Computing and Informatics, Vol. 43, 2024, 1–37, doi: 10.31577/cai_2024_1_1

FEDDRL: TRUSTWORTHY FEDERATED LEARNING MODEL FUSION METHOD BASED ON STAGED REINFORCEMENT LEARNING

Leiming Chen, Weishan Zhang, Cihao Dong,
Ziling Huang, Yuming Nie

School of Computer Science and Technology
China University of Petroleum (East China)
Qingdao 266580, China
e-mail: chenleiming2020@163.com, zhangws@upc.edu.cn

Zhaoxiang Hou

Digital Research Institute
ENN Group
Langfang 065001, China
e-mail: houzhaoxiang@enn.cn

Sibo Qiao*

School of Software
Tiangong University
Tianjin 300387, China
e-mail: siboqiao@126.com

Chee Wei Tan

School of Computer Science and Engineering
Nanyang Technological University
Singapore 639798, Singapore
e-mail: cheewei.tan@ntu.edu.sg

* Corresponding author

Abstract. Federated learning facilitates collaborative data analysis among multiple participants while preserving user privacy. However, conventional federated learning approaches, typically employing weighted average techniques for model fusion, confront two significant challenges: 1. The inclusion of malicious models in the fusion process can drastically undermine the accuracy of the aggregated global model. 2. Due to the heterogeneity problem of devices and data, the number of client samples does not determine the weight value of the model. To solve those challenges, we propose a trustworthy model fusion method based on reinforcement learning (FedDRL), which includes two stages. In the first stage, we propose a reliable client selection mechanism to exclude malicious models from the fusion process. In the second stage, we propose an adaptive model fusion method that dynamically assigns weights based on model quality to aggregate the best global models. Finally, we validate our approach against five distinct model fusion scenarios, demonstrating that our algorithm significantly enhanced reliability without compromising accuracy.

Keywords: Federated learning, model fusion, model attack, reinforcement learning

1 INTRODUCTION

With the advent of deep learning technologies, various industries have been integrating these technologies into their sectors, promoting the development of intelligent transportation, smart logistics, and healthcare systems. These technologies are crucial in reducing production and management costs, enhancing operational efficiency, and accelerating industry digitization. However, supervised learning remains the primary method for training deep learning models, where the volume and diversity of samples are essential for creating high-quality models. Consequently, acquiring extensive and varied data samples has emerged as the initial step in training deep learning models. This approach has led to sample sources expanding from single industries to collaborations across multiple sectors to develop large-scale datasets. To achieve multi-party joint data analysis under the condition of protecting data security and privacy, Google has proposed federated learning technology for the first time. Although federated learning solves the problem of user privacy protection, the traditional federated learning algorithm assumes that all participants are trustworthy. On the contrary, in the actual scenario, if participants exhibit malicious behavior and intentionally contribute harmful models to the fusion process, it can significantly disrupt the global model's convergence. Thus, creating adaptive defenses for federated learning systems becomes increasingly crucial [1]. Identifying methods to remove malicious models in federated learning model fusion has become a critical issue. Simultaneously, when a client submits low-quality models for fusion, determining how to adaptively adjust each model's fusion weights based on their quality is also an urgent problem needing resolution. Some studies have applied reinforcement learning tech-

niques to address these weighting issues. For instance, the Favor [2] method uses the DDPG [3] to assign weights to participant models. Additional research has applied reinforcement learning to address device selection [4, 5], resource optimization [6, 7], and communication optimization in IoT federated learning contexts.

Reinforcement learning (RL) employs a trial-and-error strategy. The essence of this approach is training an intelligent agent that interacts with the external environment through varied actions. The environment then provides feedback in the form of rewards and penalties based on the agent's actions, guiding the agent toward optimal action selection by maximizing reward value. However, employing reinforcement learning presents certain challenges. Firstly, continuous training is required for sample collection through environmental interaction. When the cost of such interactions is prohibitive or unacceptable (for example, in our scenario, where the server must frequently calculate the global model's parameters), the efficiency of sample collection significantly impacts the reinforcement learning training duration. Secondly, when the agent's action space is vast and continuous, it leads to prolonged sampling periods. These issues mean traditional single-agent reinforcement learning training approaches can be exceedingly time-consuming. Applying reinforcement learning in federated learning requires addressing these problems, as increasing participant numbers escalates the agent training time. Therefore, optimizing the action space for reinforcement learning to expedite the agent training process is an essential challenge to address.

Why opt for phased reinforcement learning? We take an example to explain this problem. Consider a robot learning to cook through reinforcement learning with the process divided into washing, chopping, and cooking stages. The robot must master each stage to prepare a successful dish. Traditional reinforcement learning aims to identify the optimal action across all stages simultaneously; however, mastering the initial stage is essential before progressing. By adopting a phased learning approach, the robot sequentially masters each stage, streamlining the learning process and leading to more effective outcomes. Similarly, if malicious models are not initially filtered out, the agent's trial-and-error costs in weight assignment for these models will increase. To resolve these issues, we propose a staged reinforcement learning algorithm (FedDRL).

The contributions of the paper are as follows.

- We design a federated learning framework that employs reinforcement learning for model fusion, designed to select trustworthy clients and optimally assign model weights.
- We propose an adaptive client selection strategy based on the A2C algorithm, dynamically identifying and selecting trustworthy clients while excluding malicious ones from the model fusion process based on situational analysis.
- We propose an adaptive weight assignment method that adaptively adjusts the weights according to the quality of their uploaded models.

- We present five types of model fusion scenarios to validate the performance of each algorithm. We also compare the performance of our algorithm with the baseline algorithm on three public datasets.

2 RELATED WORK

2.1 Federated Learning

Research in federated learning primarily aims to address two challenges: enhancing the generalization of the global model on the server side and personalizing the model on the client side. Consequently, federated learning algorithms are bifurcated into server-side and client-side optimization strategies. Google initially introduced the FedAvg algorithm [8] to address the problem of server-side global model fusion. To improve global model convergence, Karimireddy et al. developed the Scaffold method [9], which mitigates client-side drift by integrating a control variable. Similarly, Li et al. introduced FedProx [10], applying a regularization function to client models to correct deviations. Additionally, Wang et al. unveiled FedNova [11], addressing global model convergence issues by normalizing parameters on both client and server ends. Furthermore, Li et al. have introduced the MOON [12] technique, leveraging model comparison learning to enhance global model convergence. Chen et al. [13] also proposed a client identification method based on model parameter features to achieve trustworthy federated learning.

While those approaches enhance the global model's convergence speed, practical federated learning situations reveal variances in the quality of models trained by individual participants. These discrepancies stem from the diversity in computational resources and the calibre of data samples available to each participant. Additionally, variations arise due to the quantity and type of samples possessed by each participant, a phenomenon known as Non-IID (Non-Independent and Identically Distributed). Consequently, these factors complicate the attainment of optimal global model aggregation in the Non-IID environments.

2.2 Challenges of Non-IID Data Distribution

The Non-IID data issue significantly impacts federated learning models' convergence. Zhao et al. explored various federated learning methods' performance on non-IID datasets, demonstrating significant accuracy challenges [14]. Accordingly, several studies have addressed the non-IID dilemma in federated learning. For instance, Zhang et al. proposed the FedPD approach [15], optimizing models and communication for non-convex objective functions. Moreover, Gong et al. introduced AutoCFL [16], utilizing a weighted voting client clustering strategy to mitigate non-IID and imbalanced data effects. Huang et al. developed FedAMP [17], which addresses Non-IID data-induced client-side model personalization issues through personalized

model updates. Li et al. devised FedBN [18], incorporating a batch normalization layer into local models to address feature shift challenges due to data heterogeneity. Briggs et al. suggested a hierarchical clustering method (FL+HC) [19], improving Non-IID dataset model performance by grouping clients for independent model training. Additionally, Gao et al. offered the FedDC approach [20], bridging client and global model parameter disparities through a control variable. Lastly, Mu et al. introduced FedProc [21], directing client model training by integrating a comparative loss between client and global models. Chen et al. [22] proposed a federated learning method based on adaptive knowledge distillation to improve the accuracy of heterogeneous model scenarios.

Although these methodologies advance Non-IID issue mitigation in federated learning, they typically assign uniform fusion weights to all clients, failing to exclude malicious or low-quality model contributions. Consequently, dynamically selecting clients for fusion and adaptively calculating each model's weight remains critical for successful global model integration.

2.3 Federated Reinforcement Learning

Given the adaptive learning potential of reinforcement learning, its application within federated learning contexts has garnered interest. Some research has concentrated on leveraging reinforcement learning to boost global model performance. For instance, Wang et al. introduced the Favor method [2], which adaptively selects clients for model fusion. Sun et al. developed the PG-FFL framework [23], addressing the challenge of client weight computation during model fusion. Additional studies have applied reinforcement learning for device optimization within federated IoT frameworks. For example, Zhang et al. utilized the DDPG algorithm [4] for optimal device selection. Zhang et al. also formulated the FedMarl strategy [24], employing multi-agent reinforcement learning for node selection. Similarly, Yang et al. proposed a digital twin architecture (DTEI) [5], applying reinforcement learning for device selection issues. Other investigations have addressed resource optimization and scheduling challenges within IoT contexts, such as Zhang et al.'s RoF methodology [6], which leverages multi-intelligent reinforcement learning for optimal resource scheduling. Additionally, Rjoub et al. have developed trusted device selection techniques [25] and the DDQN-Trust method [7], utilizing Q-learning to assess devices' credit scores for optimal scheduling. To ameliorate federated learning communication issues, Yang et al. introduced a reinforcement learning-based model evaluation method [26], selecting optimal devices for training and fusion. Nevertheless, while these efforts predominantly focus on IoT environment applications – such as device selection, resource optimization, and communication enhancement – they seldom address federated learning's model weight calculation challenges. Therefore, Zhang et al. proposed the R^2Fed framework [27], employing the DDPG reinforcement learning method for adaptive client weight calculation. Chen et al. [28] also constructed a trustworthy federated learning platform based on reinforcement learning methods.

Although current research addresses the issue of weight allocation in federated learning, it often neglects the training efficiency of the agents. Therefore, optimizing the training efficiency of agents is a significant challenge that needs attention.

3 METHOD

3.1 Problem Definition

In this section, we scrutinize the prevailing challenges of the current federated learning approach and subsequently propose a solution. In federated learning, the objective is to get the global model by amalgamating local models from all clients through server-side aggregation. We define n clients as involved in model fusion, and the client is denoted as C_i where $C_i \in \{C_1, C_2, C_3, \ldots, C_n\}$. Each client has a network model M_i, where $M_i \in \{M_1, M_2, M_3, \ldots, M_n\}$. Each client has its private data D_i, where $D_i \in \{D_1, D_2, D_3 \ldots D_n\}$. The number of samples in each dataset is S_i, where $S_i \in \{S_1, S_2, S_3 \ldots S_n\}$. The total number of samples is $\sum_{i=1}^{N} S_i$. We define the θ_i as a model parameter of M_i, where $\theta_i \in \{\theta_1, \theta_2, \theta_3, \ldots, \theta_n\}$.

Additionally, the server-side model aggregation process per round is defined as shown in Equation (1):

$$\theta_{\text{global}} = \sum_{i=1}^{N} w_i \theta_i, \quad \text{where } w_i = \frac{S_i}{\sum_{i=1}^{N} S_i}, \quad w_i \geq 0, \sum_{i=1}^{N} w_i = 1. \tag{1}$$

The w_i is the fusion weight of each model parameter.

Traditional Federated Learning typically employs a weighted average approach for computing model fusion weights, with each model's weight determined by its corresponding client's data sample size relative to the total. Thus, clients contributing more data exert a greater influence on the aggregated model. However, this method fails to consider the quality of each client's model and the potential inclusion of malicious models in real-world scenarios. We illustrate the deficiencies of the traditional federated fusion algorithm through two scenarios:

Scenario 1: A client's data represents 20 % of the total, yet its model's accuracy is merely 53 %. Employing the conventional federated fusion algorithm in this case would detrimentally impact the global model's accuracy.

