Privacy-aware Edge Computation Offloading with Federated Learning in Healthcare Consumer Electronics System

Yiming Fei, Hao Fang, Zheng Yan, Lianyong Qi, Muhammad Bilal, Yang Li, Xiaolong Xu*, and Xiaokang Zhou*

***Abstract*—The continuous iteration of consumer electronics has significantly promoted the development of medical devices, which has enabled the collection of large amounts of heterogeneous medical data. These data are offloaded from local devices to cloud servers for processing using traditional methods, which results in high transmission latency and the risk of privacy leakage. Additionally, researchers have employed federated learning to protect data privacy, but this approach can lead to a straggler effect due to the limited computational and communication resources of terminal devices. To address these issues, a computation offloading framework is designed to optimize task and resource allocation, addressing multi-optimization problems and mitigating the straggler effect in federated learning. Moreover, a novel computation offloading method within a federated learning framework assisted by edge computing, named DRWB, is proposed. Specifically, we develop a deep reinforcement learning-based approach to transfer lagging training tasks to idle edge servers, enhancing data processing speed, minimizing transmission delays, and protecting user privacy. Extensive experimental results demonstrates that the DRWB method outperforms baseline methods, showcasing superior performance in handling heterogeneous medical data tasks.**

***Index Terms*—Smart healthcare, Deep reinforcement learning, Federated Learning, Edge computing, Computation offloading**

I. Introduction

INTEGRATING wearable biosensors into the Internet of Things (IoT) framework is transforming the healthcare industry, particularly in smart healthcare [1]. For example, smartwatches from major manufacturers can collect sensitive user data daily. This private data, processed and analyzed with machine learning, provides users with personalized services and recommendations [2]. In a more medical-specific context, wearable biosensors offer healthcare providers detailed, patient-specific data [3]. By analyzing this data with deep learning models, healthcare providers can achieve more accurate diagnoses, timely interventions, and customized treatment plans. This personalized approach undoubtedly has the potential to significantly enhance patient health outcomes [4].

In conventional centralized machine learning systems, the data from the smart medical framework has to be sent to a central server for analysis. However, as medical devices increase in number, the data collected by them is also growing exponentially [5]. If a purely centralized solution is used for data processing, communication delays will be caused due to insufficient bandwidth, which will also lead to data processing delays [6]. In addition, placing these sensitive data on a central server or outsourcing them to a third party for data processing may lead to serious privacy and security issues. Data related to health information is highly bound to individuals and is extremely sensitive and private [7]. With the anticipated rise in the number of wearable devices, the generation of health data will become increasingly decentralized. A straightforward centralized data processing approach will be neither secure nor practical in such a scenario. The emergence of federated learning (FL) effectively addresses these challenges. FL enhances data privacy by enabling models to be trained locally on devices. As a result, data security is improved, and both latency and data transmission costs are reduced [8].

Nevertheless, due to the heterogeneity and large volume of medical data that needs to be processed on terminal devices(TDs), a purely FL framework inevitably faces the problem of training lag. The laggard effect is caused by different nodes having inconsistent training speeds due to variations in computing power, network conditions, and other factors, thus slowing down the training progress of the overall model [9]. This problem is particularly prominent in the field of smart medical care, due to significant variations in the speed and accuracy of data collection among different sensors, and the transmission to the TD is also significantly different. At the same time, the processor capabilities of the TD are also different [10].

Recent advancements in smartphone processors have made

Yiming Fei is with the School of Software, Nanjing University of Information Science and Technology, Nanjing 210044, China. (e-mail: 202212210038@nuist.edu.cn).

Hao Fang is with Reading Academy, NanJing University of Information Science and Technology, Nanjing 210044, China. (e-mail: fanghuor@163.com).

Zheng Yan is with the School of Cyber Engineering, Xidian University, Xi'an, Shaanxi, 710026, China (email: zyan@xidian.edu.cn).

Lianyong Qi is with the College of computer science and technology, China University of Petroleum (East China), Qingdao, China. (email: lianyongqi@gmail.com).

Muhammad Bilal is with the School of Computing and Communications, Lancaster University, United Kingdom. (email: m.bilal@ieee.org).

YangYang Li is with the College of Mechanical and Electrical Engineering, Shihezi University, Shihezi 832003, China (email: liyang328@shzu.edu.cn).

Xiaolong Xu is a professor in the School of Software, Nanjing University of Information Science and Technology, Nanjing 210044, China. (e-mail: xlxu@ieee.org).

Xiaokang Zhou is with the Faculty of Business and Data Science, Kansai University, Japan. (email: zhou@kansai-u.ac.jp).

*Xiaolong Xu and Xiaokang zhou are the co-corresponding authors.

on-device data processing a viable option [11]. However, the limited computational capacity and battery life of these devices pose substantial constraints, making it impractical to process extensive volumes of heterogeneous data exclusively on them [12]. In our envisioned framework, the bulk of processing tasks are offloaded to strategically positioned edge servers. These servers, with their robust capabilities, are inherently equipped to handle such intensive computational tasks. Thus, although FL and edge computing have demonstrated great potential in medical data processing and privacy protection, efficiently integrating these two technologies remains a pressing issue that needs to be addressed [13].

In this context, we propose an innovative task offloading methodology grounded in deep reinforcement learning. By dynamically evaluating factors such as data size, task priority, and edge server proximity, our method determines the most efficient offloading strategies. This intelligent allocation of processing tasks between end devices and edge servers optimizes system efficiency and reduces latency. Consequently, our approach enhances the performance of smart healthcare systems and significantly improves the overall quality of patient care. The main contributions of this paper include:

- To enhance medical data processing speed, we propose a computation offloading model within a federated learning framework that optimizes task and resource allocation, addressing the multi-optimization problem.
- Propose a deep reinforcement learning-based method to optimize offloading strategies, enhancing efficiency by offloading intensive tasks to idle edge servers, reducing the computational burden on end devices, and protecting patient privacy.
- Comparative analysis with simulated and real datasets shows that our method consistently outperforms existing approaches, verifying the efficiency and reliability of our offloading strategy.

The paper is structured as follows: Section II reviews existing research on edge computing in smart healthcare. Section III describes the proposed system model, including problem formulation and constraints. Section IV details the design of the DRWB. Section V presents experimental results, comparing our methods with existing solutions. Finally, Section VI concludes the paper and suggests future research directions.

II. Related Work

In this section, we review the existing research on smart healthcare and computation offloading within the context of edge computing scenarios.

A. Improvement of Federated Learning

Federated learning (FL) is a groundbreaking machine learning approach. It addresses the problem of data silos and ensures data privacy. In this decentralized system, multiple clients, including mobile devices, institutions, and organizations, work together. They collaborate with central servers without sharing their actual data. This method keeps sensitive information secure while enabling collective learning and model enhancement. The potential of FL has prompted significant scholarly efforts to advance its development. Hanzely et al. [14] present an optimization approach for training FL models. They frame it as an empirical risk minimization problem to create a unified global model. This model is trained using private data that is distributed among all participating devices. Acar et al. [15] introduce a FL approach tailored to specific objectives for edge devices. Their method involves training a global meta-model collaboratively via a server, with each device locally customizing the trained model to meet its specific needs. Li et al. [16] present PyramidFL, an approach that accelerates FL training and enhances final model performance through fine-grained client selection. This method leverages intra-client and inter-client data and system heterogeneity to optimize client utility efficiently.

