Joint Computation Offloading and Resource Allocation in Multi-Edge Smart Communities With Personalized Federated Deep Reinforcement Learning

Zheyi Chen, *Member, IEEE*, Bing Xiong, Xing Chen, *Member, IEEE*, Geyong Min, *Member, IEEE*, and Jie Li, *Fellow, IEEE*

***Abstract*—Through deploying computing resources at the network edge, Mobile Edge Computing (MEC) alleviates the contradiction between the high requirements of intelligent mobile applications and the limited capacities of mobile End Devices (EDs) in smart communities. However, existing solutions of computation offloading and resource allocation commonly rely on prior knowledge or centralized decision-making, which cannot adapt to dynamic MEC environments with changeable system states and personalized user demands, resulting in degraded Quality-of-Service (QoS) and excessive system overheads. To address this important challenge, we propose a novel Personalized Federated deep Reinforcement learning based computation Offloading and resource Allocation method (PFR-OA). This innovative PFR-OA considers the personalized demands in smart communities when generating proper policies of computation offloading and resource allocation. To relieve the negative impact of local updates on global model convergence, we design a new proximal term to improve the manner of only optimizing local Q-value loss functions in classic reinforcement learning. Moreover, we develop a new partial-greedy based participant selection mechanism to reduce the complexity of federated aggregation while endowing sufficient exploration. Using real-world system settings and testbed, extensive experiments demonstrate the effectiveness of the PFR-OA. Compared to benchmark methods, the PFR-OA achieves better trade-offs between delay and energy consumption and higher task execution success rates under different scenarios.**

***Index Terms*—Mobile edge computing, computation offloading, resource allocation, deep reinforcement learning, personalized federated learning.**

I. Introduction

SMART cities use intelligent technologies to empower community governance and services for improving the efficiency of community running and the quality of citizens' lives [1]. In smart communities, mobile End Devices (EDs) are interconnected via wireless links, forming the Internet-of-Things (IoT) [2]. The EDs commonly own certain capabilities of data collection and task processing that can support emerging intelligent mobile applications to some extent, such as smart transport, smart grids, and autonomous driving. However, due to the limited capacities of computing and power storage on EDs, it is hard to meet the high demands of intelligent mobile applications for low delay and sustainable processing [3]. In cloud computing, computation-intensive and delay-sensitive tasks are usually uploaded to the remote cloud with sufficient resources for execution [4]. However, the long transmission distance between EDs and the cloud often leads to excessive delay, which seriously degrades the Quality-of-Service (QoS) [5].

To alleviate the contradiction between the high demands of intelligent mobile applications and the limited capacities of EDs, the integration of the emerging Mobile Edge Computing (MEC) with Wireless Power Transmission (WPT) is deemed as a feasible and promising solution [6]. In MEC, more computing resources are deployed at the network edge close to EDs, which can extend the computing capacities of EDs by offloading their tasks to MEC servers for execution [7]. Meanwhile, EDs can be charged via WPT to maintain their power demands for long-term running. However, MEC servers are equipped with fewer resources compared to cloud data centers, and thus excessive delay may happen if too many tasks are offloaded simultaneously. Moreover, offloading decisions are constrained and affected by many factors such as the attributes and characteristics of tasks, the power storage of EDs, and the available resource status of MEC servers [8]. Therefore, it is extremely challenging

Manuscript received 26 November 2023; revised 26 March 2024; accepted 30 April 2024. Date of publication 3 May 2024; date of current version 5 November 2024. This work was supported in part by the National Natural Science Foundation of China under Grant 62202103 and Grant U21A20472, in part by the National Key Research and Development Plan of China under Grant 2021YFB3600503, in part by Central Funds Guiding the Local Science and Technology Development under Grant 2022L3004, in part by Fujian Province Technology and Economy Integration Service Platform under Grant 2023XRH001, and in part by Fuzhou-Xiamen-Quanzhou National Independent Innovation Demonstration Zone Collaborative Innovation Platform under Grant 2022FX5. Recommended for acceptance by C. Assi. *(Corresponding author: Xing Chen.)*

Zheyi Chen, Bing Xiong, and Xing Chen are with the College of Computer and Data Science, Fuzhou University, Fuzhou 350116, China, also with the Engineering Research Center of Big Data Intelligence, Ministry of Education, Fuzhou 350002, China, and also with the Fujian Key Laboratory of Network Computing and Intelligent Information Processing, Fuzhou University, Fuzhou 350116, China (e-mail: z.chen@fzu.edu.cn; xiongbingfzu@foxmail.com; chenxing@fzu.edu.cn).

Geyong Min is with the Department of Computer Science, Faculty of Environment, Science and Economy, University of Exeter, EX4 4QF Exeter, U.K. (e-mail: g.min@exeter.ac.uk).

Jie Li is with the Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai 200240, China (e-mail: lijiecs@sjtu.edu.cn).

Digital Object Identifier 10.1109/TMC.2024.3396511

to develop an effective and efficient solution for computation offloading and resource allocation in complex and dynamic MEC environments with multiple constraints.

Most of the classic solutions for computation offloading and resource allocation are based on rules [9], heuristics [10], [11], [12], and control theory [13], [14], [15]. Although they can handle this complex problem to some extent, they commonly rely on some prior knowledge of systems (e.g., state transitions, demand changes, and energy consumption) to formulate appropriate policies for computation offloading and resource allocation. Therefore, they may work well in specific scenarios but cannot well fit in real-world MEC systems with high dynamics and complexity, causing degraded QoS and excessive system overheads. In contrast, Deep Reinforcement Learning (DRL) can better adapt to MEC environments and make policies with higher generalization abilities [16]. Recently, there have been some DRL-based studies on computational offloading and resource allocation. They commonly adopted value-based DRL methods such as Deep Q-Network (DQN) and Double Deep Q-Network (DDQN) [17], [18], [19], whose action space expands exponentially with the increasing number of EDs, resulting in huge complexity. Moreover, the value-based DRL discretizes the continuous space of resource allocation, which may lead to inaccurate policies and undesired results. To better cope with the problem of continuous control, some studies used policy-based DRL methods such as Deep Deterministic Policy Gradient (DDPG) [20], [21], which avoid exponential growth in action space by separating action selection and value evaluation. However, the policy-based DRL is prone to the Q-value overestimation issue, which may cause great fluctuations in the training process and policies falling into the local optimum.