Scenario 2: A client engaged in model fusion launches malicious attacks, intentionally skewing its model's output to reflect a mere 10 % accuracy. If such malicious models are incorporated through the standard fusion process, the accuracy of the global model would be severely compromised.

Addressing these challenges necessitates an adaptive weight calculation strategy capable of nullifying malicious models by assigning them a weight of zero, thus excluding them from the fusion process. Concurrently, this approach should dynamically adjust the weights of each client's model, prioritizing those of higher quality to enhance the global model's overall accuracy.

Adopting a single-agent reinforcement learning strategy to tackle these issues introduces new challenges. As the number of clients increases, so does the agent's action space as well, prolonging the training duration. Additionally, a single-agent framework is limited to interacting with just one environment, further extending the sampling period. We propose a bifurcated solution inspired by hierarchical reinforcement learning to mitigate these concerns, thereby streamlining the lengthy reinforcement learning training process. This solution comprises two primary stages: the selection of trustworthy clients and the assignment of optimal weights.

Stage 1: During this phase, the objective is to identify K trustworthy models from a pool of N for inclusion in the global model fusion. Identifying clients who have uploaded malicious models is challenging. We address this by employing reinforcement learning to dynamically select and autonomously screen client models, as delineated in Equation (2).

$$\{M_a, M_{\mathrm{b}}, \ldots, M_{\mathrm{k}}\} \leftarrow \mathrm{SelectTrustworthyModel}\left(\{M_1, M_2, \ldots, M_{\mathrm{n}}\}\right). \quad (2)$$

Stage 2: Building on the first step, we then allocate optimal weights to the verified models to bolster the global model's accuracy, formalized in Equation (3).

$$\{W_1, W_2, \ldots, W_{\mathrm{n}}\} \leftarrow \mathrm{AdaptCalculateWeight}\left(\{M_a, M_b, \ldots, M_{\mathrm{k}}\}\right). \quad (3)$$

Here, AdaptCalculateWeight(.) signifies a method for adaptive weight computation, and W_i represents the optimal computational weight assigned to each client's output.

3.2 A Trustworthy Federated Learning Approach Based on Staged Reinforcement Learning

To address these challenges, we introduce a trusted federated learning framework anchored in staged reinforcement learning (FedDRL). This framework unfolds across two distinct phases. In the first phase, we propose an adaptive client selection strategy aimed at identifying and selecting trustworthy clients for participation in model fusion. Subsequently, in the second phase, we formulate a model weight assignment algorithm designed to dynamically allocate fusion weight values to models based on the prevailing fusion environment. The process is depicted in Figure 1.

3.2.1 Adaptive Client Selection Method

In federated learning, selecting clients is analogous to navigating a Markov decision process. Suppose there are N client models available for upload to a server, which requires a subset for global model aggregation. The collection of these clients forms a node state. Our goal is to select trustworthy clients through agent training adaptively. Prior to agent training, it is essential to outline the fundamental elements of reinforcement learning as follows:

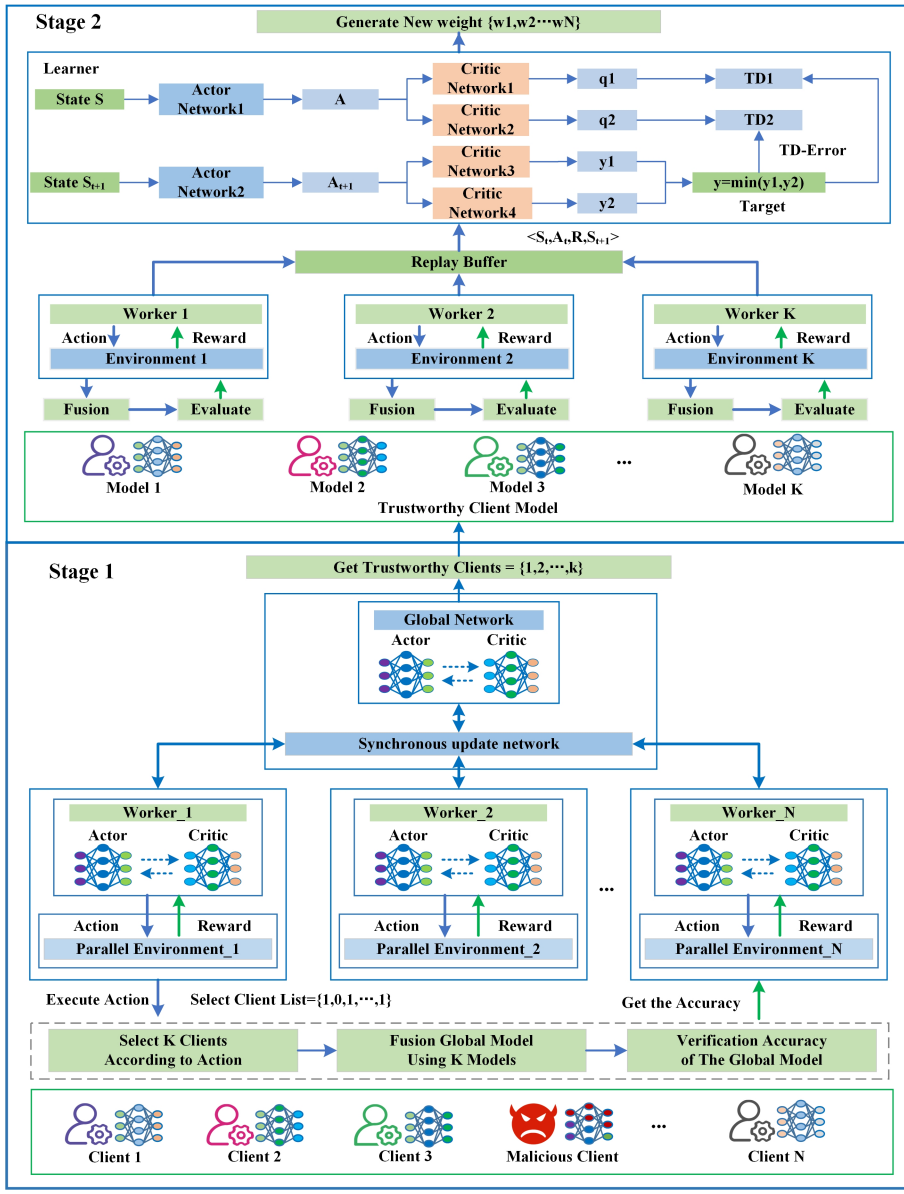


Figure 1. The process of FedDRL framework

State: We define the state at moment t as S^t. To better allow the state to contain key information about the clients, we design the state space to include the accuracy and Loss values of the K client models, the accuracy of the global model after the fusion of the K models, and the number of clients selected. The state space is then used as the state space for the client models, and the state space is used as the state space for the client models. The State space is shown as Equation (4).

$$S^t = \left\{ l_1^t, l_2^t, \ldots, l_k^t, acc_1^t, acc_2^t, acc_3^t \ldots acc_k^t, acc_{global}^t, K \right\}. \tag{4}$$

Action: Our main objective is to eliminate the malicious models and select the trusted models to participate in the fusion. To achieve this purpose, we randomly choose several models to fusion the global model and evaluate the accuracy of global models. If a malicious model is selected to participate in the fusion, the accuracy of the global model will be very low. If only trusted clients are selected to participate in the fusion, the accuracy of the global model will also increase.

We define the action of selecting a model as $a^t \in \{0, 1\}$, where one means that the model is selected and 0 means that the model is not selected. Thus, n models correspond to the action space that can be expressed as Equation (5).

$$A^t = \left\{ a_1^t, a_2^t, a_3^t \ldots a_n^t \right\}, \quad a^t \in \{0, 1\}. \tag{5}$$

Reward: In order to be able to make the agent select as many trusted clients as possible, we define a composite reward function that consists of a global model accuracy improvement reward and a reward for the number of clients selected. We define the accuracy of the global model using all models fused as Acc_{all}, and at the same time, we define the accuracy of the global model obtained from the mth randomly selected client model as Acc_m, and compute the reward value Reward_1 by the difference of $(Acc_m - Acc_{all})$. Meanwhile, in order to allow the agent to select all trusted clients as much as possible, we define the reward corresponding to selecting K number of nodes as Reward_2, and the total reward as the sum of the two parts, as shown in Equation (6).

$$\text{Reward} = \begin{cases} \alpha \cdot (Acc_m - Acc_{all}) + \beta \cdot K, & Acc_m > Acc_{all}, \\ 0, & Acc_m \leq Acc_{all}. \end{cases} \tag{6}$$

The α and β are fixed values that can balance the two rewards.

The goal of the agent is to obtain the maximum long-term reward value based on the discount factor γ, and the process is expressed as Equation (7).

$$R_\gamma = \sum_{t=1}^{T} \gamma_t r_t. \tag{7}$$

Once we have defined the base elements of reinforcement learning, We use a distributed A2C approach to train the agent; A2C is an improved method-based A3C algorithm [29]. Figure 1 shows the A2C architecture, which consists of a central node and K workers. Each worker contains an Actor and a Critic network, where the actor network generates action, and the Critic network evaluates the action and gives the corresponding reward. Meanwhile, each worker independently interacts with the related environment to achieve sampling and training of the Actor and Critic networks. In addition, the Actor and Critic networks of the central node are used to synchronize the network information of each worker and to achieve the fusion and sharing of network parameters of multiple workers.

Therefore, our main objective is to train Actor and Critic networks. We define the Actor network parameters as $\pi(\theta)$ and the Critic network parameters as $V(w)$. The process of the worker and the central node is as follows.

Step 1: Each worker initializes the local network by pulling the global network model parameters from the centre node. Then, each worker trains the Actor and Critic networks by interacting with the environment independently. Finally, the two networks are uploaded to the central node.

Step 2: After the central node collects the network parameters uploaded by all workers, it updates the global model by the weighted averaging method. Then, the server sends the two networks to each worker.

Steps 1 and 2 are repeated according to the total number of times to obtain the final global model.

The training process for the Step 1 neutralization network is as follows: The gradient of the primary communication algorithm of the policy network is calculated as Equation (8).

$$\nabla_\theta J(\theta) = \nabla_\theta \log \pi\left(a_t \mid s_t; \theta\right) A\left(s_t, a_t; \mathrm{w}\right), \tag{8}$$

where $A\left(s_t, a_t; \theta_v\right)$ is the advantage function. The k-step sampling strategy is used in the A2C algorithm to calculate the advantage function, so the definition is expressed as Equation (9).

$$A\left(s_t, a_t; \theta_v\right) = \sum_{i=0}^{k-1} \gamma^i r_{t+i} + \gamma^k V\left(s_{t+k}; \mathrm{w}\right) - V\left(s_t; \mathrm{w}\right). \tag{9}$$

The Loss function of the Actor network is calculated as in Equation (10), and the Critic network is calculated as in Equation (11).

$$\nabla_\theta J(\theta) = \nabla_\theta \log \pi\left(a_t \mid s_t; \theta\right) \left(\sum_{i=0}^{k-1} \gamma^i r_{t+i} + \gamma^k V\left(s_{t+k}; \mathrm{w}\right) - V\left(s_t; \mathrm{w}\right) \right), \tag{10}$$

$$\nabla_w J(w) = \nabla_w \left(\sum_{i=0}^{k-1} \gamma^i r_{t+i} + \gamma^k V\left(s_{t+k}; \mathrm{w}\right) - V\left(s_t; w\right) \right)^2. \tag{11}$$

We update the Actor and Critic network parameters using the derivative formula as Equation (12).

$$w \leftarrow w + \nabla_w J(w), \quad \theta \leftarrow \theta + \nabla_\theta J(\theta). \tag{12}$$

Finally, each worker uploads the Actor and Critic network to the server. Then, the network parameters of the server are calculated using the weighted average method. The process is shown in Equation (13).

$$w_{global} = \frac{1}{n}\sum_{1}^{n} w_i, \quad \theta_{global} = \frac{1}{n}\sum_{1}^{n} \theta_i, \quad i \in [1, n]. \tag{13}$$

When the parameters of the Actor and Critic networks in the central stage are updated, the central node sends down these two networks to all workers, and each worker uses the updated networks to continue interacting with the external environment. The process is repeated for the specified number of rounds until the agent at the central node can obtain a stable reward value.