B. Edge Computing-Assisted Smart Healthcare

The immediate gathering and evaluation of health data demand compliance with standards for latency and energy efficiency. Edge computing, as an emerging paradigm, offers significant assistance to healthcare capabilities in meeting these demands [17]. Numerous researchers have conducted extensive work in this field. Ming et al. [18] introduce a blockchain-enabled platform for edge computing, which utilizes distributed edge servers in conjunction with smart contracts to expedite data processing and authenticate the identity and reliability of network entities. Yadav et al. [19] propose Vehicular Fog Computing (VFC) to optimize overloaded cloudlet resources, reduce latency, and save energy, addressing energy-latency tradeoffs and efficient resource allocation challenges. Das et al. [20] propose an integrated IoT-edge-fog-cloud computing framework, aimed at enhancing green healthcare services through efficient spatio-temporal data analytics and path prediction in exigent situations like natural disasters.

Peng et al. [21] present a novel multi-objective metaheuristic method for computation offloading in healthcare systems using AR applications in edge computing, focusing on preserving user privacy while minimizing latency, energy consumption, and maintaining load balance, demonstrating its efficiency through extensive experiments. Tawalbeh et al. [22] introduce a secure IoT model for healthcare systems, optimized for both low latency and high energy efficiency, analyzing its impact on power and bandwidth through simulations with the Mobile Cloud Computing Simulator (MCCSim), revealing increased delay and power consumption with higher data encryption and decryption rates. Zhou et al. [23] propose a Federated Distillation and Blockchain empowered Secure Knowledge Sharing model to enable efficient and secure data sharing in Internet of Medical Things environments by transforming data sharing into a collaborative model knowledge sharing problem with improved flexibility, reduced communication, and enhanced fairness in node selection and load balancing. Yadav et al. [24] propose the CORL scheme, using reinforcement learning to minimize latency and energy in

IoMT-based healthcare, optimizing resource allocation amidst battery and latency constraints.

C. Deep Reinforcement Learning for Computation Offloading

Since the inception of the edge computing concept, task offloading has become an enduring topic accompanying edge computing, with many researchers conducting extensive work in this area. Wu et al. [25] propose a semi-Markov decision process for vehicular fog computing-assisted platoons, aimed at minimizing offloading delay by considering vehicle resources and random arrivals, with its effectiveness validated through simulation experiments and comparisons with benchmark strategies. Binh et al. [26] propose a novel vehicular edge–cloud computing network tailored for the execution of delay-sensitive tasks within the IoT framework. This network adeptly combines edge and cloud resources for efficient task offloading. They frame the offloading issue as a Markov decision process. To address this, they introduce an advanced task offloading technique based on a dueling actor-insulator network architecture. Chen et al. [27] propose predicting vehicle mobility in IoV-MEC environments to enable efficient, dynamic computation offloading, enhancing QoS and response fairness amidst limited RSU resources.

Although many researchers have conducted extensive work on task offloading in the field of edge computing, few have integrated healthcare Capabilities with task offloading and edge computing to fully harness the potential of artificial intelligence in healthcare. Therefore, we propose a computation offloading strategy based on deep reinforcement learning.

III. System Model

In this section, we begin by presenting a succinct summary of the system model investigated in this study. Next, we delve into the intricacies of the delay model. We then expound on the workload model and the computation model. Ultimately, we encapsulate the problem and express it mathematically. The specific scene diagram is shown in Fig. 1.

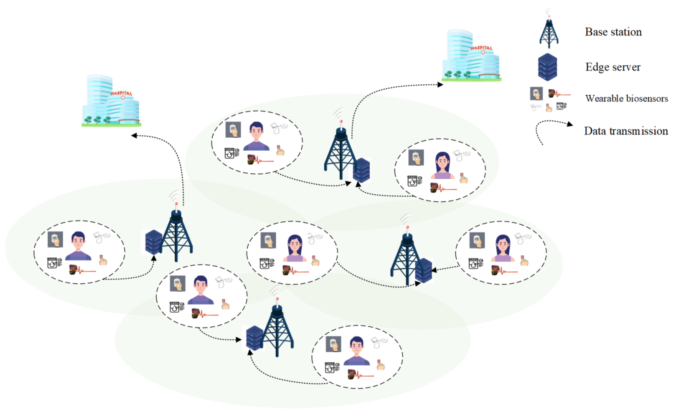


Fig. 1: A framework for Smart Healthcare.

A. Network Model

We focus on a singular event that comprises a series of time slots presented by $T = \{1, \ldots, T\}$. The duration of every time slot within this series is set to Δ seconds. In the context of smart healthcare, the three-tier architecture encompassing TDs, edge nodes, and cloud servers is depicted in Fig. 1.

The set $N = \{n_1, n_2, \ldots, n_i, \ldots\}$ represents the TDs. Due to the typically low transmission power of smart devices like smartwatches and wearable medical equipment, direct data transmission from TDs to edge servers (ES) is often impractical. Instead, smartphones serve as intermediaries, owing to their higher transmission power and broader connectivity capabilities compared to other TDs. Smartphones can effectively relay monitoring data from lower-power devices to edge servers, bridging the gap in transmission requirements. Relevant studies have shown that smartphones can act as efficient intermediaries in data offloading due to their ability to manage both short-range (e.g. Bluetooth, Wi-Fi) and long-range (e.g. 4G, 5G) communications [28].

At each time slot, a computationally intensive task is generated, which places demands on the TD with constrained computing resources C_n. We presume that within a single time slot, the movement of the TDs will not exceed the coverage distance of any ES.

The consortium of ESs is denoted by $S = \{s_1, s_2, \ldots, s_\alpha, \ldots\}$. In the proposed model, it is posited that edge servers are strategically positioned in the vicinity of cellular base stations. The computation resources inherent to an edge server are represented by C_s, while its bandwidth resources are indicated by B_s. Given that both computational and bandwidth resources are finite, TDs at the edge node end make determinations on whether to offload tasks to an ES and choose the suitable server for offloading. When tasks arrive at the edge node, the server elects either to process the tasks or to continue offloading them to the cloud server. Within our framework, we surmise that cloud servers possess ample computational resources, thus, our consideration is confined to the latency involved in offloading tasks from the ES to cloud and the return of results.

In terms of connection modes, two primary modes are considered: E2E (End-to-Edge) and E2C (End-to-Cloud). In the E2E mode, TDs offload tasks directly to the nearest edge server, which can process them locally. In the E2C mode, the edge server offloads tasks to the cloud for further processing. The choice of mode depends on factors such as the computational load of the task, available edge resources, and network conditions.

B. Computation Model

In the proposed computational model, the majority of tasks can be executed on TDs and edge servers, with only a minimal number of tasks requiring transmission to cloud servers. Although the cloud server possesses substantial computational resources, making processing latency relatively negligible, we provide a brief model here for completeness.

When offloading to the cloud, the computational latency of the cloud server for task processing can be defined as:

$$T_{\text{cloud}}^{n_i,t} = \frac{\text{res}_{n_i}^t}{f_{\text{cloud}}}, \tag{1}$$

where $\text{res}_{n_i}^t$ represents the computational requirements of the task from TD n_i, and f_{cloud} is the computation capacity of the cloud server, typically much higher than edge or TDs. Due to the cloud's high processing speed, $T_{\text{cloud}}^{n_i,t}$ generally has a minimal impact on overall latency.

The energy consumption for the cloud server during task processing can similarly be expressed as:

$$E_{\text{cloud}}^t = \xi_{\text{cloud}} \cdot \text{res}_{n_i}^t \cdot (f_{\text{cloud}})^2, \tag{2}$$

where ξ_{cloud} is the energy consumption coefficient of the cloud processor. However, given the vast computational capacity and efficiency of the cloud infrastructure, the impact on total energy consumption is minimal in this context.

The computational latency of TD n_k in t time slot can be represented as:

$$T_{\text{loc}}^{n_i,t} = \frac{\text{res}_{n_i}^t}{f_{n_i}}, \tag{3}$$

where $\text{res}_{n_i}^t$ indicats the computation resources required for the local task. f_{n_i} denotes the computation capacity of the TD n_i. It can be simplified to the frequency of the CPU. The energy consumption of the TD during task processing is expressed as:

$$E_{n_i}^t = \xi \cdot \text{res}_{n_i}^t \, (f_{n_i})^2, \tag{4}$$

where ξ symbolizes as the energy coefficient of the CPU in TDs for processing tasks.