Moreover, most of the existing solutions commonly adopted a centralized training manner, where all the information about EDs may need to be uploaded to a central server. This manner enables models to perform well with rich training samples but might cause severe network congestion and privacy leakage. To ameliorate this problem, it is necessary to perform multi-edge collaboration to achieve better load balancing and system performance, and thus some distributed training manners (e.g., multi-agent DRL) can be deemed as potentially feasible research directions [22]. In multi-agent DRL, each agent regards other agents as environment variables and interacts with the environment independently, and then uses the feedback from the environment to improve its policy. However, when some agents lack training samples, the performance of local models will be seriously limited, making it hard to efficiently achieve model convergence. In contrast, Federated Reinforcement Learning (FRL) implements a collaborative model training on data silos with the original purpose of privacy protection [23]. With FRL, MEC servers only upload their model updates to a central server for federated aggregation without sharing private data during the training process, and the aggregated global model will be distributed to MEC servers for the next round of training on a single agent. Therefore, the FRL can achieve comparable results to the centralized training manner at a faster convergence speed, which can also improve the issue of lacking training samples. However, different smart communities may own personalized demands on QoS and system overheads, and the classic FRL cannot handle this problem because it just naively averages model parameters over MEC servers. Meanwhile, FRL may experience unsatisfying model convergence when facing heterogeneous data, and the training process may move towards the direction of the federated average, but it might not be the global optimum.

To address the above important challenges, we propose a novel Personalized Federated deep Reinforcement learning based computation Offloading and resource Allocation method (PFR-OA). The main contributions of this paper are summarized as follows.

- We design a new multi-edge smart community system consisting of communication, computing, and energy harvesting models, where the task execution delay and energy consumption are formalized as the optimization objectives under multiple constraints.
- For single-edge scenarios, we propose an improved twin-delayed DRL-based algorithm. First, we prove the joint computation offloading and resource allocation as an NP-hard problem. Next, to solve this problem and relieve the negative impact of local updates on global model convergence during distributed training, we design a new proximal term to improve the way of only optimizing local Q-value loss function in classic DRL. Moreover, to avoid the action dispersion caused by the high frequency of error updates in the actor's network, we reduce the variance of action-value estimation by decreasing the frequency of network updates.
- For multi-edge scenarios, we develop a novel personalized FL-based training framework for DRL. During the training process, we consider the personalized demands of smart communities on QoS and system overheads. In particular, the proposed proximal term can attenuate the effect of local update dispersion, enabling the training to quickly converge to the global optimum. Moreover, we design a new partial-greedy based participant selection mechanism, which reduces the complexity of federated aggregation and endows the training with sufficient exploration ability.
- Using the real-world system settings and testbed, extensive experiments are conducted to validate the effectiveness of the proposed PFR-OA. The results show that the PFR-OA achieves better trade-offs between delay and energy consumption and exhibits higher task execution success rates than benchmark methods under different scenarios. Notably, the PFR-OA reaches a faster convergence speed than advanced DRL-based and FRL-based methods. Moreover, we further verify the practicality and superiority of the PFR-OA via real-world testbed experiments.

The rest of this paper is organized as follows. In Section II, we analyze the related work. Section III describes the system model and formalizes the optimization problem. In Section IV, we present the proposed PFR-OA in detail. Section V evaluates the performance of PFR-OA via extensive comparison experiments. In Section VI, we conclude this paper.

TABLE I
COMPARATIVE ANALYSIS OF RELATED STUDIES ("+": INVOLVED, "-": NOT INVOLVED)

Reference	Scenario		Problem		Method						Training Manner		
	Single-edge	*Multi-edge*	*Offloading*	*Resource allocation*	*Rule-based*	*Heuristics*	*Control theory*	*Classic ML*	*Value-based DRL*	*Policy-based DRL*	*Centralized*	*Distributed*	*Personalized*
Ma et al. [9]	-	+	+	-	+	-	-	-	-	-	+	-	-
Chen et al. [10]	+	-	+	-	-	+	-	-	-	-	+	-	-
Ning et al. [11]	-	+	+	-	-	+	-	-	-	-	-	+	-
Ding et al. [12]	+	-	-	+	-	+	-	-	-	-	+	-	-
Ma et al. [13]	-	+	+	-	-	-	+	-	-	-	-	+	-
Hoa et al. [14]	-	+	+	+	-	-	+	-	-	-	-	+	-
Song et al. [15]	-	+	+	+	-	-	+	-	-	-	-	+	-
Xia et al. [24]	-	+	+	+	-	-	+	-	-	-	-	+	-
Mao et al. [25]	-	+	+	+	-	-	-	+	-	-	-	+	-
Wu et al. [17]	-	+	+	+	-	-	-	-	+	-	-	+	-
Hsieh et al. [26]	-	+	+	+	-	-	-	-	-	+	-	+	-
Su et al. [18]	+	-	+	+	-	-	-	-	+	-	+	-	-
Tang et al. [19]	-	+	+	-	-	-	-	-	+	-	-	+	-
Zhang et al. [27]	-	+	+	-	-	-	-	-	-	+	-	+	-
Ho et al. [28]	+	-	+	-	-	-	-	-	+	-	+	-	-
Fan et al. [20]	+	-	+	+	-	-	-	-	-	+	+	-	-
Jiang et al. [21]	-	+	+	-	-	-	-	-	-	+	+	-	-
Hu et al. [29]	+	-	+	+	-	-	-	-	-	+	+	-	-
Liu et al. [30]	-	+	+	-	-	-	-	-	+	-	+	-	-
Liang et al. [31]	-	+	+	-	+	-	-	-	-	-	+	-	-
Deng et al. [32]	+	-	+	+	-	-	+	-	-	-	+	-	-
Gao et al. [33]	-	+	+	-	-	-	-	-	-	+	-	+	-
Dai et al. [34]	-	+	+	-	-	-	-	+	-	-	-	+	-
Nguyen et al. [35]	-	+	+	+	-	-	-	-	-	+	-	+	-
Zhang et al. [36]	-	+	+	+	-	-	-	-	+	-	-	+	-
Zhu et al. [37]	-	+	-	+	-	-	-	-	+	-	-	+	-
Proposed PFR-OA	-	+	+	+	-	-	-	-	-	+	-	+	+

II. RELATED WORK

In recent years, the problems of computation offloading and resource allocation have received much attention, and many scholars have contributed to solving these two important problems. This section reviews and analyzes related studies from three perspectives including classic methods, DRL-based methods, and training manners, where the comparative analysis of related studies is given in Table I.