The process is shown in Algorithm 1.

3.2.2 Adaptive Model Weight Calculation Method

In this phase, our main objective is to achieve the optimal weight assignment for each model. For each communication round, we assume that K trustworthy client models were selected. We need to train the agents in each communication round and use the weight output of the agent to achieve the global model fusion. We first describe the process of global model fusion for agent-based actions. We define θ_i as the i^{th} client model, and the all client models as $\{\theta_1, \theta_2, \ldots, \theta_k\}$. We also define s_i as the number of samples of i^{th} client. The process is as follows:

1. In this step, the agent needs to output the weight values for each model. We define the t^{th} time, the action adopted by the agent as Equation (14).

$$\mathrm{W}^{\mathrm{t}} = \left\{w_1^t, w_2^t, w_3^t, \ldots, w_k^t\right\}. \tag{14}$$

   w_i is the i^{th} weight value output by agent for i^{th} model.

2. We aggregate the global models based on the model weights assigned by the agent, and the process is expressed as Equation (15).

$$\theta_{global}^k = \sum_{t=1}^{T} w_i^t \theta_i. \tag{15}$$

We aim to train the agent so that it can output the optimal fusion weight values based on the quality of each model. To accomplish this goal, we first describe the basic elements of reinforcement learning as follows:

Algorithm 1 The process of trustworthy client selection

Input: Client Models $\{m_1^t, m_2^t, m_3^t, \ldots m_{\mathrm{n}}^t\}$, Round T, Worker Number K, Sampling Step Length S

Output: Chosen Credible Client Model List $\mathrm{M} = \{m_2^t, m_3^t, \ldots, m_{\mathrm{k}}^t\}$

/* Each Worker Training Step */
worker $(\theta, w) \leftarrow$ *GetGlobalParamter* $(\theta_{\mathrm{global}}, w_{\mathrm{global}})$
the Client Upload Current Epoch Model, Turn to State s_0, $t_{\mathrm{start}} = t = 1$
for e from 1 to S **do**
    According to Current State s_0 Randomly Choose Action s_t
    $s_t, a_t, r, s_{t+1} \leftarrow \mathrm{Step}\,(\mathrm{a}_t)$ // Execute Action a_t to Acquire Reward r and Next State s_{t+1}
    $\mathrm{t}_{\mathrm{start}} = \mathrm{t}_{\mathrm{start}} + 1$
    if $\mathrm{s_t}$! =terminal: $\mathrm{R} \leftarrow \mathrm{V}(\mathrm{s_t}; w)$ else: $\mathrm{R} = 0$
for $\mathrm{i} \in \{t-1, \ldots, \mathrm{t_{start}}\}$ **do**
    $\mathrm{R} \leftarrow \mathrm{r_i} + \gamma \mathrm{R}$ // Compute Target TD
    $\nabla_\theta J(\theta) = \nabla_\theta \log \pi_\theta\,(a_t \mid s_t)\,(R - v\,(s_i; w))$ // Compute Strategy Gradient
    $\nabla_w J(w) = \nabla_w\,(R - v\,(s_i; w))^2$ // Compute Critic Network Gradient
    Update Actor Network Parameters: $\theta \leftarrow \theta + \nabla_\theta J(\theta)$
    Update Critic Network Parameters: $w \leftarrow w + \nabla_w J(w)$
/* Center Node Process */
for round from 1 to T **do**
    for worker$_i$ from 1 to K **do**
        Receive Each Worker Parameters(θ, w)
        Global $(\theta_{\mathrm{global}}, w_{\mathrm{global}}) \leftarrow$ **Agg** $(\{(\theta_1, w_1)\,, (\theta_2, w_2)\,, \ldots\})$ // Aggregate Parameters
        worker$_i \leftarrow$ *SendGlobal* $(\theta_{\mathrm{global}}, w_{\mathrm{global}})$ // Send New Parameters to Worker
/* Results Process */
Output Trusted Client Model List $\mathrm{M} = \{m_1^t, m_2^t, \ldots, m_k^t\}$

Environment: The external environment is the server-side global model fusion module, which fuses the global model based on the actions output by the agent and then verifies the accuracy of the global model on the reserved dataset on the server side. Finally, the server side feeds back to the agent the corresponding reward and punishment values based on the accuracy of the global model.

State: We define the agent's state information to include the number of samples corresponding to each client, the accuracy of each client's model, and the accuracy of the global model fused using the weights output by the agent, as shown in Equation (16).

$$\mathrm{S^t} = \left\{s_1^t, s_2^t, s_3^t, \ldots, s_k^t, acc_1^t, acc_2^t, acc_3^t, \ldots, acc_k^t, acc_{global}^t\right\}. \tag{16}$$

The acc^t_{global} is the accuracy of the global model fused using the weights output by the agent.

Action: In each stage, the agent needs to assign each model's weights based on the model's quality. The action space is shown as Equation (17). a^t_i denotes the weight value assigned to the i^{th} client in the state t, while the sum of the corresponding weight values of all clients is 1.

$$\mathrm{A}^{\mathrm{t}} = \left\{a^t_1, a^t_2, \ldots, a^t_k\right\}, \quad \sum_{1}^{k} a^t_i = 1, \quad a^t_i \in (0,1). \tag{17}$$

Reward: We define the model accuracy aggregated using the average method as Acc_{all}, where each model weight is $\frac{1}{N}$. At the m^{th} time, we define the weight set output by the agent as W. Then, we use the weight set to fusion the global model, and we define the accuracy of the global model as Acc_m. We calculate the reward value by subtracting the difference of Acc_m from Acc_{all}. If the calculated result is greater than zero, this indicates that the weights assigned by the agent improve the accuracy of the global model, and we give a positive reward. Conversely, we give a penalty reward. φ, ϕ denotes the reward and penalty factors, respectively. So, the reward is defined as Equation (18).

$$\text{Reward} = \begin{cases} \varphi \cdot (Acc_m - Acc_{all}), & Acc_m > Acc_{all}, \\ \phi \cdot (Acc_m - Acc_{all}), & Acc_m \leq Acc_{all}. \end{cases} \tag{18}$$

When we have finished defining the basic elements, we implement a distributed reinforcement learning approach based on TD3 [30] to train the agent. The training process is shown in Figure 1. This stage includes a central Learner and multiple Worker nodes. Each worker corresponds to a parallel environment. The workflow of each worker is as follows: first, each worker performs global model fusion based on the assigned weights; then verifies the accuracy of the global model by interacting with the parallel environment; and finally receives the reward values from the parallel environment feedback. Finally, each worker stores the corresponding ones in the sampling buffer pool. Multiple workers interact with each environment independently, thus achieving parallel sampling to improve the sampling efficiency. After each worker collects a certain batch of samples, the Learner trains the agent by taking a certain amount of sample data from the experience pool.

The TD3 algorithm consists of six network models, including an Actor network $P(w)$, two Critic networks $Q_1(\theta_1)$, $Q_2(\theta_2)$, and a target Actor-network $P'(w)$, two target Critic networks $Q'_1(\theta_1), Q'_2(\theta_2)$. Each network is shown in Figure 1. The Learner randomly draws N batches of sample data from the buffer pool every certain round to train the model. The training processes are as follows.

1. First, select the action a_{t+1} based on the target Actor-network $\mathrm{P}'(\mathrm{s}_{t+1})$. The state s_{t+1} and action a_{t+1} are input to the target Critic network $Q_1'(\theta_1')$ and $Q_2'(\theta_2')$, respectively. The two target Critic networks will calculate the predicted reward q_1 and q_2.
2. The TD target value is calculated using Equation (19), where $\mathrm{Min}(q_1, q_2)$ takes the minimum value of both.

$$y_t \leftarrow r + \gamma \,\mathrm{Min}(q1, q2). \tag{19}$$

3. Select the action based on the actor network, input the state and action into the critical network separately, and let these two networks output the corresponding prediction reward sum.
4. Calculate the TD error. The calculation formula is as Equation (20).

$$\delta_{1,t} = q_{1,t} - y_t, \quad \delta_{2,t} = q_{2,t} - y_t. \tag{20}$$

5. Update the Critic network as Equation (21).

$$\begin{aligned} \theta_1 &\leftarrow \theta_1 - \alpha \cdot \delta_{1,t} \cdot \nabla_w Q_1(s_t, a_t; \theta_1), \\ \theta_2 &\leftarrow \theta_2 - \alpha \cdot \delta_{2,t} \cdot \nabla_w Q_2(s_t, a_t; \theta_2). \end{aligned} \tag{21}$$

6. Update the strategy network every d rounds through the Actor-network output action as Equation (22).

$$\mathrm{w} \leftarrow \mathrm{w} + \beta \cdot \nabla_w P(s_t; \mathrm{w}) \cdot \nabla_w Q_1(s_t, a_t; \theta_1). \tag{22}$$

7. Update the target Actor and Critic network parameters every d rounds as Equation (23).

$$\begin{aligned} w' &\leftarrow \tau w + (1-\tau) w', \\ \theta_1' &\leftarrow \tau \theta_1 + (1-\tau) \theta_1', \\ \theta_2' &\leftarrow \tau \theta_2 + (1-\tau) \theta_2'. \end{aligned} \tag{23}$$

Repeating the above steps for the specified number of rounds, we will get the trained agent. Finally, we output the optimal value of each model through the agent. The process is shown in Algorithm 2.

4 SYSTEM DESIGN

To establish a reliable federated learning process, we developed a framework for trustworthy federated learning (FedDRL). The framework employs a staged reinforcement learning approach to achieve trustworthy federated learning. In the first

Algorithm 2 The process of model weight calculation

Input: Client Models $\{\theta_1^t, \theta_2^t, \theta_3^t, \ldots, \theta_{\mathrm{n}}^t\}$, Round R, Worker Number N, Buffer Memory Pool M

Initialize Learner Parameters: Actor Parameter P(w), Critic Network $Q_1(\theta_1), Q_2(\theta_2)$

Target Actor Parameter $P'(w')$, Target Critic Network $Q_1'(\theta_1'), Q_2'(\theta_2')$

$w' \leftarrow w, \theta_1' \leftarrow \theta_1, \theta_2' \leftarrow \theta_2$

Output: Optimized Client Model Weight $W = \{w_1^t, w_2^t, \ldots, w_{\mathrm{k}}^t\}$

/* Each Worker Sampling Step */

for worker from 1 to N **do**

  $a_t \leftarrow P(s_t, w)$ // Randomly Choose an Act from $P(s_t, w)$

  $\{w_1^t, w_2^t, w_3^t, \ldots, w_k^t\} \leftarrow Step(a_t)$

  $\theta_{global}^t \leftarrow Agg\left(\sum_{i=1}^k w_i^t \theta_i\right)$

  $R_t \leftarrow CaculateReward(ACC_t - ACC_{avg})$

  $M \leftarrow Store(\langle S_t, A_t, R_t, S_{t+1}\rangle)$

/* Center Learner Training Step */

for r from 1 to R **do**

  Randomly Sampling N Batches of Data from M

  $a_{t+1}' \leftarrow P'(s_{t+1})$

  $y \leftarrow r + \gamma Min(Q_1'(s_{t+1}, a_t'), Q_2'(s_{t+1}, a_t'))$

  Update Critic Network $\theta_1 \leftarrow argmin_{\theta_1} \frac{1}{N} \sum (y - Q_{\theta_1}(s,a))^2$

  Update Critic Network $\theta_2 \leftarrow argmin_{\theta_2} \frac{1}{N} \sum (y - Q_{\theta_2}(s,a))^2$

  Every d Rounds:

  Update Actor-Network: $\nabla_w J(w) = N^{-1} \sum \nabla_w Q_{\theta_1}(s,a) \mid_{a=P(s)} \nabla_w P(s)$

  Update Target Critic Network: $\theta_1' \leftarrow \tau\theta_1 + (1-\tau)\theta_1', \theta_2' \leftarrow \tau\theta_2 + (1-\tau)\theta_2'$

  Update Target Actor-Network: $w' \leftarrow \tau w + (1-\tau) w'$

After R Rounds, Save Trained Model

Output Optimized Model Weight W = $\{w_1^t, w_2^t, \ldots, w_k^t\}$

stage, we train agents to accomplish the selection of trustworthy clients to participate in global model fusion. Then, in the second stage, we also use the trained agent to dynamically adjust the fusion weights of each model and finally realize the optimal global model fusion. The framework workflow consists of six steps, as shown in Figure 2.