Similarly, when the TD opts to offload the task to the ES for processing, despite the server's more abundant computational resources and superior processing capabilities, the computational delay remains a factor that cannot be overlooked. We denote the set of computational resources allocated by the edge server for the task by $\mathcal{R} = \left\{ f_{s_\alpha}^{n_i} \mid n_i \in N, s_\alpha \in S \right\}$. $\mathcal{F}_{\text{Total}}$ represents the total computational resources available at the ES. From this, we can deduce the following inequality concerning the allocation of computational resources:

$$\sum_{n_i \in N} f_{s_\alpha}^{n_i} \leq \mathcal{F}_{\text{Total}}, \forall s_\alpha \in S. \tag{5}$$

Consequently, we can derive the computational delay incurred by the ES in processing the task, as illustrated by the following formula:

$$T_{s_\alpha}^t = \sum_{s_\alpha \in S} \frac{Z_{n_i,s_\alpha}^t \text{res}_{n_i}}{f_{s_\alpha}^{n_i}}, \forall n_i \in N. \tag{6}$$

Similarly, in accordance with the task offloading policy, we can establish the formula for the energy consumption incurred by the edge server while processing the task, as follows:

$$E_{s_\alpha}^t = \sum_{n_i \in N_{s_\alpha}^t} Z_{n_i,s_\alpha}^t \cdot \lambda \text{res}_{n_i}^t (f_{s_\alpha}^{n_i})^2, \forall s_\alpha \in S, \tag{7}$$

where λ represents the energy consumption coefficient of the ES's processor when processing the task.

C. Communication Model

In this subsection, the communication model will be introduced. Commencing with the Shannon Formula, the maximum achievable upload rate, if the TD opts to upload a task, is defined as follows:

$$r_{n_i}^{upload} = W \log_2 \left(1 + \frac{p_{n_i} h_{n_i}^2}{N_0 + \sum_{i \neq k} p_{n_k} h_{n_k}^2} \right), \tag{8}$$

where $r_{n_i}^{\text{upload}}$ denotes the signal-to-interference-plus-noise ratio (SINR) from TD n_i to the ES. The $\sum_{i \neq k} p_{n_i} h_{n_i}^2$ denotes the interference from other TDs. $W = \frac{B}{N}$ denotes the division of a bandwidth B into N subchannels, where each task requiring upload is allocated at most one subchannel. In other words, during time slot t, a maximum of N tasks can be uploaded simultaneously. For each edge server, the set of all its subchannels is represented by $\mathcal{N} = \{1, 2, 3, \ldots, j\}$.

According to the formula proposed in our network model, when $Z_{n_i} = 1$, it signifies that the TD offloads the task to the ES. The overall offloading strategy set is denoted by $\mathbb{Z} = \left\{ Z_{n_i,s}^j \mid n_i \in N, s \in S, j \in \mathcal{N} \right\}$. In accordance with the offloading policy set and Shannon's formula, we can derive the experimental formula for the task transmission from the TD to the ES, as follows:

$$T_{\text{up}}^{n_i} = \sum_{s_\alpha \in S} \frac{Z_{n_i,s_\alpha}^t \cdot \text{res}_{n_i}}{r_i^{\text{upload}}}, \forall n_i, t \in N. \tag{9}$$

D. Problem Definition

By combining the earlier presented models and constraints in this section, we have developed a unified optimization framework for computation offloading and resource allocation specifically tailored for smart healthcare.

We initially focus on the overall latency function. Latency is a crucial metric for enhancing smart healthcare capabilities. We denote the total latency function by $T(t, n_i, s_\alpha)$, indicating the total transmission time for offloading tasks from TDs to the ES s_α within its range during the t time slot, and the execution time of the task by both the TD and the ES. The formula is as follows:

$$\begin{aligned} T(t, n_i, s_\alpha) = \sum_{n_i \in N_{s_\alpha}^t} & \left(\left(1 - Z_{n_i,s_\alpha}^t\right) T_{\text{loc}}^{n_i,t} \right. \\ & \left. + Z_{n_i,s_\alpha}^t \left(T_{s_\alpha}^t + T_{\text{up}}^{n_i,t} \right) \right). \end{aligned} \tag{10}$$

In the context of task processing, energy expenditure is a key consideration. When TDs locally execute tasks, they incur energy costs primarily due to computational demands. Conversely, when tasks are offloaded to edge servers, the primary energy expense shifts from computation to data transmission. This distinction is crucial in optimizing energy efficiency in TDs, as it alters the primary source of energy consumption from local processing to communication with edge server processors. This nuanced understanding of energy dynamics is vital for designing energy-efficient task execution strategies in distributed computing environments. According to the formula mentioned earlier, the energy consumption of the entire system is represented by $E(t, n_i, s_\alpha)$, where the formula is as follows:

$$E(t, n_i, s_\alpha) = \sum_{n \in N^t_{s_\alpha}} \left(\left(1 - Z^t_{n_i,s_\alpha}\right) E^t_{n_i} + Z^t_{n_i,s_\alpha} E^t_{s_\alpha} \right). \tag{11}$$

Based on formulas (9) and (10), we have defined a utility function U_{cost} and introduced a weighting coefficient ξ, as delineated below:

$$U_{\text{cost}} = \xi T(t, n_i, s_\alpha) + (1 - \xi) E(t, n_i, s_\alpha). \tag{12}$$

From the modeling in this chapter, our optimization problem is described as:

$$\min_{(Z^t_{n_i}, \mathcal{R})} \xi T(t, n_i, s_\alpha) + (1 - \xi) E(t, n_i, s_\alpha), \tag{13}$$

$$\text{s.t.} \quad Z^t_{n_i,s_\alpha} \in \{0, 1\}, \forall n_i \in N, s_\alpha \in S, \tag{14}$$

$$\sum_{n_i \in N} f^{n_i}_{s_\alpha} \leq \mathcal{F}_{\text{Total}}, \forall s_\alpha \in S. \tag{15}$$

Eqs. (14) and (15) establish the rules for task offloading and resource allocation. Eq. (14) specifies that $Z^t_{n_i,s_\alpha}$ is a binary variable, where $Z^t_{n_i,s_\alpha} = 1$ indicates that a task from TD n_i is assigned to edge server s_α at time t, while $Z^t_{n_i,s_\alpha} = 0$ means no offloading occurs to that server. This constraint clearly defines whether a particular edge server is responsible for processing a specific task. Eq. (15) limits the total computational resources allocated to each edge server s_α, ensuring that the sum of allocated resources $\sum_{n_i \in N} f^{n_i}_{s_\alpha}$ does not exceed the maximum available resources, $\mathcal{F}_{\text{Total}}$. This restriction helps prevent server overload, allowing each edge server to function within its capacity for optimal performance.

IV. Design of DRWB

Deep reinforcement learning algorithms utilize deep neural networks. These networks are employed to approximate value functions, thereby guiding intelligent agents to make decisions. DQN is one of them. It combines Q-learning with deep neural networks and provides an effective tool for making decisions in complex environments.