A. Classic Methods

Classic methods for computation offloading and resource allocation commonly rely on rules, heuristics, and control theory. For example, Ma et al. [9] designed a two-directional auction mechanism and adopted linear programming to improve the computing efficiency of systems. Chen et al. [10] proposed a stochastic optimization algorithm for task offloading and frequency scaling, aiming to minimize system energy consumption. Ning et al. [11] developed an auction-based task scheduling algorithm with delay approximation to enhance system throughput. Ding et al. [12] designed a hybrid offloading strategy in NOMA-based MEC with multi-objective optimization, which reduced the energy consumption and delay of systems. Ma et al. [13] proposed an offloading scheme based on a memory mechanism of long-term and short-term events to improve system throughput. Hoa et al. [14] designed a PSO-based collaborative task offloading method to reduce the average system delay. Song et al. [15] developed a convex-optimization-based offloading method to decrease user-weighted energy consumption. Xia et al. [24] proposed a distributed offloading strategy based on Lyapunov and game theory to reduce communication overheads. Mao et al. [25] designed a privacy-preserving offloading solution for DNNs to lessen communication costs while improving model robustness.

In general, the above studies need to rely on prior knowledge of systems (e.g., state transitions, demand changes, energy consumption) to formulate proper policies of computation offloading and resource allocation, and thus they can only work well in some specific scenarios. Meanwhile, they commonly require excessive iterations to find suitable policies, which makes it difficult to meet the QoS demands of intelligent mobile applications and also causes huge system overheads. Moreover, they usually do not fully consider the dynamics of task arrival and the impact of multiple constraints on the task execution process in MEC systems.

B. DRL-Based Methods

The problem of computation offloading and resource allocation is commonly regarded as non-convex, which is extremely complicated and hard to solve directly. According to the

aforementioned analysis, classic rule-based, heuristic, and control-theoretic methods reveal limitations when they face the problem of computation offloading and resource allocation under multiple constraints in dynamic and complex MEC environments. In contrast, DRL can better fit in MEC environments and make policies for such a dynamic and complicated problem with higher generalization abilities. For example, Wu et al. [17] proposed a DQN-based offloading and resource allocation strategy, aiming to optimize the computational costs of systems. Hsieh et al. [26] adopted an actor-critic DRL-based offloading method to enhance the QoS in heterogeneous MEC networks. Su et al. [18] designed an online computation offloading method based on the Lyapunov optimization and DRL to improve the data processing ability of MEC networks. Tang et al. [19] developed a lightweight offloading solution by integrating LSTM and DQN to relieve network congestion in MEC systems. Zhang et al. [27] proposed an actor-critic DRL-based offloading method with blockchain authorization to optimize system delay and security issues. Ho et al. [28] designed a DQN-based computation offloading and resource allocation method to reduce task processing delay. Fan et al. [20] developed a DDPG-based scheme for jointly optimizing the issues of DNN deployment, task offloading, and resource allocation, aiming to decrease the system running costs. Jiang et al. [21] designed a hierarchical framework based on DDPG and DQN to reduce the task execution delay. Hu et al. [29] proposed a DRL-based computation offloading and energy transmission algorithm to reduce the delay and energy consumption of processing tasks in MEC.

Generally, most of the above studies adopt the value-based DRL to deal with the problem of computation offloading and resource allocation, whose action space grows exponentially with the increasing number of EDs, leading to high complexity and difficulty in training ideal policies. Meanwhile, the value-based DRL discretizes and samples the continuous actions of resource allocation, which seriously degrades the accuracy of selecting actions and deviates policies from optimal performance. Although the policy-based DRL can avoid the issue of action space explosion by separating action selection and value evaluation, it tends to overestimate the Q-value function during the training process, which may cause large training fluctuations and the generated policies to easily fall into the local optimum.

C. Training Manners

The training manners of the existing computation offloading and resource allocation methods can be classified into the centralized training and the distributed training. For example, Liu et al. [30] adopted a centralized training manner to obtain offloading policies with the consideration of user clustering and task priority, aiming to reduce the costs of MEC systems. Liang et al. [31] designed a centralized training way for edge-collaboration offloading scenarios to improve system throughput. Deng et al. [32] proposed an offloading algorithm based on the centralized Lyapunov optimization for enhancing system throughput. Gao et al. [33] developed an attention-based multi-agent DRL framework to lessen the completion rate of tasks with heterogeneous demands. Dai et al. [34] designed a distributed learning-based algorithm for cooperative offloading, aiming to decrease task execution delay. Nguyen et al. [35] devised a multi-agent DDPG-based task offloading method and block-mining technique to improve the utility of blockchain-enabled MEC systems. Zhang et al. [36] proposed an FRL-based task offloading and resource allocation algorithm to optimize task execution delay with the constraints on communication and computation. Zhu et al. [37] designed an FRL-based resource allocation method to maximize resource utilization in edge-based IoT environments.

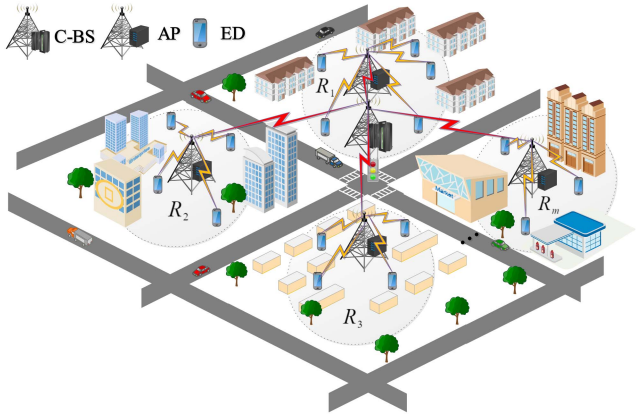


Fig. 1. The proposed system of multi-edge smart communities.

In general, most of the above studies adopt a centralized training manner, where all the information about EDs may need to be uploaded to a central server. Although sufficient training samples allow models to achieve good performance, the issues of severe network congestion and privacy leakage may occur when using a centralized training manner. These issues can be relieved to some extent by using distributed training manners (e.g., multi-agent DRL) that collaborate multiple edges for improving load balancing and system performance, where each agent views other agents as environment variables and interacts with the environment independently. However, if some agents lack training samples, their training effects will be seriously restricted, resulting in poor model performance. In contrast, FRL only uploads model updates without sharing private data, which implements collaborative model training on data silos with a privacy-preserving purpose, and thus the MEC servers that lack training samples can obtain well-performed aggregated models via interacting with the central server. However, the classic FRL just averages the model parameters of MEC servers during the training process, which does not well consider the personalized demands of QoS and system overheads in different edge environments and cannot converge to the global optimum.