Step 1 (Local Model Training): Each client downloads the global model, initializes its parameters accordingly, and conducts model training using local private data.

Step 2 (Upload Model): After local model training, each client uploads its model parameters to the server.

Step 3 (Select Trustworthy Clients): Upon receiving client model parameters, the server employs the *SelectTrustClient*(.) algorithm to train an agent. Subsequently, the trained agent selects trustworthy clients.

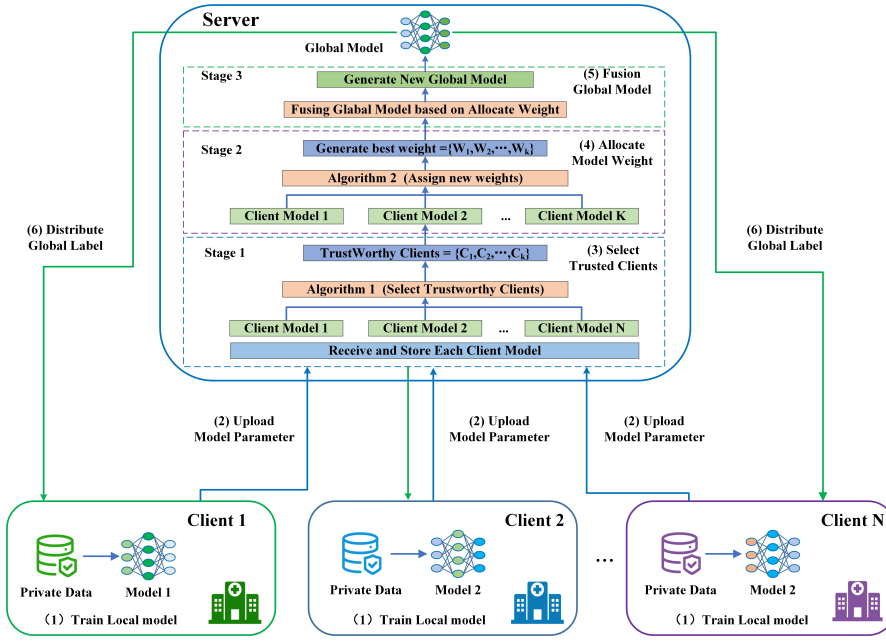


Figure 2. The system architecture of FedDRL

Step 4 (Assigning Model Weights): The server utilizes models from trustworthy clients and performs global model fusion. Then, it employs the *AdaptCalculateWeight*(.) algorithm to train an agent, which optimizes weight assignments for each client model.

Step 5 (Fusing Global Model): The server fuses the global model using the calculated weights from the previous step.

Step 6 (Distribute Global Model): The server disseminates the global model to all clients, initiating the subsequent federation task.

The federation task is set to execute a specified number of communication rounds until the final global model is obtained. This process is shown in Algorithm 3.

Algorithm 3 The FedDRL framework

Input: Private Dataset $\{D_1, D_2, \ldots, D_{\mathrm{n}}\}$, communication round E
Output: The Global model $\{M_{global}\}$

/* Client Process */
for C_i from 1 to N **do**
    $M_i \leftarrow$ GetGlobalModel($round = i$) // Get the global model and init client model
    $M_i \leftarrow$ TrainLocalModel(D_i) // Train model M_i based Dataset $\{D_i\}$
    Server $\leftarrow$ Send(M_i)
/* Server Process */
for e from 1 to E **do**
    Store($\{M_1, M_2, \ldots, M_n\}$) $\leftarrow$ Receive (M_i) // Receive Client Model
    /* FedDRL Algorithm Process */
    Train the Stage 1 Agent
    Update the SelectTrustClient (.) Algorithm parameters // According to Algorithm 1
    $\{M_a, M_b, \ldots, M_{\mathrm{k}}\} \leftarrow$ SelectTrustClient ($\{M_1, M_2, \ldots, M_n\}$)
    Train the Stage 2 Agent
    Update the AdaptCalculateWeight (.) Algorithm parameters // According to Algorithm 2
    $\{W_1, W_2, \ldots, W_{\mathrm{n}}\} \leftarrow$ AdaptCalculateWeight ($\{M_a, M_b, \ldots, M_k\}$)
    $M_{global} \leftarrow$ FusionGlobalModel ($\{W_1, W_2, \ldots, W_n\}$)
    $C_i \leftarrow$ SendGlobalModel (M_{global})

5 EXPERIMENT

5.1 Experiment Setup

5.1.1 Experiment Datasets

We evaluated the FedDRL framework using three distinct image classification datasets:

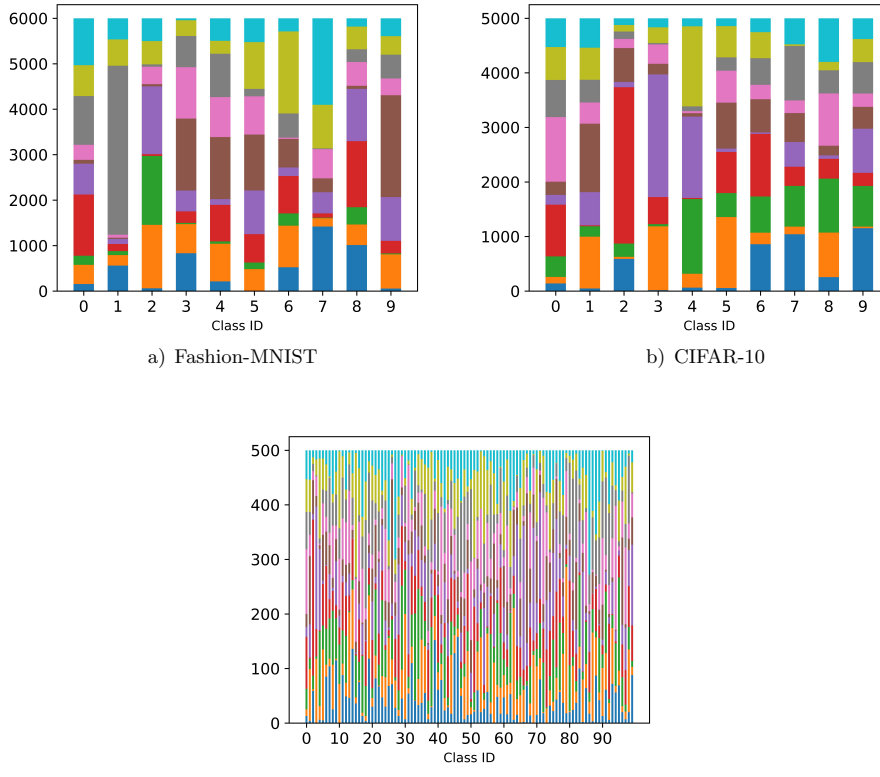
Fashion-MNIST: This dataset includes 60 000 training samples and 10 000 test samples, each a 28×28 grayscale image, classified into one of 10 categories.

CIFAR-10: The CIFAR-10 dataset comprises 60 000 32×32 colour images, evenly distributed across ten classes, each containing 6 000 images.

CIFAR-100: Similar in size to CIFAR-10 but with a broader spectrum, CIFAR-100 features 100 classes with 600 images each, totalling 60 000 colour images.

Data Set Partitioning: For simulating non-IID data distribution among clients. We utilized the Dirichlet function to segregate data across various clients in the open-source dataset. This method can partition the data for each client by adjusting the alpha parameter. As the alpha parameter approaches zero, clients'

data distributions are skewed towards specific classes within the dataset. Conversely, as alpha increases towards infinity. Using the CIFAR-10 dataset as a case study, we set alpha to 1, thereby dividing the three datasets among ten clients. In the figure, Different categories are represented by distinct colours, and the length of each segment within the graphs reflects the sample count within that category. The resulting data distribution is illustrated in Figure 3.



a) Fashion-MNIST

b) CIFAR-10

c) CIFAR-100

Figure 3. The non-IID distribution of 10 clients (alpha = 1)

5.1.2 Comparison of Methods

We contrasted the FedDRL algorithm with two established federated learning approaches.

FedAvg [8]: Serving as the foundational benchmark in federated learning, the FedAvg method determines the weight of each client model based on the proportion of samples contributed by the client relative to the aggregate sample size.

FedProx [10]: Enhancing the FedAvg approach, FedProx incorporates a regularization term within the client model, thereby refining federated learning performance.

5.1.3 Experimental Metrics

We employed accuracy as the metric to gauge the performance of the global model in multi-classification tasks and across individual clients. Assuming n clients engage in model fusion with m communication rounds, the accuracy of the global model in the t^{th} round is denoted as $A^{(t)}_{\text{global}}$. The collective global model accuracies across all rounds are represented as follows:

$$A_{\text{global}} = \{A^1_{\text{global}}, A^2_{\text{global}}, \ldots, A^m_{\text{global}}\}. \tag{24}$$

We denote the accuracy of the c^{th} client's model as A_c. Additionally, we document each client model's accuracy per round, compiling these as follows:

$$A_{\text{c}} = \{A^1_{\text{c}}, A^2_{\text{c}}, \ldots, A^m_{\text{c}}\}, c \in [1, 2, \ldots, n]. \tag{25}$$

5.1.4 Experimental Configuration

Hardware Configuration: The experiments were conducted on a workstation equipped with an Intel i9-12900k CPU, 64 GB RAM, and an NVIDIA RTX3090 GPU.

Software Configuration: We utilized two distinct frameworks for the Federated Learning and Reinforcement Learning experiments. Federated Learning trials were carried out using FedBolt, our custom-built framework, enabling simulation of varied client numbers and data distributions. For reinforcement learning model training, we employed the stablebaseline3 framework, designing two distinct algorithms for trusted client selection and model weight assignment.

Network Setup: We implemented different network architectures tailored to each dataset. For CIFAR-10 and CIFAR-100, a 6-layer CNN was utilized for model training. Conversely, a 4-layer MLP was developed for the Fashion-MNIST dataset.

Agent Network Setup: Implementing a staged reinforcement learning strategy necessitated the training of two distinct agents. The initial phase, adhering to Section 3.2.1, utilizes the A2C algorithm, with each worker and the central node comprising a 6-layer MLP Actor and Critic network. For the second phase, the TD3 algorithm outlined in Section 3.2.2 was employed for agent training, where each module within the TD3 setup incorporates a 6-layer MLP, with further details available in Section 3.2.2.

5.2 Experimental Results

We evaluate the FedDRL framework through four experiments: client attack scenarios, low-quality model fusion, hybrid scenarios, and multi-agent training efficiency. The client attack experiments assess the efficacy of the trustworthy client selection algorithm (stage 1). The low-quality model fusion experiment examines the adaptive weight calculation method (stage 2). The hybrid experiments, combining client attacks and low-quality model elements, validate the comprehensive performance of FedDRL. The final experiment focuses on the training efficiency of multi-agents.