A. DQN

Q-learning is a RL algorithm designed to learn the action-value function $Q(s, a)$. This function estimates the expected return of executing action a in state s. The fundamental formula of Q-learning is:

$$\begin{aligned} Q_{\text{new}}(s_t, a_t) \leftarrow\ & Q(s_t, a_t) + \alpha[r_{t+1} \\ & + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)]. \end{aligned} \tag{16}$$

1) Double DQN: Double DQN (DDQN) consists of two Q-networks: a main network and a target network. The main network calculates the action-state value, guiding the model to select actions and representing the current strategy. The target network evaluates the value of the current state, effectively decoupling the value functions of the main and target networks. This structure helps mitigate the overestimation problem commonly encountered in the Q-learning algorithm. The loss function is as follows:

$$\begin{aligned} L(w) = \mathbb{E}[(r + \gamma Q(s', \arg\max_{a'} Q(s', a', w), w^-) \\ - Q(s, a, w))^2], \end{aligned} \tag{17}$$

where $\arg\max_{a'} Q(s', a', w)$ uses the current network w for action selection, and $Q(s', \arg\max_{a'} Q(s', a', w), w^-)$ uses the target network w^- for value evaluation.

2) Dueling DQN: An innovative aspect of the Dueling DQN is its unique network architecture. This architecture independently estimates the state value. Additionally, it calculates the advantage function for each action. To mitigate action selection bias, the formulation adjusts the calculation of the advantage function, ensuring it remains zero-centered for the chosen policy action [29]. Specifically, the final module of the network is designed to perform the following mapping:

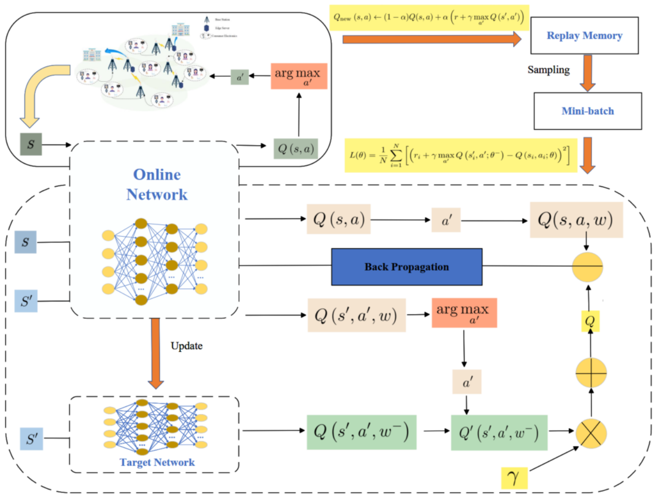
$$\begin{aligned} Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + (A(s, a; \theta, \alpha) - \\ \max_{a' \in \mathcal{A}} A(s, a'; \theta, \alpha)). \end{aligned} \tag{18}$$

This approach normalizes the advantage values by subtracting the maximum advantage observed across all possible actions from the advantage of the selected action. Consequently, this subtraction biases the advantage of the chosen action to zero, thereby facilitating more stable and accurate value estimates and mitigating the risk of overestimating the value of state-action pairs. This formula can also be expressed as:

$$\begin{aligned} Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + \\ \left(A(s, a; \theta, \alpha) - \frac{1}{|\mathcal{A}|} \sum_{a'} A(s, a'; \theta, \alpha) \right). \end{aligned} \tag{19}$$

3) D3QN: D3QN combines the features of DDQN and Dueling DQN, and handles Q-value estimation by using dual networks and advantage function separation [30]. Specifically, D3QN uses one network to select actions and another independent target network to estimate the value of the selected action, while introducing a dueling structure that separates state value and action advantage into the network architecture.

The schematic diagram of D3QN is shown in Fig. 2. Back propagation in the diagram is used to update the weights of the Online Network by minimizing the loss function $L(\theta)$, which measures the difference between the predicted Q-values and the target Q-values. The multiplication sign $(\times)$ represents the discount factor γ applied to the future Q-values, indicating the importance of future rewards. The addition sign $(+)$ combines the weighted current Q-value and the discounted future Q-value to compute the updated Q-value $Q_{\text{new}}(s, a)$, ensuring



$Q_{\text{new}}(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha\left(r + \gamma \max_{a'} Q(s',a')\right)$
Replay Memory
Sampling
Mini-batch
$L(\theta) = \frac{1}{N}\sum_{i=1}^{N}\left[\left(r_i + \gamma \max_{a'} Q(s'_i,a';\theta^-) - Q(s_i,a_i;\theta)\right)^2\right]$
$\arg\max_{a'}$
a'
S
$Q(s,a)$
Online Network
S'
$Q(s,a,w)$
Back Propagation
$Q(s',a',w)$
Q
Update
$Q(s',a',w^-)$
$Q'(s',a',w^-)$
Target Network
γ

Fig. 2: Dueling Double Deep Q Network.

the network considers both immediate and future rewards in its decisions. Combining these two algorithms, the Q-value of D3QN is as follows:

$$\begin{aligned}Q(s,a;\theta,\alpha,\beta) &= V(s;\theta,\beta) + \\ &\gamma\left(A(s,a;\theta,\alpha) - \max_{a'} A(s,a';\theta,\alpha')\right),\end{aligned} \tag{20}$$

where θ and α' are the parameters of the target network, used to estimate the maximum advantage, $V(s;\theta,\beta)$ is the state value function for state s, and $\max_{a'} A(s,a';\theta,\alpha')$ computes the maximum advantage across all possible actions from the target network.

B. Federated Learning

Assume there are K edge nodes in the smart healthcare, we define the optimization objective as follows:

$$\min_{w} f(w) = \sum_{k=1}^{K} \frac{b_k}{n} F_k(w), \tag{21}$$

where b_k denotes the number of data samples at node k, n represents the total number of samples across all nodes, and $\sum_{k=1}^{K} b_k = n$. Here, $F_k(w)$ represents the local loss function for the k-th node, which is used to measure the model's performance on local data at each edge node. In this context, a common choice for $F_k(w)$ is the cross-entropy loss, particularly if we are dealing with classification tasks.

Each edge node optimizes its loss function on its local data:

$$w_k^{(t+1)} \leftarrow w^{(t)} - \eta \nabla F_k(w^{(t)}), \tag{22}$$

where $w^{(t)}$ is the model parameter after the t-th training round, and η is the learning rate. After each training round, each edge node sends its locally updated model $w_k^{(t+1)}$ to the central server.

The central server aggregates all edge nodes' local model updates to create an updated global model. This aggregation is a weighted average of the local models, where each model's weight corresponds to the data size at each node:

$$w^{(t+1)} = \sum_{k=1}^{K} \frac{b_k}{n} w_k^{(t+1)}. \tag{23}$$

This aggregated global model is then distributed back to the edge nodes, which will use it as the starting point for their next training round. This iterative process allows the model to learn from decentralized data while preserving privacy by keeping data on local devices.

After the ES completes the task processing, it returns the results to the TD. Considering that the data volume of the downward transmission is significantly less than that of the upload task, we simplify the model by disregarding the impact of this latency.

C. Computation Offloading Strategy with DRL

In the model we propose, the TDs face an unknown network environment. After generating a task, the TD makes a choice between local processing and uploading to an edge server, representing a model-free learning process. We define this joint optimization problem as a stochastic game:

$$\mathcal{V} = (\mathcal{S}, \mathcal{A}, \mathcal{R}). \tag{24}$$

We consider all TDs $N = \{n_1, n_2, \ldots, n_k\}$ as agents participating in this game, with their states denoted by $\mathcal{S}$, actions by $\mathcal{A}$, and rewards by $\mathcal{R}$. Subsequently, we will provide a detailed analysis of these three elements.

State. Our state representation is a multi-dimensional feature vector that provides a comprehensive snapshot of the current system. It includes crucial data about user equipment (UEs), edge servers, network bandwidth, and other pivotal parameters that influence decision-making in task offloading. This rich representation allows the model to make informed decisions by considering a wide range of factors, including network conditions, computational loads, and resource availability. The state is represented as:

$$\mathcal{S}(t) = \{\Upsilon(t), C_1(t), C_2(t), B(t)\}. \tag{25}$$

where at t time slot, $\Upsilon(t)$ represents the task size, $C_1(t)$ denotes the computational resources of the TDs, $C_2(t)$ signifies the computational resources of ESs, and $B(t)$ indicates ESs' available bandwidth resources.