III. System Model and Problem Formulation

As illustrated in Fig. 1, the proposed system of multi-edge smart communities consists of a Central Base Station (C-BS) and m smart communities, denoted by the set $R = \{R_i, i \in m\}$. In the smart community R_i, an Access Point (AP) interacts with the C-BS and there are n EDs, donoted by the set $ED_i =$

TABLE II
MAJOR NOTATIONS USED IN THE PROPOSED MODEL

Notation	Definition
R	Set of smart communities
ED_i	Set of EDs in R_i
M_i	MEC server in R_i
h	Time-slot
t	Sub-slot
$Task_{i,j}(h)$	A task generated by the j-th ED in R_i
$D_{i,j}(h)$	Data volume of $Task_{i,j}(h)$
$C_{i,j}(h)$	Required computational resources of $Task_{i,j}(h)$
T_d	Maximum tolerable delay of $Task_{i,j}(h)$
$r_{i,j}(t)$	Uplink date rate of $ED_{i,j}$
$w_{i,j}(t)$	Proportion of bandwidth allocated to $Task_{i,j}(h)$
$B_i(t)$	Available upload bandwidth
$P_{i,j}(t)$	Transmission power of $ED_{i,j}$
$g_{i,j}(t)$	Channel gain between $ED_{i,j}$ and M_i
σ^2	Average power for Gaussian white noise
$l_{i,j}(t)$	Distance between $ED_{i,j}$ and M_i
$T^{tr}_{i,j}(h)$	Delay of uploading $Task_{i,j}(h)$
$E^{tr}_{i,j}(h)$	Energy consumption of uploading $Task_{i,j}(h)$
$T^{l}_{i,j}(h)$	Delay of executing $Task_{i,j}(h)$ on $ED_{i,j}$
$E^{l}_{i,j}(h)$	Energy of executing $Task_{i,j}(h)$ on $ED_{i,j}$
k	Capacitance coefficient
$f_{i,j}$	Computing capability (i.e., CPU frequency) of $ED_{i,j}$
$T^{m}_{i,j}(h)$	Delay of executing $Task_{i,j}(h)$ by M_i
$T^{w}_{i,j}(h)$	Waiting time of $Task_{i,j}(h)$
$E^{m}_{i,j}(h)$	Energy consumption of executing $Task_{i,j}(h)$ by M_i
$\beta_{i,j}(t)$	Proportion of resources allocated to $Task_{i,j}(h)$
$F_i(t)$	Available computational resources of M_i
P_m	Computing power of M_i
$b_{\max}$	Maximum battery capacity of an ED
$b_{i,j}(t)$	Battery power of $ED_{i,j}$
e_t	Amount of harvested energy by an ED

$\{ED_{i,j}, i \in m, j \in n\}$. Each AP is equipped with an MEC server (denoted by M_i) that can process the tasks offloaded by EDs and feedback results, and it can also transmit energy to the EDs within its communication coverage through the wireless network. Moreover, each ED is equipped with a rechargeable battery that can receive and store energy to power the processes of task offloading and processing. For clarity, Table II lists the major notations used in the proposed model.

In the proposed system, we adopt a discrete-time running mode, which contains H time-slots with the same span, where $h = 1, 2, \ldots, H$. At the beginning of h, $ED_{i,j}$ generates a task, denoted by $Task_{i,j}(h) = (D_{i,j}(h), C_{i,j}(h), T_d)$, where $D_{i,j}(h)$ indicates the data volume, $C_{i,j}(h)$ indicates the required computational resources, and T_d indicates the maximum tolerable delay. Different from the existing studies, the task attributes follow the Poisson process within fixed intervals with random values. If a task cannot be completed within maximum tolerable delay and available power, it will be determined to be failed. Specifically, the tasks generated by $ED_{i,j}$ will be placed in its buffer queue, and the tasks that first enter the queue will be completed before subsequently arriving tasks can be executed. Moreover, tasks can be processed locally or offloaded to the MEC server for execution. Furthermore, as shown in Fig. 2, we divide each time-slot into T sub-slots for fine-grained model training, where $t = 1, 2, \ldots, T$, aiming to avoid excessive delays or task failures caused by inappropriate

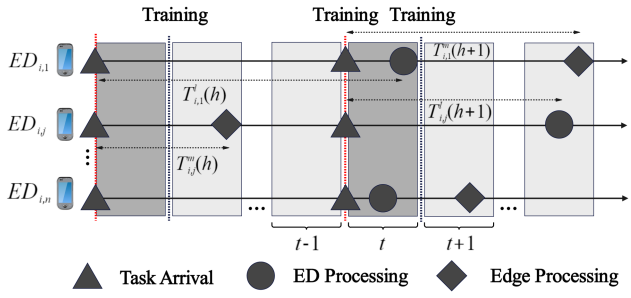


Fig. 2. An example of time-slot division and the life cycle of tasks.

coarse-grained decisions. Moreover, Fig. 2 also illustrates the life cycle of tasks including the task arrival and processing, where triangles indicate the moment that tasks arrive, circles indicate the moment that tasks are completed by EDs, and diamonds indicate the moment that tasks are completed by the edge. Taking $ED_{i,1}$ as an example, $T^{l}_{i,1}(h)$ indicates the time required for processing the task on the ED, where the task is generated by $ED_{i,1}$ at the time-slot h. $T^{m}_{i,1}(h+1)$ indicates the time required for processing the task on the edge, where the task is generated by $ED_{i,1}$ at the time-slot $h+1$. When there are more sub-slots, the waiting time for the tasks on the buffer queue might be reduced with better policies, but it will increase the complexity and time of model training. Therefore, the values of T should be properly chosen for different requirements and scenarios.

