Computing, Communications and IoT Applications (ComComAp), Shenzhen, China, 26–28 November 2021; pp. 150–155. [CrossRef]
30. Xiong, Z.; Cheng, Z.; Xu, C.; Lin, X.; Liu, X.; Wang, D.; Luo, X.; Zhang, Y.; Qiao, N.; Zheng, M.; et al. Facing Small and Biased Data Dilemma in Drug Discovery with Federated Learning. *bioRxiv* **2020**. [CrossRef]
31. Jallepalli, D.; Ravikumar, N.C.; Badarinath, P.V.; Uchil, S.; Suresh, M.A. Federated Learning for Object Detection in Autonomous Vehicles. In Proceedings of the IEEE 7th International Conference on Big Data Computing Service and Applications, BigDataService, Oxford, UK, 23–26 August 2021; pp. 107–114. [CrossRef]
32. Li, Q.; Diao, Y.; Chen, Q.; He, B. Federated Learning on Non-IID Data Silos: An Experimental Study. In Proceedings of the 2022 IEEE 38th International Conference on Data Engineering (ICDE), Virtual, 9–12 May 2021; pp. 965–978. [CrossRef]
33. Sunehag, P.; Lever, G.; Gruslys, A.; Marian Czarnecki, W.; Zambaldi, V.; Jaderberg, M.; Lanctot, M.; Sonnerat, N.; Leibo, J.Z.; Tuyls, K.; et al. Value-Decomposition Networks for Cooperative Multi-Agent Learning. *arXiv* **2017**, arXiv:1706.05296. [CrossRef]
34. Zhang, H.; Cisse, M.; Dauphin, Y.N.; Lopez-Paz, D. Mixup: Beyond Empirical Risk Minimization. In Proceedings of the 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, 30 April–3 May 2018. Conference Track Proceedings 2017. [CrossRef]

35. Thulasidasan, S.; Chennupati, G.; Bilmes, J.A.; Bhattacharya, T.; Michalak, S. On Mixup Training: Improved Calibration and Predictive Uncertainty for Deep Neural Networks. *Adv. Neural. Inf. Process. Syst.* **2019**, *32*.
36. Zhou, Z.; Qi, L.; Shi, Y. Generalizable Medical Image Segmentation via Random Amplitude Mixup and Domain-Specific Image Restoration. In Proceedings of the 17th European Conference, Computer Vision–ECCV 2022, Tel Aviv, Israel, 23–27 October 2022; Springer: Cham, Switzerland, 2020; pp. 420–436. [CrossRef]
37. Sun, L.; Xia, C.; Yin, W.; Liang, T.; Yu, P.S.; He, L. Mixup-Transformer: Dynamic Data Augmentation for NLP Tasks. In Proceedings of the 28th International Conference on Computational Linguistics, Barcelona, Spain, 8–13 December 2020; pp. 3436–3440. [CrossRef]
38. Guo, H.; Mao, Y.; Zhang, R. Augmenting Data with Mixup for Sentence Classification: An Empirical Study. *arXiv* **2019**, arXiv:1905.08941. [CrossRef]
39. Chou, H.P.; Chang, S.C.; Pan, J.Y.; Wei, W.; Juan, D.C. Remix: Rebalanced Mixup. In Proceedings of the Computer Vision–ECCV 2020 Workshops, Glasgow, UK, 23–28 August 2020; Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). Volume 12540, pp. 95–110. [CrossRef]
40. Alam, F.; Alam, T.; Ofli, F.; Imran, M. Social Media Images Classification Models for Real-Time Disaster Response. *arXiv* **2021**, arXiv:2104.04184v1. [CrossRef]
41. Sandler, M.; Howard, A.; Zhu, M.; Zhmoginov, A.; Chen, L.-C. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In Proceedings of the 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–22 June 2018; pp. 4510–4520.
42. Tensorflow Federated Using TFF for Federated Learning Research | TensorFlow Federated. Available online: https://www.tensorflow.org/federated/tff_for_research (accessed on 25 December 2022).
43. Xu, Z.-Q.J.; Zhang, Y.; Luo, T.; Xiao, Y.; Ma, Z. Frequency Principle: Fourier Analysis Sheds Light on Deep Neural Networks. *Commun. Comput. Phys.* **2019**, *28*, 1746–1767. [CrossRef]
44. Li, X.; Huang, K.; Yang, W.; Wang, S.; Zhang, Z. On the Convergence of FedAvg on Non-IID Data. *arXiv* **2019**, arXiv:1907.02189. [CrossRef]
45. Yang, H.; Xiong, Z.; Zhao, J.; Niyato, D.; Xiao, L.; Wu, Q. Deep Reinforcement Learning Based Intelligent Reflecting Surface for Secure Wireless Communications. *IEEE Trans. Wirel. Commun.* **2020**, *20*, 375–388. [CrossRef]