Action. The action space, a critical component of our model, delineates the feasible decisions that an agent can execute in any given state. It is formulated as a set of tuples $\{(d_i, c_i) | i = 1, 2, \ldots, n\}$, where d_i indicates the binary decision of offloading a task to an edge server (1 for offloading, 0 for local processing, Same as Z above). And c_i represents the allocation of computational resources for the task. We assume that computing resources must be sufficient if tasks are chosen to be executed locally. This approach permits precise control over the task offloading process, permitting the strategic assignment of tasks to optimize resource utilization.

Reward. The reward function is carefully constructed to guide the agent towards making decisions that optimize the use of computational resources while balancing factors like latency and energy consumption. The reward function takes into account various parameters like task execution time, energy efficiency, and system throughput to evaluate the efficacy of an action. Given that the optimization problem aims at concurrently minimizing both delay and energy consumption, our reward function is accordingly defined.

D. DRWB Algorithm

In our design, the central server works alongside clients to collaboratively update the model. This method is akin to incorporating a new client into the existing federated learning framework, without factoring in communication aspects such as data offloading. Consequently, the convergence performance closely resembles that of federated learning. Building on the foundational work on federated learning by McMahan et al. [31], we define the client-side loss function in the following manner:

$$l(w)=\frac{1}{m}\sum_{j=1}^{m} l_j(w). \tag{26}$$

The server-side loss function is expressed as:

$$l(w) = \sum_{p=1}^{P} \frac{1}{B_p - m_p} L_p(w), \tag{27}$$

where B_p represents the data size of client p, and

$$L_p(w) = \frac{1}{m_p} \sum_{j\in \mathcal{Q}_p} l_j(w). \tag{28}$$

Once the clients and server finish their local updates, a global aggregation phase begins. During this phase, the server merges the parameters from all clients with its own model parameters to create a unified global model. The global model is subsequently distributed to all clients. The global model parameter can be defined as follows

$$w = \frac{1}{B}\left(\sum_{p\in\mathcal{P}} (B_p - m_p) w_p + m w_s\right). \tag{29}$$

In our approach, clients experiencing dropouts will offload a portion of their data to the edge server, which aids in collaboratively updating the model. This method is analogous to federated learning, with the primary difference being that the server acts as a more powerful client [32]. Our main goal is to mitigate the effects of client dropouts rather than merely accelerating inference. Therefore, the focus of our design is on the volume of data offloaded. According to the above formula, we demonstrate that partitioning a portion of the data does not impede the convergence of federated learning:

Algorithm 1: Dueling Double Deep Q Network for the offloading strategy

```
1  Initialize prediction network with random weights $\theta$
2  Initialize target network with weights $\theta^- = \theta$
3  Initialize environment $\mathcal{E}$ representing task offloading
     conditions
4  for episode $= 1$ to $M$ do
5      Initialize state $s_1$ based on initial task offloading
         conditions, including computation,
         communication, and energy parameters
6      for $t = 1$ to $T$ do
7          With probability $\epsilon$, select a random action $a_t$
             considering computation, communication, and
             energy consumption
8          Otherwise, select $a_t = \arg\max_a Q(s_t, a; \theta)$,
             where
9          $a_t \in$
             {set of possible offloading decisions considering
10         computation and energy consumption}
11         Execute action $a_t$ in the computation offloading
             environment
12         Observe new state $s_{t+1}$ and reward $r_t$ based on
             computation cost, communication cost, and
             energy consumption
13         Store transition $(s_t, a_t, r_t, s_{t+1})$ in replay
             memory $D$
14         Sample random Mini-batch of transitions from
             $D$: $(s_j, a_j, r_j, s_{j+1})$
15         if episode terminates at step $j+1$ then
16             Set $y_j = r_j$
17         else
18             Set $y_j = r_j +$
                 $\gamma Q(s_{j+1}, \arg\max_{a'} Q(s_{j+1}, a'; \theta), \theta^-)$
19         end
20         Perform a gradient descent step on
             $(y_j - Q(s_j, a_j; \theta))^2$ with learning rate $\alpha$
21         if $t \mod C = 0$ then
22             Update target network: $\theta^- = \theta$
23         end
24     end
25 end
26 return $\theta$
```

$$\begin{aligned} w(t) &= \frac{1}{B}\left[\sum_{p\in\mathcal{P}} (B_p - m_p) w_p(t) + m w_s(t)\right] \\ &= \frac{\sum_{p\in\mathcal{P}} B_p w(t-1)}{B} \\ &\quad - \eta\nabla \frac{\sum_{p\in\mathcal{P}} (B_p - m_p) L_p(w(t-1)) + m L_s(w(t-1))}{B} \\ &= w(t-1) - \eta \nabla L(w(t-1)). \end{aligned} \tag{30}$$

We define two pivotal functions for the simplified offloading strategy: offfloading cost function and offloading decision threshold function. The offfloading cost function is as follows:

$$C(s) = \frac{\text{Required Resources}(s)}{\text{Available Resources}}$$

Here, $C(s)$ represents the ratio of the resources required for task execution under state s to the resources currently available on the device. Required Resources(s) denotes the total resources needed for the task at state s, while Available Resources refers to the current available resources on the device.

The offloading decision threshold function is as follows:

$$\Theta = 1 + \delta$$

In this expression, Θ is the decision threshold used to determine the necessity of offloading, where δ is a small positive number representing a resource buffer to ensure system stability amid uncertainty and dynamic changes.

Based on the aforementioned formulas, we can further explain the uninstallation decision in section III. The offloading decision rule can be defined as follows:

$$\text{Offload Decision} = \begin{cases} \text{True} & \text{if } C(s) \geq \Theta \\ \text{False} & \text{otherwise} \end{cases}$$

If the calculated offloading cost function $C(s)$ exceeds or equals the offloading decision threshold Θ, the decision is made to offload tasks to the ES; otherwise, the task remains to be executed on the local device.

According to the offloading strategy proposed above, Algorithm 1 details a corresponding algorithm. In this algorithm, a selected client offloads a portion of its data to an ES. The ES processes the offloaded data and aggregates the results with the client's trained model. This aggregated model is then uploaded to the central server. Through multiple rounds of aggregation and updating, the client continually refines its model using the aggregated data, significantly reducing inference time. After several rounds of training, the client obtains a robust model that effectively guides its decision-making processes. In the algorithm, we set $\epsilon = 0.1$ for the exploration rate, which determines the likelihood of selecting a random action at each step to encourage exploration. This value was chosen based on a balance between exploration and exploitation, as a lower ϵ favors more deterministic decision-making, while a higher ϵ would increase randomness. The initialization of prediction and target networks, as well as the environment setup, are constant-time operations, with complexity $O(1)$. The outer loop runs for M episodes, each with T time steps, yielding a complexity of $O(M \times T)$. Selecting an action at each time step has complexity $O(A)$, where A is the number of possible actions. For each mini-batch from replay memory D, the Q-value calculation and backpropagation have complexity $O(B \times N)$, where B is the batch size and N the network size. Every C steps, the target network is updated with complexity $O(N)$, as it involves copying N weights. The total time complexity is $O(M \times T \times (A + B \times N))$. Memory grows linearly with the size of replay memory D.

V. Experimental Evaluation

In this section, we evaluate the performance of the DRWB method. To ascertain the efficacy of our proposed algorithm, several control groups have been established for comparative analysis.