A. Communication Model

When uploading $Task_{i,j}(h)$ to M_i for execution via the AP in R_i, the uplink date rate of $ED_{i,j}$ is defined as

$$r_{i,j}(t) = w_{i,j}(t) B_i(t) \log_2 \left(1 + \frac{P_{i,j}(t) g_{i,j}(t)}{\sigma^2 l_{i,j}(t)}\right), \quad (1)$$

where $B_i(t)$ indicates the available upload bandwidth at the sub-slot t, $w_{i,j}(t)$ indicates the proportion of bandwidth allocated to $Task_{i,j}(h)$, $P_{i,j}(t)$ indicates the transmission power of $ED_{i,j}$, $g_{i,j}(t)$ indicates the channel gain between M_i and $ED_{i,j}$, σ^2 indicates the average power of Gaussian white noise, and $l_{i,j}(t)$ indicates the distance between M_i and $ED_{i,j}$.

Thus, the delay of uploading $Task_{i,j}(h)$ is defined as

$$T^{tr}_{i,j}(h) = \frac{D_{i,j}(h)}{r_{i,j}(t)}. \quad (2)$$

Accordingly, the energy consumption of uploading $Task_{i,j}(h)$ is defined as

$$E^{tr}_{i,j}(h) = P_{i,j}(t) T^{tr}_{i,j}(t). \quad (3)$$

Since the results of executing tasks are much smaller than the data volume uploaded by tasks, the delay and energy consumption of downloading results from M_i to $ED_{i,j}$ can be commonly neglectable [38].

B. Computation Model

In the proposed model, all EDs and MEC servers can offer computing services, and thus we consider the local and edge computing modes as follows.

1) Local Computing Mode: When a task is executed on an ED, the delay and energy consumption of executing the task are defined as

$$T_{i,j}^{l}(h) = \frac{C_{i,j}(h)}{f_{i,j}}, \tag{4}$$

$$E_{i,j}^{l}(h) = kC_{i,j}(h)f_{i,j}^{2}, \tag{5}$$

where $f_{i,j}$ indicates the computing capability (i.e., CPU frequency) of $ED_{i,j}$ and k is the capacitance coefficient.

2) Edge Computing Mode: When offloading a task to the MEC server for execution, the delay and energy consumption of executing the task are defined as

$$T_{i,j}^{m}(h) = \frac{C_{i,j}(h)}{\beta_{i,j}(t)F_i(t)}, \tag{6}$$

$$E_{i,j}^{m}(h) = P_m T_{i,j}^{m}(h), \tag{7}$$

where $\beta_{i,j}(t)$ indicates the proportion of computational resources allocated by M_i to $Task_{i,j}(h)$, $F_i(t)$ indicates the available computational resources of M_i, and P_m indicates the computing power of M_i.

C. Energy Harvesting Model

In the proposed system, all EDs are equipped with rechargeable batteries with a maximum capacity of $b_{\max}$. At the beginning of t, the battery power of $ED_{i,j}$ is $b_{i,j}(t)$. During the process of harvesting energy, $ED_{i,j}$ receives energy through WPT and deposits it into the battery in the form of energy packets, and the amount of harvested energy by an ED during t is denoted as e_t, which can be used to execute tasks locally or offload tasks to M_i for execution. Specifically, for different system states during t, we consider different situations of power variations on $ED_{i,j}$ as follows.

- If the task buffer queue of $ED_{i,j}$ is empty, there is only charging but no energy consumption. Thus, at the beginning of $t+1$, the battery power of $ED_{i,j}$ is

$$b_{i,j}(t+1) = \min(b_{i,j}(t) + e_t, b_{\max}). \tag{8}$$

- If $Task_{i,j}(h)$ is completed on $ED_{i,j}$, the battery power of $ED_{i,j}$ at the beginning of $t+1$ is

$$b_{i,j}(t+1) = \min\{\max\{b_{i,j}(t) + e_t - E_{i,j}^{l}(h), 0\}, b_{\max}\}. \tag{9}$$

- If $Task_{i,j}(h)$ fails on $ED_{i,j}$ because $T_{i,j}^{l}(h)$ exceeds T_d or $E_{i,j}^{l}(h)$ exceeds $b_{i,j}(t)$, the battery power of $ED_{i,j}$ at the beginning of $t+1$ is

$$b_{i,j}(t+1) = \min\{\max\{b_{i,j}(t) + e_t - E_{i,j}^{l}(h), 0\}, b_{\max}\}. \tag{10}$$

- If $Task_{i,j}(h)$ is offloaded to M_i and be completed successfully, the battery power of $ED_{i,j}$ at the beginning of $t+1$ is

$$b_{i,j}(t+1) = \min\{\max\{b_{i,j}(t) + e_t - E_{i,j}^{tr}(h) - E_{i,j}^{m}(h), 0\}, b_{\max}\}. \tag{11}$$

- If $Task_{i,j}(h)$ fails because $T_{i,j}^{tr}(h)$ exceeds T_d or $E_{i,j}^{tr}(h)$ exceeds $b_{i,j}(t)$, the battery power of $ED_{i,j}$ at the beginning of $t+1$ is

$$b_{i,j}(t+1) = \min\{\max\{b_{i,j}(t) + e_t - \frac{\frac{h}{T}}{T_{i,j}^{tr}(h)} E_{i,j}^{tr}(h), 0\}, b_{\max}\}. \tag{12}$$

- If $Task_{i,j}(h)$ fails on M_i because $T_{i,j}^{m}(h)$ exceeds T_d or $E_{i,j}^{m}(h)$ exceeds $b_{i,j}(t)$, the battery power of $ED_{i,j}$ at the beginning of $t+1$ is

$$b_{i,j}(t+1) = \min\{\max\{b_{i,j}(t) + e_t - E_{i,j}^{tr}(h) - \frac{\frac{h}{T} - T_{i,j}^{tr}(h)}{T_{i,j}^{m}(h)} E_{i,j}^{tr}(h), 0\}, b_{\max}\}. \tag{13}$$

D. Formulation of Optimization Objective

Based on the above system models, the delay and energy consumption of executing a task with different offloading decisions are respectively defined as

$$T_{i,j}(h) = (1-\alpha_{i,j}(t))T_{i,j}^{l}(h) + \alpha_{i,j}(t)(T_{i,j}^{tr}(h) + T_{i,j}^{m}(h)) + T_{i,j}^{w}(h), \tag{14}$$

$$E_{i,j}(h) = (1-\alpha_{i,j}(t))E_{i,j}^{l}(h) + \alpha_{i,j}(t)(E_{i,j}^{tr}(h) + E_{i,j}^{m}(h)), \tag{15}$$

where $\alpha_{i,j}(t) \in \{0,1\}$ is the offloading decision, which indicates that a task will be executed locally or offloaded to the MEC server for execution. The waiting time of $Task_{i,j}(h)$ is denoted as $T_{i,j}^{w}(h)$, indicating the time from task generation to the beginning of task execution.