5.2.1 Malicious Client Attack Experiment

In this experiment, we define three types of client-side attacks in federated learning to evaluate our FedDRL framework under adversarial conditions. The experiment spans different client numbers and attack types across three datasets, detailed in Table 1.

Type 1: The client directly uploads the initialized model or makes the model accuracy less than 10 % by modifying the model's hyperparameters.

Type 2: We use falsified data to perform the attack. We use a certain percentage of forged data to participate in model training (e.g., mix the CIFAR-10 dataset with 80 % of CIFAR-100 data and generate these CIFAR-100 data labels as CIFAR-10 corresponding label types). We conduct the attack by faking sample data to train the client's local model, thus reducing the client model's accuracy.

Type 3: We select some clients to simulate the attack and divide the training process of these clients into standard and attack rounds. In the standard round, each client does not perform the attack behavior. Instead, each client deliberately uploads the prepared malicious model in the attack round. We also set that these clients alternately initiate the attack behavior.

According to the experimental setup, we compared the FedDRL algorithm with the FedAvg and FedProx. In the attack experiments, we set the total number of communication rounds to 100 rounds, and each client performs local model training with one epoch. To show the attack behavior of each client and the accuracy of different algorithms in more detail, we counted the accuracy of each client's local model and the accuracy of the server-side global model in each communication round. The specific experimental results are shown in Table 2.

To show the effect of the FedDRL algorithm on global model fusion at each communication round, we conducted experiments using the CIFAR10 dataset on 5, 10, and 15 clients. We compared FedDRL with the FedAvg and FedProx algorithms for the global model accuracy.

We analyze the experimental results for different numbers of client models and other client data. In attack type 2, our algorithm outperforms the FedAvg algorithm

Number of Clients	Attack Type	Malicious ID	Number of Samples	Accuracy of Models ($\leq$)
5	Type 1	Client1	7 750	$A \leq 10\,\%$
	Type 2	Client1	7 750	$10\,\% \leq A \leq 20\,\%$
	Type 3	Client1	7 750	Attack round $A \leq 10\,\%$
10	Type 1	Client1, Client6	4 222, 4 938	$A \leq 10\,\%$
	Type 2	Client1, Client6	4 222, 4 938	$10\,\% \leq A \leq 20\,\%$
	Type 3	Client1, Client6	4 222, 4 938	Attack round $A \leq 10\,\%$
15	Type 1	Client1, Client6, Client11	3 670, 3 314, 4 454	$A \leq 10\,\%$
	Type 2	Client1, Client6, Client11	3 670, 3 314, 4 453	$10\,\% \leq A \leq 20\,\%$
	Type 3	Client1, Client6, Client11	3 670, 3 314, 4 453	Attack round $A \leq 15\,\%$

Table 1. Experimental setup for malicious attack scenarios

DataSet	**Method**	Clients = 5			Clients = 10			Clients = 15		
		Type 1	Type 2	Type 3	Type 1	Type 2	Type 3	Type 1	Type 2	Type 3
Fashion -MNIST	FedAvg	0.862	0.875	0.863	0.792	0.881	0.821	0.776	0.878	0.812
	FedProx	0.873	**0.884**	0.864	0.791	0.882	0.824	0.764	0.879	0.811
	Ours	**0.885**	0.878	**0.883**	**0.877**	**0.886**	**0.887**	**0.881**	**0.886**	**0.882**
Cifar10	FedAvg	0.596	0.691	0.751	0.335	0.681	0.314	0.139	0.664	0.197
	FedProx	0.586	**0.732**	**0.775**	0.331	**0.716**	0.363	0.122	**0.719**	0.202
	Ours	**0.731**	0.701	0.747	**0.694**	0.711	**0.727**	**0.679**	0.702	**0.689**
Cifar100	FedAvg	0.376	0.412	0.298	0.273	0.416	0.287	0.162	0.398	0.176
	FedProx	0.398	**0.442**	0.321	0.223	**0.436**	0.208	0.172	**0.426**	0.183
	Ours	**0.412**	0.438	**0.431**	**0.426**	0.432	**0.421**	**0.423**	0.412	**0.422**

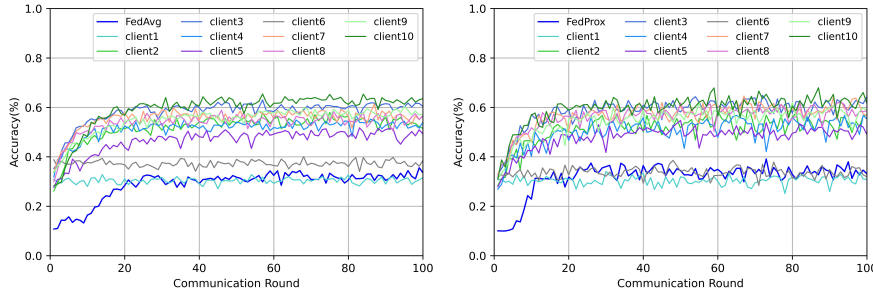
Table 2. Accuracy of each algorithm under different malicious attack scenarios

and slightly underperforms the FedProx algorithm alone. In attack types 1 and 3, our algorithm outperforms the comparison algorithm in most cases, especially when multiple malicious clients are involved in model fusion.

To show the relationship between the global and client model's accuracy in each attack scenario. We conducted more detailed experiments on the Cifar10 dataset.

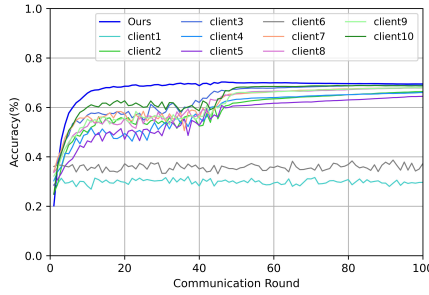
In attack type 1, the global model accuracy plummets with increasing malicious clients under FedAvg and FedProx, dropping below 40 % and 20 % in 10 and 15 client setups, respectively. Conversely, FedDRL's dynamic client selection maintains higher reliability. However, our trained agent can dynamically select trusted clients for model fusion and eliminate malicious models from participating, so our algorithm has higher reliability. The experimental results are shown in Figure 4.

In attack type 2, our algorithm is better than FedAvg but lower than FedProx. The FedProx algorithm uses control parameters to force the models of each client to

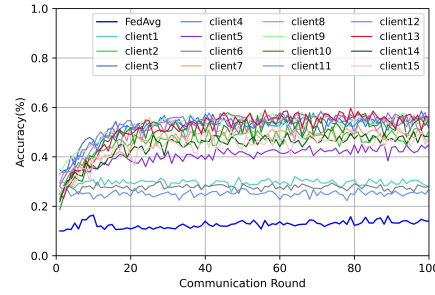


a) 10 clients FedAvg

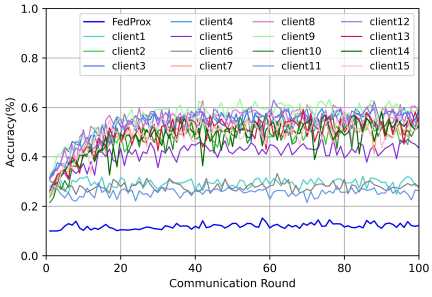
b) 10 clients FedProx



c) 10 clients FedDRL



d) 15 clients FedAvg



e) 15 clients FedProx

f) 15 clients FedDRL

Figure 4. The accuracy of the global model for a different number of clients in the attack type 1

converge to the global model, which will improve the global model's accuracy by improving the malicious model's accuracy to some extent. Our trained agent will filter out low-accuracy models to participate in the fusion after several communication rounds. The experimental results are shown in Figure 5.

In attack type 3 scenarios, the FedAvg and FedProx algorithms experience significant fluctuations in global model accuracy due to alternating attack behaviors by malicious clients. Conversely, the agent within the FedDRL framework adaptively selects trusted clients, effectively excluding malicious entities from participating in model fusion, thereby enabling the FedDRL algorithm to operate with stability. The experimental results are shown in Figure 6.

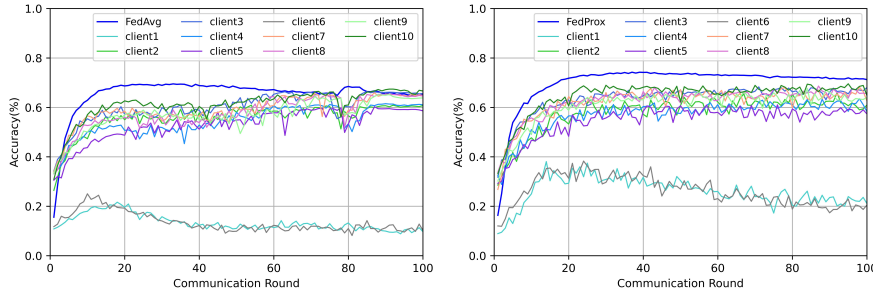
5.2.2 Low-Quality Model Fusion Experiments

In evaluating our FedDRL framework, we undertook validation using the Fashion-MNIST, CIFAR-10, and CIFAR-100 datasets. Given their open-source nature, these datasets are of high quality, leading to minimal variance in model accuracy among clients utilizing them directly. Thus, to simulate real-world conditions, we incorporated low-quality models into the global fusion process. We established a model accuracy threshold, ensuring that models uploaded by low-quality clients did not exceed this threshold in any communication round.

Experiments were carried out on the three datasets, with client groups of varying sizes – 5, 10, and 15 – participating in the global model fusion. We applied a Dirichlet distribution with parameter alpha = 1 to achieve dataset segmentation among clients. We set some clients to upload low-quality models; after several communication rounds, we controlled these client models' accuracy in global fusion, ensuring it remained within the 40 % to 55 % range.

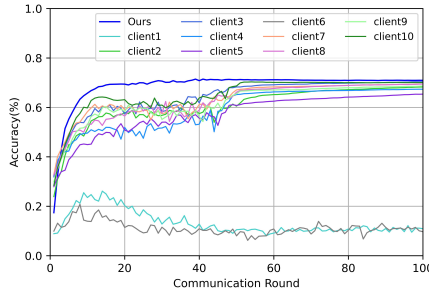
Details of these low-quality model experiment configurations are specified in Table 3. The FedDRL algorithm was compared against the FedAvg and FedProx methods across 100 communication rounds, with each client executing one epoch of local model training. Results are summarized in Table 4.

Employing the CIFAR-10 dataset for illustrative purposes, we performed comparative analyses for setups with 10 and 15 clients, respectively; the findings are depicted in Figure 7. The experiments indicate that the accuracy of the FedAvg and FedProx methods deteriorates as the prevalence of low-quality models increases. This decline can be attributed to these algorithms' reliance on sample count for determining the fusion weight values of the models, where the inclusion of low-quality models adversely impacts the global model's accuracy. Conversely, FedDRL surpasses both methodologies in terms of global model convergence speed and accuracy. This is because FedDRL adaptively recalibrates the weights assigned to each client's model based on quality, thereby diminishing the adverse effects of low-quality models on the global model's accuracy and consequently hastening the global model's convergence rate.