A. Parameter Settings

Our experimental procedures were primarily executed on a personal workstation powered by an 11th Gen Intel(R) Core(TM) i7-11700KF @ 3.60GHz , paired with one NVIDIA GeForce RTX 3090 graphics card. During the processing by TDs, the computational capacity of these devices is set to be in the range of [1, 1.5] GHz [33]. When tasks are offloaded to edge servers, the computational capacity of these servers is defined to be within [5, 7] GHz [34]. The energy consumption coefficient is established as $[10^{-26}, 3 \times 10^{-26}]$ [35]. The bandwidth of the edge servers is set to 20 MHz [36].

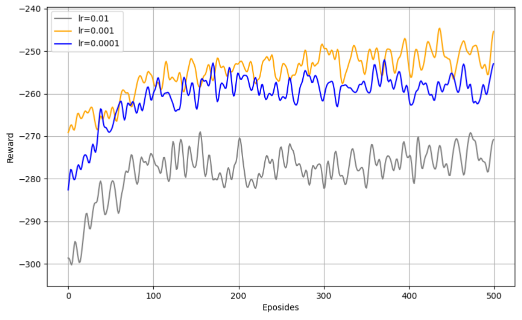
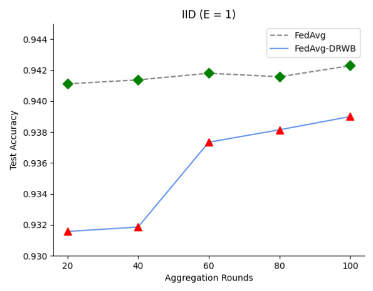


Fig. 3: Convergence performance of different learning rates.

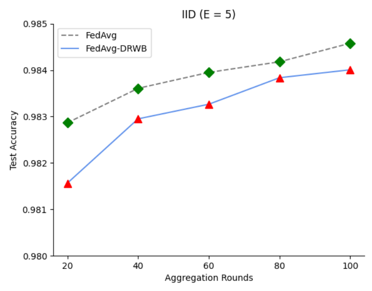
We referred to the initial federated learning scheme [31]. The MNIST data was divided into IID and Non-IID datasets [37]. For the IID data, it was shuffled and then split among 50 clients, with each client receiving 600 examples. For the Non-IID data, the data was first sorted by digit labels and then divided into 100 shards, each containing 300 examples. Each of these shards was assigned to 2 out of the 50 clients. In our experiments, we employed a deep reinforcement learning model with a nine-layer Convolutional Neural Network (CNN) architecture. The network structure consists of layers organized as follows: two 5×5 convolutional layers with 32 filters each, followed by 2×2 max-pooling and local response normalization layers. This setup is repeated before a fully connected layer with 256 nodes, leading to a final fully connected layer with 10 nodes, followed by a softmax layer. The total number of units, z, depends on the dataset, with $z = 1568$ for MNIST-O and MNIST-F and $z = 2048$ for CIFAR-10 [38].

B. Comparative Strategies

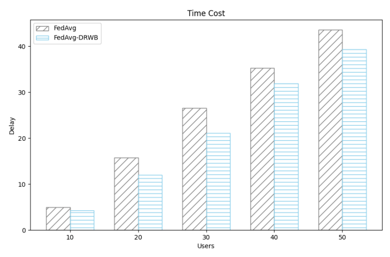
1) **FedAVG [31]:** We chose FedAVG as a baseline due to its foundational role in federated learning, making it a relevant benchmark for evaluating any enhancements in the offloading or resource allocation domains.



(a) Comparison of test accuracy (E = 1)

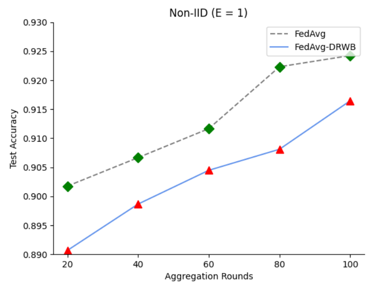


(b) Comparison of test accuracy (E = 5)

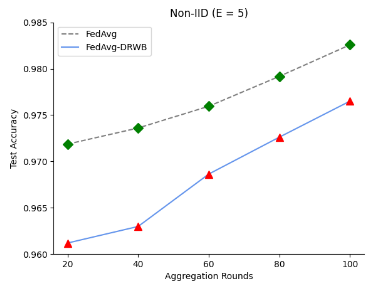


(c) Comparison of time cost (E = 5)

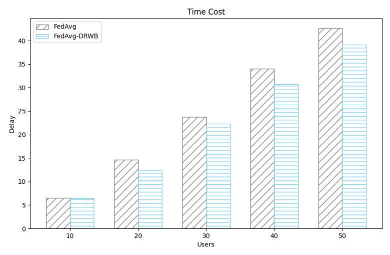
Fig. 4: Learning performance for IID datasets.



(a) Comparison of test accuracy (E = 1)



(b) Comparison of test accuracy (E = 5)



(c) Comparison of time cost (E = 5)

Fig. 5: Learning performance for Non-IID datasets.

2) **DQN:** DQN is a reinforcement learning algorithm that uses a deep neural network to approximate the Q-value function, allowing for efficient decision-making in complex environments.
3) **DDQN:** DDQN is an improvement over DQN, addressing the issue of overestimation in Q-learning by using a separate target network to evaluate Q-values. This helps reduce bias in action-value estimation, which is crucial in applications with resource constraints.
4) **DUELING-DQN:** Dueling-DQN is an advanced version of DQN that separates the estimation of state values and advantages, helping the network to evaluate actions more efficiently in states where specific actions do not significantly affect the outcome.

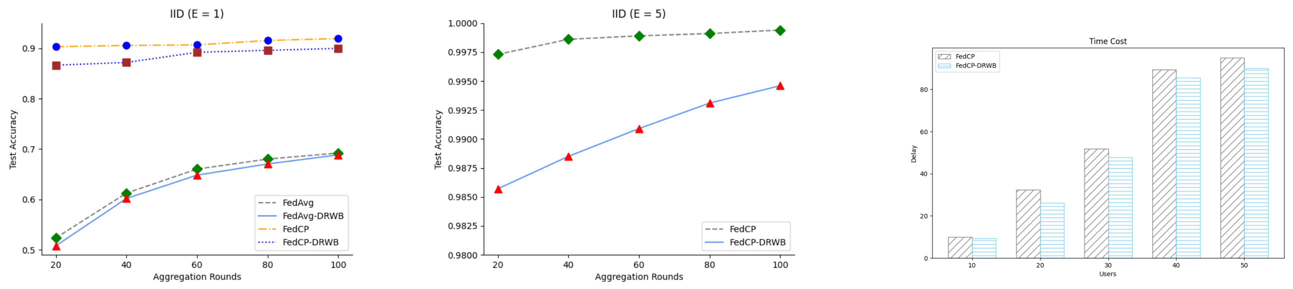
C. *Simulation results*

We selected three learning rates of 0.01, 0.001 and 0.0001 for training. Fig. 3 describes the change of the reward function in 500 training cycles. We selected a learning rate of 0.001 as the learning rate parameter. Compared with the other two learning rates, it has better convergence performance and can converge faster. Due to the existence of the penalty factor and the discount factor, the function still has some fluctuations.

Fig. 4 illustrates the comparative learning performance of federated learning and our proposed approach on an IID dataset. The results show that, despite varying data distributions and the count of epochs in each round, the accuracy of both methods remains quite similar, aligning with our previous analysis. When the aggregation round is relatively small, there is still a certain gap between the two, but when the aggregation round increases, the gap between the two is almost negligible. We also increased the count of training cycles each client completes on its local dataset in each round. And we find that when the training sessions increase, the starting value disparity becomes minimal. When the accuracy is almost the same, our proposed method can effectively shorten the inference time and reduce the lagging effect. Compared to the baseline experiment, our method reduces time cost by 3.75% to 11.68% on the standard dataset.