To minimize the delay and energy consumption of executing tasks, the optimization objective is formulated as

$$\begin{aligned} (P1)\ & \min_{\alpha,\beta,w} \sum_{j=1}^{n}\sum_{h=0}^{H} q_{i,t}T_{i,j}(h) + q_{i,e}E_{i,j}(h) \\ s.t. & C1: \alpha_{i,j}(t) \in \{0,1\} \\ & C2: T_{i,j}(h) \le T_d \\ & C3: E_{i,j}(h) \le b_{i,j}(t) \\ & C4: \sum_{j=1}^{n} \alpha_{i,j}(t)w_{i,j}(t) = 1 \\ & C5: \sum_{j=1}^{n} \alpha_{i,j}(t)\beta_{i,j}(t) = 1 \end{aligned}, \tag{16}$$

where $q_{i,t}$ and $q_{i,e}$ are the weights of delay and energy consumption, which also indicate the personalized demands of various users' tasks in different smart communities for delay and energy consumption, respectively. $C1$ indicates that a task can only be executed locally or offloaded to the MEC server for execution. $C2$ indicates that the delay of executing a task cannot exceed the maximum tolerable delay. $C3$ indicates that the energy consumption of executing a task cannot exceed the

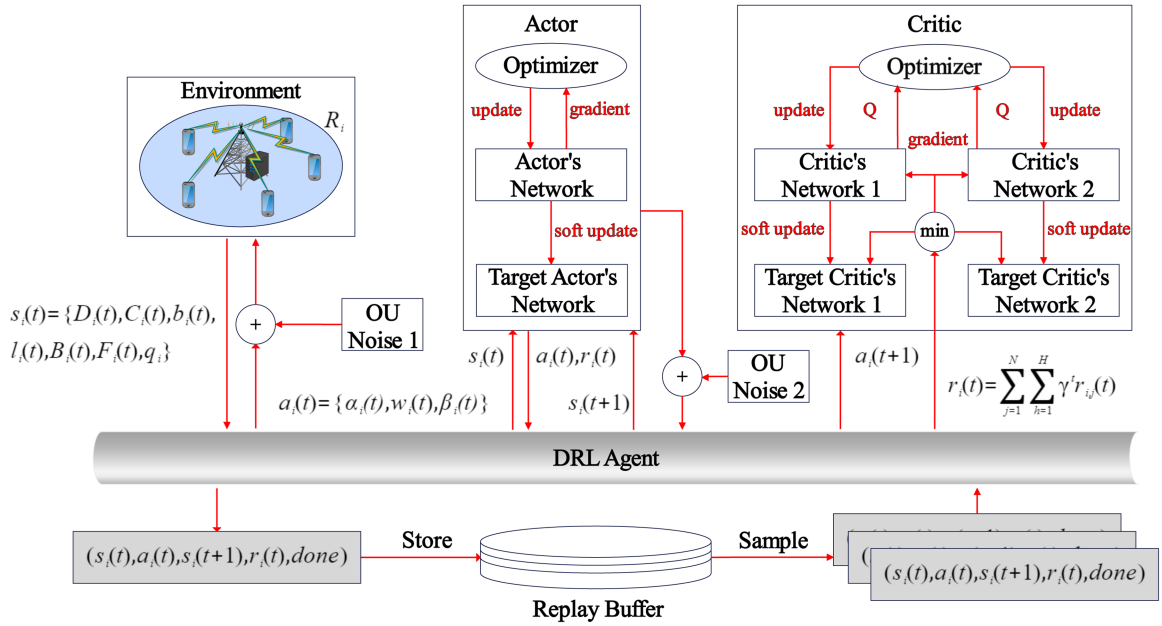


Fig. 3. Improved twin-delayed DRL-based computation offloading and resource allocation for single-edge scenarios.

available battery power of an ED. $C4$ indicates that the sum of the proportion of bandwidth allocated for uploading tasks should be 1. $C5$ indicates that the sum of the proportion of the computational resources allocated for executing offloaded tasks should be 1.

IV. The Proposed PRF-OA

To address the above optimization problem, we propose a novel Personalized Federated deep Reinforcement learning based computation Offloading and resource Allocation method (PFR-OA). First, for single-edge scenarios, we design an improved twin-delayed DRL-based algorithm to approximate the optimal policy (Section IV-A). Next, for multi-edge scenarios, we develop a new distributed training framework based on personalized FL (Section IV-B) to further enhance the model adaptiveness and training efficiency.

A. Improved Twin-Delayed DRL for Computation Offloading and Resource Allocation

P1 can be transformed into a classic problem of online budgeted maximum coverage, which has been proven to be NP-hard [39]. In this problem, the element e_i is selected at each step that contains costs and values, and thus all the selected elements during the whole process can be denoted as the set $E = \{e_1, e_2, \ldots, e_n\}$. The optimization objective of this problem is to find a set $E' \subseteq E$ that can maximize the total values while the total costs do not exceed the budget. For *P1*, we regard the task to be processed at each sub-slot as an element in E, the computational resources allocated to the task as costs, and the rewards of completing the task as values. The objective is to find a set $\{\alpha', \beta', w'\} \subseteq \{\alpha, \beta, w\}$ (i.e., an optimized policy of computation offloading and resource allocation) that can maximize the rewards without exceeding the constraints on available bandwidth and computational resource, which is the optimization objective of *P1*. Therefore, *P1* is an NP-hard problem. To solve this complicated problem, we model it as a Markov Decision Process (MDP) and propose a new DRL-based solution.

Specifically, we design an improved twin-delayed DRL-based algorithm to address *P1* for single-edge scenarios, aiming to minimize the delay and energy consumption of executing tasks. Based on an actor-critic framework, the classic twin-delayed DRL combines deep deterministic policy gradient and dual Q-learning, which performs well on many continuous-control problems. However, the classic twin-delayed DRL adopts a manner of local updating, which reveals the negative impact on global model convergence during distributed training. Moreover, the high frequency of error updates to the actor's network results in serious action dispersion. To address these issues, the proposed improved twin-delayed DRL-based algorithm lessens the unreasonable update frequency of the actor's network by introducing a new proximal term, which attenuates the dispersion of local updating and reduces the variance of action-value estimation, therefore generating better policies. As shown in Fig. 3, the DRL agent selects actions under different states by interacting with the single-edge environment and continuously optimizes the policies of computation offloading and resource allocation referring to the reward signals from the environment. For a DRL problem, the state space, action space, and reward function are defined as follows.