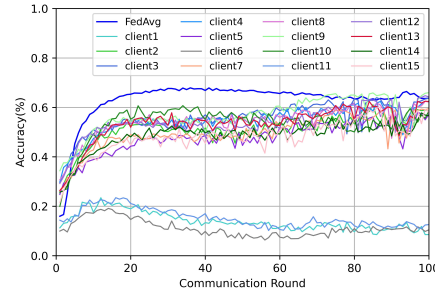


a) 10 clients FedAvg

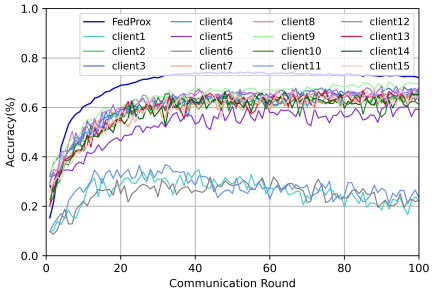
b) 10 clients FedProx



c) 10 clients FedDRL



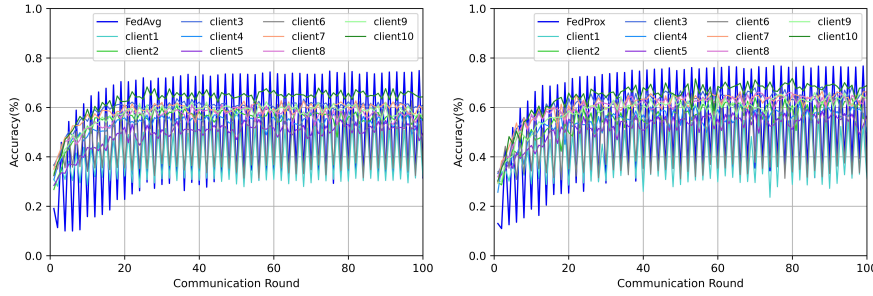
d) 15 clients FedAvg



e) 15 clients FedProx

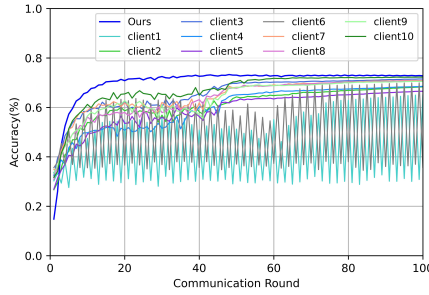
f) 15 clients FedDRL

Figure 5. The accuracy of the global model for a different number of clients in the attack type 2

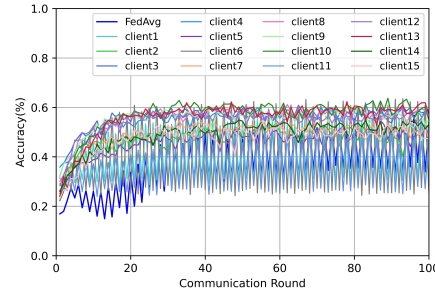


a) 10 clients FedAvg

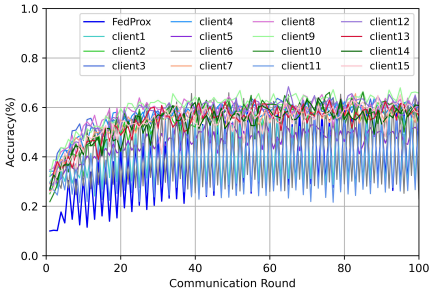
b) 10 clients FedProx



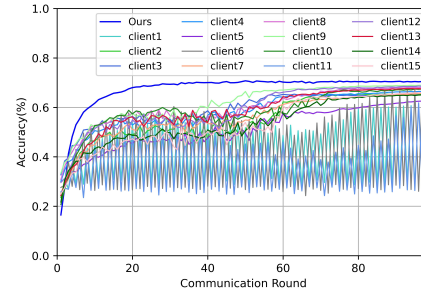
c) 10 clients FedDRL



d) 15 clients FedAvg

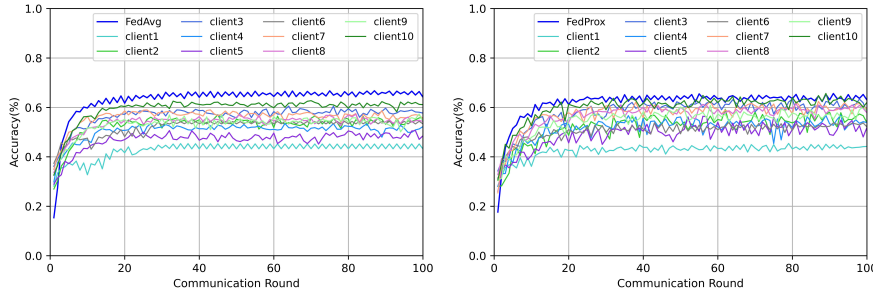


e) 15 clients FedProx



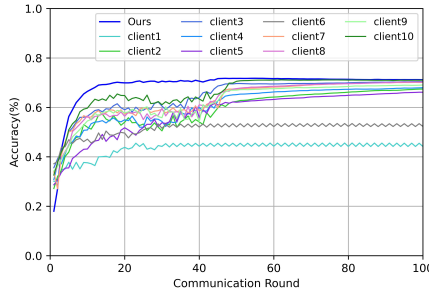
f) 15 clients FedDRL

Figure 6. The accuracy of the global model for a different number of clients in the attack type 3

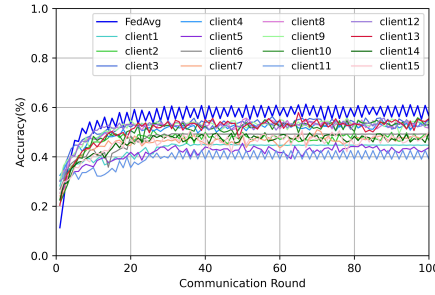


a) 10 clients FedAvg

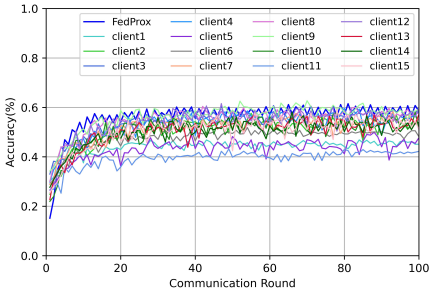
b) 10 clients FedProx



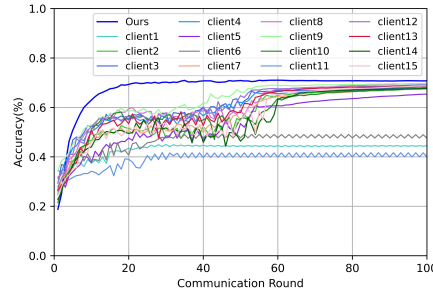
c) 10 clients FedDRL



d) 15 clients FedAvg



e) 15 clients FedProx



f) 15 clients FedDRL

Figure 7. The accuracy of a global model for the different numbers of clients in Low-quality scenario

Number of Clients	Dataset	Low-quality Model ID	Number of Samples	Accuracy of Models (≤)
5	Fashion-MINST	Client1	9 061	53 %
	CIFAR-10	Client1	7 750	52 %
	CIFAR-100	Client1	9 278	22 %
10	Fashion-MINST	Client1, Client5	5 071, 7 245	51 %, 52 %
	CIFAR-10	Client1, Client5	4 222, 6 039	50 %, 54 %
	CIFAR-100	Client1, Client5	4 191, 5 491	22 %
15	Fashion-MINST	Client1, Client5, Client10	4 405, 3 752, 1 809	52 %, 51 %, 53 %
	CIFAR-10	Client1, Client5, Client10	3 670, 3 128, 1 509	49 %, 52 %, 55 %
	CIFAR-100	Client1, Client5, Client10	3 073, 3 494, 2 910	22 % 19 % 23 %

Table 3. Experimental settings for low-quality model experiments

Method	Fashion-MINST			CIFAR-10			CIFAR-100		
	C = 5	C = 10	C = 15	C = 5	C = 10	C = 15	C = 5	C = 10	C = 15
FedAvg	0.857	0.858	0.841	0.705	0.664	0.602	0.386	0.373	0.365
FedProx	0.865	0.861	0.829	0.714	0.652	0.607	0.402	0.391	0.386
Ours	**0.885**	**0.887**	**0.884**	**0.725**	**0.706**	**0.698**	**0.422**	**0.418**	**0.407**

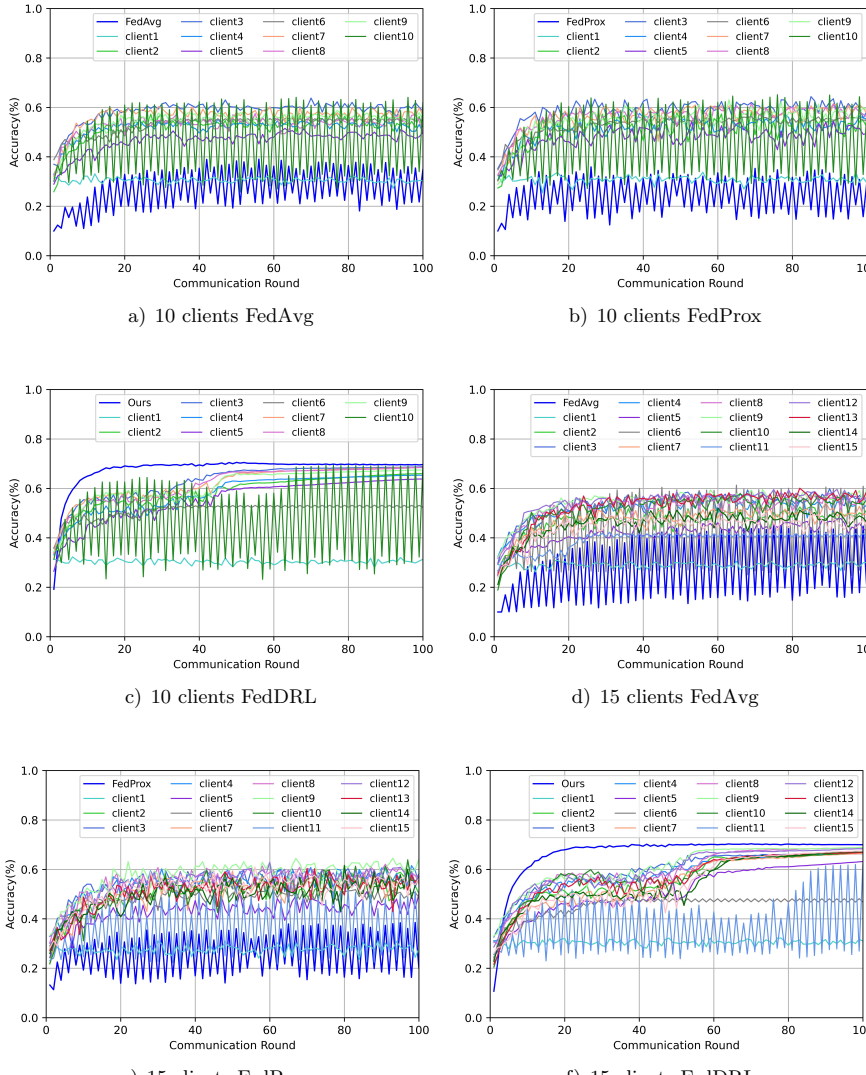
Table 4. Accuracy of each algorithm for low-quality modeling experiments

5.2.3 Hybrid Experiment

In this section, we establish a hybrid scenario incorporating two types of attacking clients (type 1 and type 3) alongside clients submitting low-quality models. We assess the effectiveness of the FedDRL algorithm within this mixed scenario and benchmark it against the FedAvg and FedProx approaches.

Employing the CIFAR-10 dataset, we set different numbers of clients (10, 15) participating in global model fusion, respectively. Client 1 persistently uploads merely the initial model at each round. Client 6 emulates the submission of low-quality models for fusion, and Client 10 or 11 engages in attack behaviour during odd communication rounds but normally participates during even rounds. The remainder of the nodes contribute routinely to each cycle of the federated learning tasks. The experimental setup specifics are delineated in Table 5.

After completing 100 communication rounds, we present the global model accuracy for each algorithm in Table 6. The comparative global model accuracies and individual client model accuracies per communication round, as determined by these three algorithms, are depicted in Figure 8. The experimental outcomes from the hybrid scenario reveal that the FedAvg and FedProx algorithms falter in properly conducting global model fusion due to the adversarial behavior of certain



a) 10 clients FedAvg

b) 10 clients FedProx

c) 10 clients FedDRL

d) 15 clients FedAvg

e) 15 clients FedProx

f) 15 clients FedDRL

Figure 8. Comparison of global model accuracy between different algorithms

Number of Clients	Client ID	Type	Number of Samples	Model Accuracy
10	Client1	Attack Type 1	4 222	$A \leq 10\,\%$
	Client6	Low-quality Model	4 938	$45\,\% \leq A \leq 50\,\%$
	Client10	Attack Type 3	3 560	Attack round $A \leq 15\,\%$
15	Client1	Attack Type 1	3 670	$A \leq 10\,\%$
	Client6	Low-quality Model	3 314	$45\,\% \leq A \leq 50\,\%$
	Client11	Attack Type 3	4 453	Attack round $A \leq 15\,\%$

Table 5. Experimental settings for hybrid scenarios

clients. Incorporating malicious models under traditional algorithmic frameworks significantly degrades the global model's accuracy.