Fig. 5 illustrates the learning performance of federated learning and our proposed method on a Non-IID dataset. Due to the fact that each device's data distribution does not correspond to the global data distribution, meaning that each device has incomplete class coverage. The accuracy is inevitably lower compared to the IID dataset. However, we also observed as both the number of aggregation rounds and the value of E rise, particularly with an increase in E, the accuracy of our proposed method improves rapidly, becoming almost indistinguishable from that of the standard federated learning algorithm. On the Non-IID dataset, the presented method can effectively reduce the inference time and performs better than on the IID dataset. Compared to the baseline method, our approach achieves an average time cost reduction of 8.97%.

As illustrated in Fig. 6, the Cifar-10 dataset presents more challenges compared to MNIST dataset. We conducted similar experiments utilizing the Cifar-10 dataset. The findings reveal that our proposed method performs consistently with both



(a) Comparison of test accuracy (E = 5)

(b) Comparison of time cost (E = 5)

Fig. 6: Learning performance for the Cifar10 datasets.

the baseline algorithm and the latest personalized federated learning algorithm, FedCP [39], just as it did with the MNIST dataset. Our approach notably decreases processing time while maintaining a high level of accuracy. Compared to the baseline algorithm, our proposed algorithm achieves a latency improvement ranging from 5.45% to 23.47%.

Figure. 7 illustrates the convergence time of different strategies with 20 users and varying join ratios. It is noteworthy that the convergence time of compared method when the join ratio is 20% represents the latency across the full 30 communication rounds, as FedAvg does not reach the target accuracy of 80% within 30 rounds at this participation level. We observe that the convergence time of Fed-DRWB is consistently lower than that of the baseline method, particularly as the join ratio increases. Specifically, Fed-DRWB demonstrates a convergence time that is approximately 1.1 times faster than FedAvg at higher join ratios. Furthermore, as the join ratio increases, the convergence time for both methods decreases. This trend occurs because a higher join ratio implies that a larger portion of users with trained models participate in each communication round, thus accelerating the convergence process.

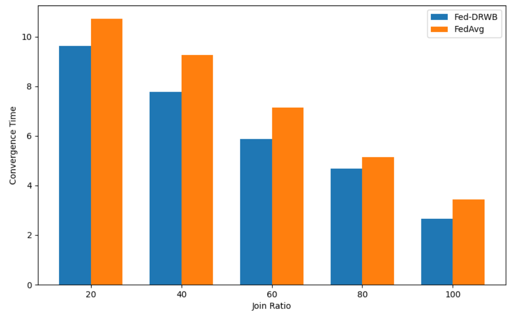


Fig. 7: Convergence time of different strategies in IoV different join ratios.

Based on the data in Fig. 8, our proposed model demonstrates significant efficiency improvements over other models in terms of delay reduction across different numbers of UEs. Specifically, the proposed method reduces delay from 16.7% to 35.5% compared to DDQN, from 10.5% to 24.1% compared to Dueling, and from 15.1% to 21.6% compared to DQN. These reductions become more pronounced as the number of UEs increases, highlighting our method's robust performance in minimizing delay in high-load scenarios.

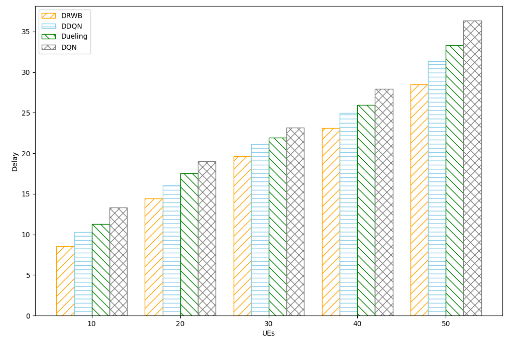


Fig. 8: Delay with different UEs.

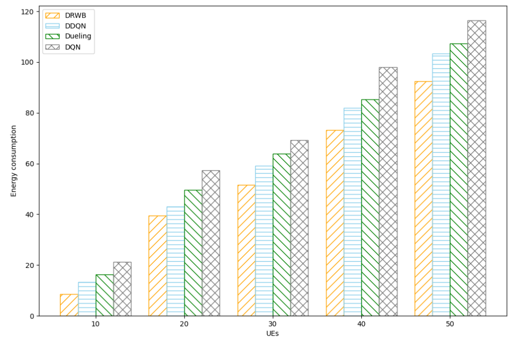


Fig. 9: Energy consumption with different UEs.

In Fig. 9, as the number of users grows, it is observed that the server's energy consumption also rises. Nevertheless, the energy consumption increase associated with our algorithm is considerably lower compared to the other three methods. This efficiency is attributed to our algorithm's superior ability to leverage the idle instances of the edge computer, thereby optimizing energy usage more effectively. Quantitatively, our proposed algorithm reduces energy consumption by 8.4% to 12.9% compared to DDQN, from 13.8% to 20.4% compared to Dueling, and from 20.5% to 31.1% compared to DQN. These results highlight the significant energy-saving potential of our method, especially critical in resource-limited and real-time scenarios typical of smart healthcare.

VI. Conclusion

In this paper, we introduce a novel algorithm based on deep reinforcement learning. This algorithm is specifically designed to alleviate the straggler effect in federated learning. Servers with weaker computational capabilities offload part of the computation to edge servers, significantly improving system efficiency while protecting patient privacy. Our approach is evaluated using well-known public datasets, showcasing its effectiveness.Additionally, we conducted a quantitative analysis of simple task offloading, where our proposed algorithm also demonstrated excellent performance.

In future work, we plan to use more precise medical image datasets for further research. Moreover, due to the mobility of wearable devices, we need to conduct more in-depth studies in dynamic environments.

Acknowledgment

This work was supported by the National Natural Science Foundation of China under Grant (No. 62372242 and 92267104), and in part by Natural Science Foundation of Jiangsu Province of China under Grant (No. BK20211284).

References

[1] A. A. Laghari, K. Wu, R. A. Laghari, M. Ali, and A. A. Khan, "A review and state of art of internet of things (iot)," *Archives of Computational Methods in Engineering*, pp. 1–19, 2021.

[2] X. Zhou, C. K. Leung, I. Kevin, K. Wang, and G. Fortino, "Editorial deep learning-empowered big data analytics in biomedical applications and digital healthcare," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 21, no. 4, pp. 516–520, 2024.

[3] J.-H. Syu, J. C.-W. Lin, G. Srivastava, and K. Yu, "A comprehensive survey on artificial intelligence empowered edge computing on consumer electronics," *IEEE Transactions on Consumer Electronics*, vol. 69, no. 4, pp. 1023–1034, 2023.

[4] P. Dong, Z. Ning, M. S. Obaidat, X. Jiang, Y. Guo, X. Hu, B. Hu, and B. Sadoun, "Edge computing based healthcare systems: Enabling decentralized health monitoring in internet of medical things," *IEEE Network*, vol. 34, no. 5, pp. 254–261, 2020.

[5] J. Liu, Y. Di, X. Zhou, X. Mao, L. Qi, L. Shi, and Y. Dong, "A low-latency synchronization scheme for vehicle information based on cloud-edge collaboration," *IEEE Transactions on Consumer Electronics*, 2024.

[6] X. Zhou, X. Zheng, X. Cui, J. Shi, W. Liang, Z. Yan, L. T. Yang, S. Shimizu, I. Kevin, and K. Wang, "Digital twin enhanced federated reinforcement learning with lightweight knowledge distillation in mobile networks," *IEEE Journal on Selected Areas in Communications*, vol. 41, no. 10, pp. 3191–3211, 2023.

[7] A. Rahman, M. S. Hossain, G. Muhammad, D. Kundu, T. Debnath, M. Rahman, M. S. I. Khan, P. Tiwari, and S. S. Band, "Federated learning-based ai approaches in smart healthcare: concepts, taxonomies, challenges and open issues," *Cluster computing*, vol. 26, no. 4, pp. 2271–2311, 2023.