- *State space:* At sub-slot t, the system state is defined as

$$s_i(t) = \{D_i(t), C_i(t), b_i(t), l_i(t), B_i(t), F_i(t), q_i\}, \tag{17}$$

where $D_i(t) = \{D_{i,j}(t), j \in N\}$ indicates the set of data volumes for all tasks, $C_i(t) = \{C_{i,j}(t), j \in N\}$ indicates the set of required computational resources for all tasks, $b_i(t) = \{b_{i,j}(t), j \in N\}$ indicates the set of battery power for all EDs, $l_i(t) = \{l_{i,j}(t), j \in N\}$ indicates the set of distances between M_i and all EDs, $B_i(t)$ and $F_i(t)$ indicate the available bandwidth and computational resources of M_i, respectively. $q_i = \{q_{i,t}, q_{i,e}, q_{i,p}\}$ indicates the personalized demands for delay, energy consumption, and task success rate. Since different smart communities might own personalized demands on QoS and system overheads, various values of $q_{i,t}$, $q_{i,e}$, and $q_{i,p}$ are considered to simulate this distinction.

- *Action space:* At sub-slot t, the DRL agent makes an action of computation offloading and resource allocation based on the current system state, which is defined as

$$a_i(t) = \{\alpha_i(t), w_i(t), \beta_i(t)\}, \tag{18}$$

where $\alpha_i(t)$ indicates the set of offloading decisions for all tasks, and $w_i(t)$ and $\beta_i(t)$ indicate the sets of the proportion of bandwidth and computational resources allocated to all tasks, respectively. To satisfy $C1$, the continuous action $\alpha_i(t)$ output by the activation function *tanh* is rounded as the offloading decision. According to the constraints on $w_i(t)$ and $\beta_i(t)$ in $C4$ and $C5$, the remaining bandwidth and computational resources are allocated proportionally.

- *Reward function:* The optimization objective of *P1* is to minimize the weighted sum of delay and energy consumption. Thus, at sub-slot t, the instant reward of processing a task is defined as

$$r_{i,j}(t) = \begin{cases} -(q_{i,t}T_{i,j}(h) + q_{i,e}E_{i,j}(h)), & \text{Succeed} \\ -q_{i,p}, & \text{Unsatisfy } C2 \\ -q_{i,p}, & \text{Unsatisfy } C3 \end{cases} \tag{19}$$

If a task can be successfully completed, the instant reward will be the opposite of the weighted sum of delay and energy consumption. If $C2$ or $C3$ cannot be satisfied, the instant reward will be $q_{i,p}$, which is used as a penalty for failing to complete the task. Therefore, the long-term reward, which guides the training and update process, is defined as

$$r_i(t) = \sum_{j=1}^{N}\sum_{h=1}^{H}\gamma^t r_{i,j}(t), \tag{20}$$

where γ^t is the discount factor.

The main steps of the proposed improved twin-delayed DRL-based computation offloading and resource allocation algorithm are given in *Algorithm 1*. First, the actor's network μ_i and two critic's networks $Q_{i,1}$ and $Q_{i,2}$ are initialized, and the target actor's network μ_i' and two target critic's networks $Q_{i,1}'$ and $Q_{i,2}'$ are initialized accordingly (Line 1). To address the Q-value overestimation issue in classic actor-critic-based DRL, we introduce two critic's networks that separate action selection and Q-value update, aiming to improve the training stability. Next, we initialize the number of training epoch P, the number of time-slots H, the number of sub-slots T, the update frequencies of FL fp and the actor's network fa, the replay buffer G_i, the batch size N and the learning rate τ (Line 1).

Algorithm 1: Improved Twin-Delayed DRL for Computation Offloading and Resource Allocation.

1 **Initialize:** $\mu_i(s|\theta^{\mu})$, $Q_{i,1}(s,a|\theta^{Q_1})$, $Q'_{i,2}(s,a|\theta^{Q'_2})$, μ'_i, $Q'_{i,1}$, $Q'_{i,2}$, P, H, T, fp, G_i, N, and τ.
2 **for** $epoch = 0, 1, 2, ..., P$ **do**
3 **if** $epoch \ \% \ f_p == 0$ **then**
4 Upload $Q_{i,1}(s,a|\theta^{Q_{i,1}})$, $Q_{i,2}(s,a|\theta^{Q_{i,2}})$, and $\mu_i(s|\theta^{\mu_i})$ to the C-BS;
5 Call **Algorithm 2** to obtain $Q_{f,1}(s,a|\theta^{Q_{f,1}})$, $Q_{f,2}(s,a|\theta^{Q_{f,2}})$, and $\mu_f(s|\theta^{\mu_f})$;
6 Conduct soft update: $Q_{i,1}(s,a|\theta^{Q_{i,1}}) = f_q Q_{f1}(s,a|\theta^{Q_{i,1}}) + (1-f_q)Q_{i,1}(s,a|\theta^{Q_{i,1}})$, $Q_{i,2}(s,a|\theta^{Q_{i,2}}) = f_q Q_{f,2}(s,a|\theta^{Q_{f,2}}) + (1-f_q)Q_{i,2}(s,a|\theta^{Q_{i,2}})$, and $\mu_i(s|\theta^{\mu_i}) = f_q\mu_f(s|\theta^{\mu_f}) + (1-f_q)\mu_i(s|\theta^{\mu_i})$;
7 **end**
8 Receive the initial state $s_0 = env_i.reset()$;
9 **for** $t = 0, 1, 2, ..., H * T$ **do**
10 Select the action a_t, where $a_i(t) = \mu_i(s(t)|\theta^{\mu_i}) + \varepsilon_1, \varepsilon_1 \sim clip(N(0,\sigma_1), -c, c)$;
11 Execute $a_i(t)$ and receive $r_i(t)$ and the next state $s_i(t+1)$, where $r_i(t), s_i(t+1) = env_i.step(a_i(t))$;
12 Store the state-transition process in G_i: $G_i.push(s(t), a(t), r(t), s(t+1), done)$;
13 Randomly select N samples from G_i: $N * (s_t, a_t, r_t, s_{t+1}) = G_i.Sample(N)$;
14 Use target actor's network to get the next action by Eq. (21);
15 Calculate target Q-value $y_{t\arg et}$ by Eq. (22);
16 Update critic's network by Eq. (23);
17 **if** $epoch \ \% \ f_a == 0$ **then**
18 Update actor's network: $\nabla_{\theta^{\mu_i}} J(\theta^{\mu_i}) = (N^{-1}\sum_1^N \nabla_{a(t)} Q_{i,1}(s(t), a(t)|_{a=\mu_i\theta^{\mu_i}} \nabla_{\theta^{\mu_i}} \mu_{\theta^{\mu_i}}(s(t))) + \frac{\lambda_{\mu_i}}{2}||\theta^{\mu_i} - \theta^{\mu_f}||^2$;
19 Update target actor's and critic's networks: $\theta^{Q'_{i,z}} \leftarrow \rho\theta^{Q_{i,z}} + (1-\rho)\theta^{Q'_{i,z}}$, $\theta^{\mu'_i} \leftarrow \rho\theta^{\mu_i} + (1-\rho)\theta^{\mu'_i}$;
20 **end**
21 **end**
22 **end**