Method	Fashion-MINST		CIFAR-10		CIFAR-100	
	C = 10	C = 15	C = 10	C = 15	C = 10	C = 15
FedAvg	0.835	0.823	0.368	0.348	0.223	0.238
FedProx	0.821	0.846	0.308	0.341	0.241	0.266
Ours	**0.876**	**0.883**	**0.701**	**0.698**	**0.426**	**0.418**

Table 6. Accuracy of each algorithm for hybrid scenarios experiments

The experimental outcomes show that FedAvg and FedProx's global model accuracies suffer from malicious attacks due to their weighted average-based fusion, which does not block harmful participants. Conversely, the FedDRL algorithm, through its two-stage approach, initially filters out malicious models from fusion and subsequently applies an adaptive weight strategy to diminish the impact of substandard models. Consequently, our algorithm maintains operational integrity even within this complex scenario.

5.2.4 Agent Training Efficiency in the FedDRL Framework

In this segment, our primary objective is to assess the training efficiency of agents within the FedDRL framework. To expedite the training process, we have implemented optimizations in two key areas. Initially, we adopted a distributed reinforcement learning methodology, enabling multi-agents to interact concurrently with the external environment. Concurrently, we introduced a memory cache module designed to prevent redundant sampling by multiple agents.

Experimental Scenarios: Our investigation encompasses varied attack scenarios across two distinct datasets: Fashion-MNIST and Cifar-10. In each scenario, we involve a total of 10 and 15 clients in the federated task, including 2 and 3 malicious clients accordingly.

Comparison Experiments: To ascertain the efficacy of the FedDRL framework, we initiated experiments featuring 1, 5, 10, and 20 agents. To guarantee the

stability of the reward values acquired by the final agents, we designated the number of iterations for each experimental group to be 10 000, 15 000, 20 000, and 25 000, correspondingly.

Experimental Metrics: Our evaluation involves counting the iterations necessary for reinforcement learning to reach stable rewards across different agent counts. We employ a sliding window approach to compute the average reward, depicting the progression of rewards attained by the agents. We define r_t the agent's reward obtains in the t^{th} interaction and the sliding window as W. The formula for calculating the average reward is represented as Equation (26):

$$\bar{R} = \frac{1}{W}\sum_{t=1}^{W} r_t. \tag{26}$$

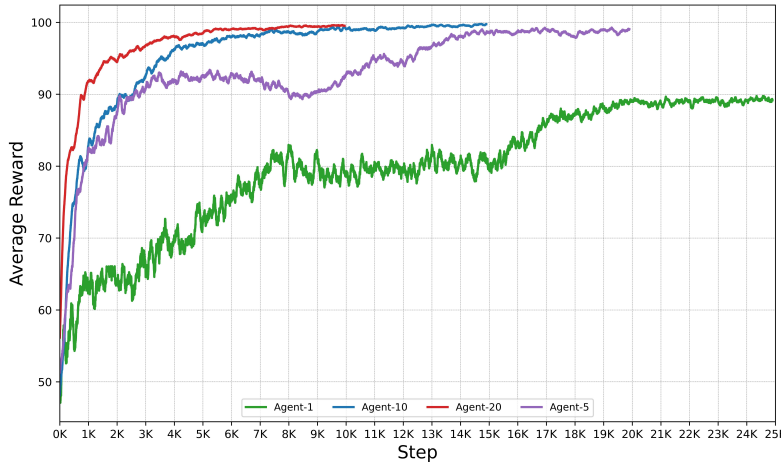
Reward Parameter Setting: Our reward function comprises two components: the global model accuracy reward and the reward for the number of credible nodes. For this experiment, these parameters are set to $\alpha = 100$ and $\beta = 10$, respectively.

Dataset	Attack Type	The Number of Agents			
		$N = 1$	$N = 5$	$N = 10$	$N = 20$
Fashion-MNIST	Type 1	25 000	20 000	16 000	**8 000**
	Type 2	25 000	21 000	15 000	**9 000**
CIFAR-10	Type 1	25 000	20 000	12 000	**10 000**
	Type 2	25 000	20 000	14 000	**9 000**

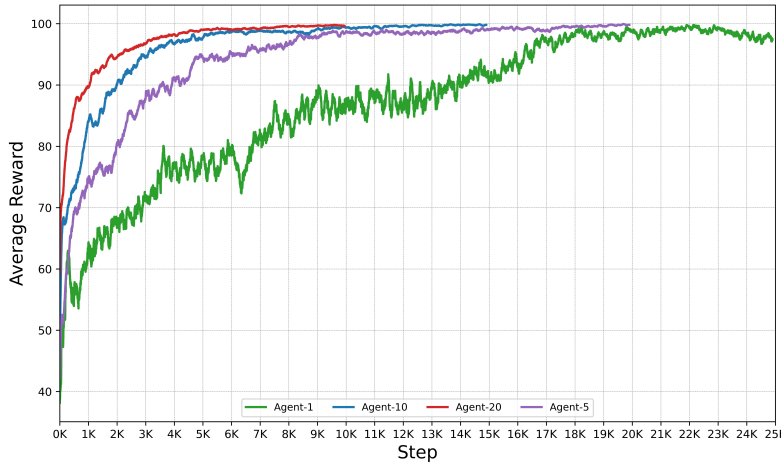
Table 7. The iterations of obtaining stable rewards for the different numbers of agents

Experimental Results: In accordance with our experimental setup, we recorded the reward values for each iteration of the agents, as detailed in Figure 9. We systematically arranged this information into Table 7 for enhanced clarity regarding the actual iterations across different experiments.

The data reveals a notable trend: The single agent does not get the optimal reward in some attack scenarios, because the single agent is easy to fall into the local optimal solution. Meanwile, an increase in agents correlates with reducing the iterations required to achieve a stable reward. However, this relationship is not strictly proportional because the multi-agent independently train their respective Actor and Critic networks. Each agent necessitates a distinct number of iterations to ensure the stability of its individual networks. Nevertheless, the simultaneous interaction of multiple agents with the environment markedly decreases the sampling time, demonstrating a clear trade-off between computational resources and time. This strategy underlines the significant computational resources required, highlighting a deliberate exchange of increased computational demand for reduced computational time.



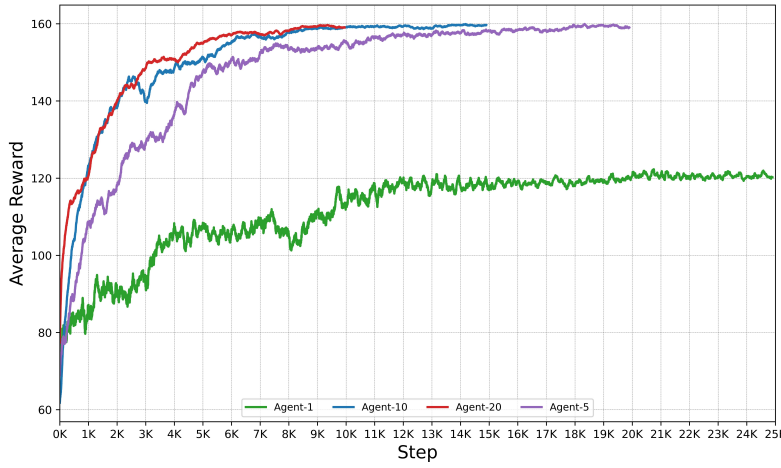
a) Fashion-MNIST (Attack Type 1)



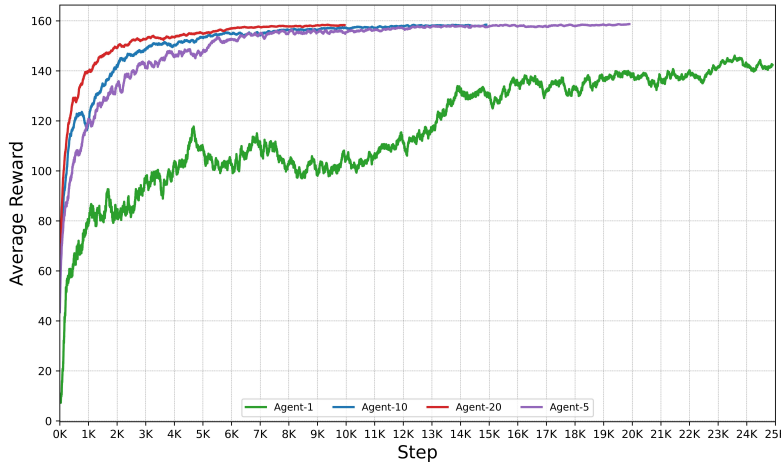
b) Fashion-MNIST (Attack Type 2)

6 CONCLUSION

To the realize trustworthy federated learning, we propose a trusted reinforcement learning framework (FedDRL) based on staged reinforcement learning. The framework comprises two phases: selecting trusted clients and adaptive weight assignment. In the first phase, we design a reward strategy to train the agent, which allows the trained agent to exclude malicious client models from participating in the model fusion based on the environment, and it also adaptively selects trustworthy clients



c) Cifar-10 (Attack Type 1)



d) Cifar-10 (Attack Type 2)

Figure 9. The iterations of obtaining stable rewards for different numbers of agents

for the model fusion. In the second phase, we design a dynamic model weight calculation method, which can adaptively calculate the corresponding weights based on the model quality of each client. In addition, we propose a distributed reinforcement learning method to accelerate agent training. Finally, we design five model fusion scenarios to validate our approach, and the experiments show that our proposed algorithm can work reliably in various model fusion scenarios while maintaining the global model accuracy.

Although a multi-agent distributed reinforcement learning approach can accelerate the agent training process, it sacrifices computational resources for the computational time. In our future work, we will continue to explore more lightweight and trustworthy federated learning methods. We will also investigate more efficient reinforcement learning methods for credible federated learning.

Acknowledgments

The work is supported by the Singapore Ministry of Education (AcRF Tier 1 RG91/22 and NTU startup fund), the National Natural Science Foundation of China (No. 62072469), and the China Scholarship Council (No. 202206450035).

REFERENCES

[1] Li, H.—Sun, X.—Zheng, Z.: Learning to Attack Federated Learning: A Model-Based Reinforcement Learning Attack Framework. In: Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., Oh, A. (Eds.): Advances in Neural Information Processing Systems 35 (NeurIPS 2022). Curran Associates, Inc., 2022, pp. 35007–35020, https://proceedings.neurips.cc/paper_files/paper/2022/file/e2ef0cae667dbe9bfdbcaed1bd91807b-Paper-Conference.pdf.

[2] Wang, H.—Kaplan, Z.—Niu, D.—Li, B.: Optimizing Federated Learning on Non-IID Data with Reinforcement Learning. IEEE INFOCOM 2020 – IEEE Conference on Computer Communications, 2020, pp. 1698–1707, doi: 10.1109/INFOCOM41043.2020.9155494.

[3] Lillicrap, T. P.—Hunt, J. J.—Pritzel, A.—Heess, N.—Erez, T.—Tassa, Y.—Silver, D.—Wierstra, D.: Continuous Control with Deep Reinforcement Learning. CoRR, 2015, doi: 10.48550/arXiv.1509.02971.

[4] Zhang, P.—Wang, C.—Jiang, C.—Han, Z.: Deep Reinforcement Learning Assisted Federated Learning Algorithm for Data Management of IIoT. IEEE Transactions on Industrial Informatics, Vol. 17, 2021, No. 12, pp. 8475–8484, doi: 10.1109/TII.2021.3064351.