[8] X. Xu, S. Tang, L. Qi, X. Zhou, F. Dai, and W. Dou, "Cnn partitioning and offloading for vehicular edge networks in web3," *IEEE Communications Magazine*, vol. 61, no. 8, pp. 36–42, 2023.

[9] Z. Ji, L. Chen, N. Zhao, Y. Chen, G. Wei, and F. R. Yu, "Computation offloading for edge-assisted federated learning," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 9, pp. 9330–9344, 2021.

[10] D. Wu, R. Ullah, P. Harvey, P. Kilpatrick, I. Spence, and B. Varghese, "Fedadapt: Adaptive offloading for iot devices in federated learning," *IEEE Internet of Things Journal*, vol. 9, no. 21, pp. 20 889–20 901, 2022.

[11] A. Sabety, "The value of relationships in healthcare," *Journal of Public Economics*, vol. 225, p. 104927, 2023.

[12] M. M. Li, K. Huang, and M. Zitnik, "Graph representation learning in biomedicine and healthcare," *Nature Biomedical Engineering*, vol. 6, no. 12, pp. 1353–1369, 2022.

[13] X. Xu, H. Li, Z. Li, and X. Zhou, "Safe: Synergic data filtering for federated learning in cloud-edge computing," *IEEE Transactions on Industrial Informatics*, vol. 19, no. 2, pp. 1655–1665, 2022.

[14] F. Hanzely and P. Richtárik, "Federated learning of a mixture of global and local models," *arXiv preprint arXiv:2002.05516*, 2020.

[15] D. A. E. Acar, Y. Zhao, R. Zhu, R. Matas, M. Mattina, P. Whatmough, and V. Saligrama, "Debiasing model updates for improving personalized federated training," in *International conference on machine learning*. PMLR, 2021, pp. 21–31.

[16] C. Li, X. Zeng, M. Zhang, and Z. Cao, "Pyramidfl: A fine-grained client selection framework for efficient federated learning," in *Proceedings of the 28th Annual International Conference on Mobile Computing And Networking*, 2022, pp. 158–171.

[17] H. Yan, X. Xu, M. Bilal, X. Xia, W. Dou, and H. Wang, "Customer centric service caching for intelligent cyber–physical transportation systems with cloud–edge computing leveraging digital twins," *IEEE Transactions on Consumer Electronics*, vol. 70, no. 1, pp. 1787–1797, 2024.

[18] Z. Ming, M. Zhou, L. Cui, and S. Yang, "Faith: A fast blockchain-assisted edge computing platform for healthcare applications," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 12, pp. 9217–9226, 2022.

[19] R. Yadav, W. Zhang, O. Kaiwartya, H. Song, and S. Yu, "Energy-latency tradeoff for dynamic computation offloading in vehicular fog computing," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 12, pp. 14 198–14 211, 2020.

[20] J. Das, S. Ghosh, A. Mukherjee, S. K. Ghosh, and R. Buyya, "Rescue: enabling green healthcare services using integrated iot-edge-fog-cloud computing environments," *Software: Practice and Experience*, vol. 52, no. 7, pp. 1615–1642, 2022.

[21] K. Peng, P. Liu, M. Bilal, X. Xu, and E. Prezioso, "Mobility and privacy-aware offloading of ar applications for healthcare cyber-physical systems in edge computing," *IEEE Transactions on Network Science and Engineering*, vol. 10, no. 5, pp. 2662–2673, 2023.

[22] L. Tawalbeh, F. Muheidat, M. Tawalbeh, M. Quwaider, and A. A. Abd El-Latif, "Edge enabled iot system model for secure healthcare," *Measurement*, vol. 191, p. 110792, 2022.

[23] X. Zhou, W. Huang, W. Liang, Z. Yan, J. Ma, Y. Pan, I. Kevin, and K. Wang, "Federated distillation and blockchain empowered secure knowledge sharing for internet of medical things," *Information Sciences*, vol. 662, p. 120217, 2024.

[24] R. Yadav, W. Zhang, I. A. Elgendy, G. Dong, M. Shafiq, A. A. Laghari, and S. Prakash, "Smart healthcare: Rl-based task offloading scheme for edge-enable sensor networks," *IEEE Sensors Journal*, vol. 21, no. 22, pp. 24 910–24 918, 2021.

[25] Q. Wu, S. Wang, H. Ge, P. Fan, Q. Fan, and K. B. Letaief, "Delay-sensitive task offloading in vehicular fog computing-assisted platoons," *IEEE Transactions on Network and Service Management*, pp. 1–1, 2023.

[26] Z. Aghapour, S. Sharifian, and H. Taheri, "Task offloading and resource allocation algorithm based on deep reinforcement learning for distributed ai execution tasks in iot edge computing environments," *Computer Networks*, vol. 223, p. 109577, 2023.

[27] C. Ling, W. Zhang, H. He, R. Yadav, J. Wang, and D. Wang, "Qos and fairness oriented dynamic computation offloading in the internet of vehicles based on estimate time of arrival," *IEEE Transactions on Vehicular Technology*, 2024.

[28] H. Yan, M. Bilal, X. Xu, and S. Vimal, "Edge server deployment for health monitoring with reinforcement learning in internet of medical things," *IEEE Transactions on Computational Social Systems*, 2022.

[29] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, "Dueling network architectures for deep reinforcement learning," in *International conference on machine learning*. PMLR, 2016, pp. 1995–2003.

[30] N. Zhao, Y.-C. Liang, D. Niyato, Y. Pei, M. Wu, and Y. Jiang, "Deep reinforcement learning for user association and resource allocation in heterogeneous cellular networks," *IEEE Transactions on Wireless Communications*, p. 5141–5152, Nov 2019.

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

[32] Z. Ji, L. Chen, N. Zhao, Y. Chen, G. Wei, and F. R. Yu, "Computation offloading for edge-assisted federated learning," *IEEE Transactions on Vehicular Technology*, p. 9330–9344, Sep 2021.

[33] T. X. Tran and D. Pompili, “Joint task offloading and resource allocation for multi-server mobile-edge computing networks,” *IEEE Transactions on Vehicular Technology*, vol. 68, no. 1, pp. 856–868, 2018.
[34] H. Jiang, X. Dai, Z. Xiao, and A. K. Iyengar, “Joint task offloading and resource allocation for energy-constrained mobile edge computing,” *IEEE Transactions on Mobile Computing*, 2022.
[35] N. Chen, S. Zhang, Z. Qian, J. Wu, and S. Lu, “When learning joins edge: Real-time proportional computation offloading via deep reinforcement learning,” in *2019 IEEE 25th International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE, 2019, pp. 414–421.
[36] X. Xu, H. Tian, X. Zhang, L. Qi, Q. He, and W. Dou, “Discov: Distributed covid-19 detection on x-ray images with edge-cloud collaboration,” *IEEE Transactions on Services Computing*, vol. 15, no. 3, pp. 1206–1219, 2022.
[37] J. Zhang, Y. Liu, Y. Hua, H. Wang, T. Song, Z. Xue, R. Ma, and J. Cao, “Pfllib: Personalized federated learning algorithm library,” *arXiv preprint arXiv:2312.04992*, 2023.
[38] S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He, and K. Chan, “Adaptive federated learning in resource constrained edge computing systems,” *IEEE journal on selected areas in communications*, vol. 37, no. 6, pp. 1205–1221, 2019.
[39] J. Zhang, Y. Hua, H. Wang, T. Song, Z. Xue, R. Ma, and H. Guan, “Fedcp: Separating feature information for personalized federated learning via conditional policy,” in *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2023, pp. 3249–3261.