[5] Yang, W.—Xiang, W.—Yang, Y.—Cheng, P.: Optimizing Federated Learning with Deep Reinforcement Learning for Digital Twin Empowered Industrial IoT. IEEE Transactions on Industrial Informatics, Vol. 19, 2023, No. 2, pp. 1884–1893, doi: 10.1109/TII.2022.3183465.

[6] 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 Journal on Selected Areas in Communications, Vol. 39, 2021, No. 12, pp. 3688–3703, doi: 10.1109/JSAC.2021.3118352.

[7] Rjoub, G.—Wahab, O. A.—Bentahar, J.—Bataineh, A.: Trust-Driven Reinforcement Selection Strategy for Federated Learning on IoT Devices. Computing, Vol. 106, 2024, No. 4, pp. 1273–1295, doi: 10.1007/s00607-022-01078-1.

[8] McMahan, B.—Moore, E.—Ramage, D.—Hampson, S.—y Arcas, B. A.: Communication-Efficient Learning of Deep Networks from Decentralized Data. In:

Singh, A., Zhu, J. (Eds.): Proceedings of the 20th International Conference on Artificial Intelligence and Statistics. Proceedings of Machine Learning Research (PMLR), Vol. 54, 2017, pp. 1273–1282, http://proceedings.mlr.press/v54/mcmahan17a/mcmahan17a.pdf.

[9] Karimireddy, S. P.—Kale, S.—Mohri, M.—Reddi, S.—Stich, S.—Suresh, A. T.: SCAFFOLD: Stochastic Controlled Averaging for Federated Learning. In: Daumé III, H., Singh, A. (Eds.): Proceedings of the 37th International Conference on Machine Learning. Proceedings of Machine Learning Research (PMLR), Vol. 119, 2020, pp. 5132–5143, http://proceedings.mlr.press/v119/karimireddy20a/karimireddy20a.pdf.

[10] Li, T.—Sahu, A. K.—Zaheer, M.—Sanjabi, M.—Talwalkar, A.—Smith, V.: Federated Optimization in Heterogeneous Networks. In: Dhillon, I., Papailiopoulos, D., Sze, V. (Eds.): Proceedings of Machine Learning and Systems 2 (MLSys 2020). 2020, pp. 429–450, https://proceedings.mlsys.org/paper_files/paper/2020/file/1f5fe83998a09396ebe6477d9475ba0c-Paper.pdf.

[11] Wang, J.—Liu, Q.—Liang, H.—Joshi, G.—Poor, H. V.: Tackling the Objective Inconsistency Problem in Heterogeneous Federated Optimization. 2020, pp. 7611–7623, https://proceedings.neurips.cc/paper_files/paper/2020/file/564127c03caab942e503ee6f810f54fd-Paper.pdf.

[12] Li, Q.—He, B.—Song, D.: Model-Contrastive Federated Learning. Proceedings of the IEEE/CVF Computer Vision and Pattern Recognition (CVPR), 2021, pp. 10708–10717, doi: 10.1109/CVPR46437.2021.01057.

[13] Chen, L.—Zhao, D.—Tao, L.—Wang, K.—Qiao, S.—Zeng, X.—Tan, C. W.: A Credible and Fair Federated Learning Framework Based on Blockchain. IEEE Transactions on Artificial Intelligence, 2024, doi: 10.1109/TAI.2024.3355362.

[14] Zhao, Y.—Li, M.—Lai, L.—Suda, N.—Civin, D.—Chandra, V.: Federated Learning with Non-IID Data. CoRR, 2018, doi: 10.48550/arXiv.1806.00582.

[15] Zhang, X.—Hong, M.—Dhople, S.—Yin, W.—Liu, Y.: FedPD: A Federated Learning Framework with Adaptivity to Non-IID Data. IEEE Transactions on Signal Processing, Vol. 69, 2021, pp. 6055–6070, doi: 10.1109/TSP.2021.3115952.

[16] Gong, B.—Xing, T.—Liu, Z.—Xi, W.—Chen, X.: Adaptive Client Clustering for Efficient Federated Learning over Non-IID and Imbalanced Data. IEEE Transactions on Big Data, 2022, doi: 10.1109/TBDATA.2022.3167994.

[17] Huang, Y.—Chu, L.—Zhou, Z.—Wang, L.—Liu, J.—Pei, J.—Zhang, Y.: Personalized Cross-Silo Federated Learning on Non-IID Data. Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 35, 2021, No. 9, pp. 7865–7873, doi: 10.1609/aaai.v35i9.16960.

[18] Li, X.—Jiang, M.—Zhang, X.—Kamp, M.—Dou, Q.: FedBN: Federated Learning on Non-IID Features via Local Batch Normalization. CoRR, 2021, doi: 10.48550/arXiv.2102.07623.

[19] Briggs, C.—Fan, Z.—Andras, P.: Federated Learning with Hierarchical Clustering of Local Updates to Improve Training on Non-IID Data. 2020 International Joint Conference on Neural Networks (IJCNN), 2020, pp. 1–9, doi: 10.1109/IJCNN48605.2020.9207469.

[20] Gao, L.—Fu, H.—Li, L.—Chen, Y.—Xu, M.—Xu, C. Z.: FedDC: Federated Learning with Non-IID Data via Local Drift Decoupling and Correction. 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2022, pp. 10112–10121, doi: 10.1109/CVPR52688.2022.00987.

[21] Mu, X.—Shen, Y.—Cheng, K.—Geng, X.—Fu, J.—Zhang, T.—Zhang, Z.: FedProc: Prototypical Contrastive Federated Learning on Non-IID Data. Future Generation Computer Systems, Vol. 143, 2023, pp. 93–104, doi: 10.1016/j.future.2023.01.019.

[22] Chen, L.—Zhang, W.—Dong, C.—Zhao, D.—Zeng, X.—Qiao, S.—Zhu, Y.—Tan, C. W.: FedTKD: A Trustworthy Heterogeneous Federated Learning Based on Adaptive Knowledge Distillation. Entropy, Vol. 26, 2024, No. 1, Art. No. 96, doi: 10.3390/e26010096.

[23] Sun, Y.—Si, S.—Wang, J.—Dong, Y.—Zhu, Z.—Xiao, J.: A Fair Federated Learning Framework with Reinforcement Learning. 2022 International Joint Conference on Neural Networks (IJCNN), 2022, pp. 1–8, doi: 10.1109/IJCNN55064.2022.9892211.

[24] Zhang, S. Q.—Lin, J.—Zhang, Q.: A Multi-Agent Reinforcement Learning Approach for Efficient Client Selection in Federated Learning. Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 36, 2022, No. 8, pp. 9091–9099, doi: 10.1609/aaai.v36i8.20894.

[25] Rjoub, G.—Wahab, O. A.—Bentahar, J.—Cohen, R.—Bataineh, A. S.: Trust-Augmented Deep Reinforcement Learning for Federated Learning Client Selection. Information Systems Frontiers, 2022, pp. 1–18, doi: 10.1007/s10796-022-10307-z.

[26] Yang, N.—Wang, S.—Chen, M.—Brinton, C. G.—Yin, C.—Saad, W.—Cui, S.: Model-Based Reinforcement Learning for Quantized Federated Learning Performance Optimization. GLOBECOM 2022 – 2022 IEEE Global Communications Conference, 2022, pp. 5063–5068, doi: 10.1109/GLOBECOM48099.2022.10001466.

[27] Zhang, W.—Yu, F.—Wang, X.—Zeng, X.—Zhao, H.—Tian, Y.—Wang, F. Y.—Li, L.—Li, Z.: R^2 Fed: Resilient Reinforcement Federated Learning for Industrial Applications. IEEE Transactions on Industrial Informatics, Vol. 19, 2023, No. 8, pp. 8829–8840, doi: 10.1109/TII.2022.3222369.

[28] Chen, L.—Zhang, W.—Xu, L.—Zeng, X.—Lu, Q.—Zhao, H.—Chen, B.—Wang, X.: A Federated Parallel Data Platform for Trustworthy AI. 2021 IEEE 1st International Conference on Digital Twins and Parallel Intelligence (DTPI), 2021, pp. 344–347, doi: 10.1109/DTPI52967.2021.9540175.

[29] Mnih, V.—Badia, A. P.—Mirza, M.—Graves, A.—Lillicrap, T.—Harley, T.—Silver, D.—Kavukcuoglu, K.: Asynchronous Methods for Deep Reinforcement Learning. In: Balcan, M. F., Weinberger, K. Q. (Eds.): Proceedings of the 33rd International Conference on Machine Learning. Proceedings of Machine Learning Research (PMLR), Vol. 48, 2016, pp. 1928–1937, http://proceedings.mlr.press/v48/mniha16.pdf.

[30] Fujimoto, S.—Hoof, H.—Meger, D.: Addressing Function Approximation Error in Actor-Critic Methods. In: Dy, J., Krause, A. (Eds.): Proceedings of the 35th International Conference on Machine Learning. Proceedings of Machine Learning

Research (PMLR), Vol. 80, 2018, pp. 1587–1596, http://proceedings.mlr.press/v80/fujimoto18a/fujimoto18a.pdf.

Leiming Chen received a Master's degree in software engineering from the China University of Petroleum (East China). He is pursuing his Ph.D. degree at the School of Computer Science and Technology, China University of Petroleum (East China). He is also a visiting Ph.D. student at the Nanyang Technological University. His research interests include federated learning, reinforcement learning, and brain-inspired computing.

Weishan Zhang received his Ph.D. degree in mechanical manufacturing and automation from the Northwestern Polytechnical University, Xi'an, China, in 2001. He is Full Professor and the Deputy Head for Research with the Department of Software Engineering, School of Computer and Communication Engineering, China University of Petroleum, Qingdao, China. His research interests include big data platforms, pervasive cloud computing, service-oriented computing, and federated learning.

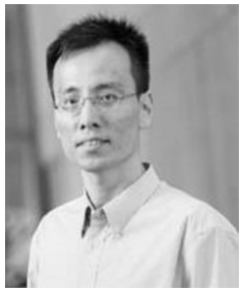
Cihao Dong is pursuing a Master's degree with the Department of Computer Technology, China University of Petroleum (East China). His research interests include graph neural networks, continual learning, and data mining.

Ziling Huang enrolled in the China University of Petroleum (East China) in 2020. He is pursuing a Bachelor's degree at the School of Computer Science and Technology in China University of Petroleum (East China). His current research interests include federated learning and data mining.

Yuming Nie is pursuing a Master's degree with the Department of Computer Technology, China University of Petroleum (East China). Her research interests include time series data analysis and data mining.

Zhaoxiang Hou is an algorithm engineer at the Digital Research Institute of ENN Group. He obtained his Master's degree from the School of Computer Science and Technology, School of Software, China University of Petroleum (East China), in 2022. His research interests include AI and federated learning.

Sibo Qiao received his Master's and Ph.D. degrees at the China University of Petroleum, Qingdao, China, in 2020 and 2023, respectively. He works in the School of Software at the Tiangong University. His research interests include federated learning, deep learning, and image processing.



Chee Wei Tan received his M.A. and Ph.D. in electrical engineering from the Princeton University. He is Associate Professor of computer science and engineering at the Nanyang Technological University. He conducts research in networks, distributed optimization, and generative AI. He has served as IEEE Distinguished Lecturer and Editor for IEEE Transactions on Cognitive Communications and Networking, IEEE/ACM Transactions on Networking, and IEEE Transactions on Communications. He has received the Princeton University Wu Prize for Excellence, the Google Faculty Award, and several teaching excellence awards. He was selected twice for the U.S. National Academy of Engineering China-America Frontiers of Engineering Symposium. He is a Co-Chair of the Cognitive Radio and AI-Enabled Networks Symposium at IEEE GLOBECOM 2025 and a member of ACM Learning at Scale Extended Steering Committee.