For each training epoch, when it comes to the round of FL update, $\mu_i(s|\theta^{\mu_i})$, $Q_{i,1}(s,a|\theta^{Q_{i,1}})$, and $Q_{i,2}(s,a|\theta^{Q_{i,2}})$ are uploaded to the C-BS (Line 4). Next, *Algorithm 2* is called to obtain aggregated models $Q_{f,1}(s,a|\theta^{Q_{f,1}})$, $Q_{f,2}(s,a|\theta^{Q_{f,2}})$, and $\mu_f(s|\theta^{\mu_f})$, which are used to replace local models by soft update (Lines 5∼6). For each sub-slot, the state $s_i(t)$ is first input to μ_i to obtain an action of computation offloading and resource allocation $a_i(t)$ and execute this action in the environment, which will feedback the instant reward and the next state $s_i(t+1)$ based on execution results (Lines 10∼11). Next, the state-transition process is stored in G_i, where N samples are randomly selected to train network parameters, and then

the target actor's network is used to get the next action (Lines 12∼14).

Specifically, the proposed algorithm uses the critic's network to fit $Q_i(s(t), a(t))$, which can accurately reflect the Q-values of each action. Meanwhile, we use the actor's network to fit the mapping between $s(t)$ and $a(t)$, and thus the DRL agent can take proper actions at different states and maximize the long-term reward. We introduce the Gaussian noise in the target actor's network to obtain $a(t+1)$, and this process is defined as

$$a_i(t+1) = \mu_i(s(t+1)|\theta^{\mu_i'}) + \varepsilon_2, \varepsilon_2 \sim clip(N(0, \sigma_2), -c, c), \tag{21}$$

where the actor's network is endowed with sufficient exploration of selecting target actions by adding noise.

Next, the target Q-value is calculated by considering the current reward and comparing two critic's networks (Line 15), which is defined as

$$y_{target} \leftarrow r(t) + \gamma \min_{z=1,2} Q_{i,z}(s(t+1), a(t+1)|\theta^{Q'_{i,z}}). \tag{22}$$

Next, the critic's network is updated by the loss back-propagated of the difference between y_{target} and the current Q-value (Line 16). Due to the variable demands on QoS and system overheads among different edge environments, we design a proximal term to replace the original loss function that only tends to minimize the difference in local Q-values, which speeds up the convergence of the FL-based training framework in *Algorithm 2* and reduces the negative impact of local updates on global model convergence during distributed training. The update process is defined as

$$\theta^{Q_{i,z}} \leftarrow \arg\min_{\theta_{i,z}} \left(N^{-1} \sum \left(y_{t\arg et} - Q_{i,z}(s(t), a(t)|\theta^{Q_{i,z}})\right)^2\right) + \frac{\lambda_{Q_{i,z}}}{2} ||\theta^{Q_{i,z}} - \theta^{Q_{f,z}}||^2. \tag{23}$$

Finally, to reduce the improper update of the actor's network, we design a soft updating mechanism, which makes the update frequency of the actor's network less than the critic's network and thus avoids the action dispersion caused by the high frequency of error updates (Lines 17∼20). With this design, the variance of action-value estimation can be effectively reduced, thus generating better policies of computation offloading and resource allocation.

B. Personalized FL-Based Training for DRL

In classic centralized training manners, all the information about EDs may need to be uploaded to a central server to train a DRL-based decision-making model of computation offloading and resource allocation. Such manners can achieve good model performance with rich training samples, but it is prone to severe network congestion and the potential risk of privacy leakage. As an emerging distributed training framework, the multi-agent DRL allows each agent to be trained independently, but the performance of local models will be seriously limited if some agents lack training samples. In contrast, the FRL can solve this issue by implementing a collaborative model training on data silos with the original purpose of privacy. However, different smart communities usually have personalized demands on QoS and system overheads. In this case, the classic FRL with the average aggregation of model parameters cannot make an effective response. To solve these issues, we design a new personalized FL-based training framework to further improve the adaptiveness and training efficiency of the DRL-based computation offloading and resource allocation model for different environments. The proposed personalized FRL-based training framework is illustrated in Fig. 4, whose main steps are given in *Algorithm 2*.

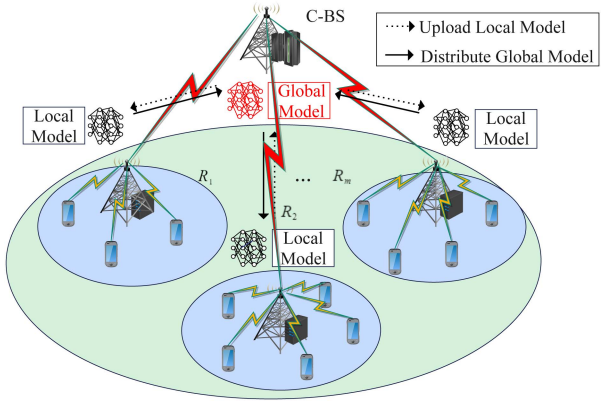


Fig. 4. The proposed personalized FRL-based training framework.

First, we initialize the federated actor's network μ_f, two federated critic's networks $Q_{f,1}$ and $Q_{f,2}$, the number of edges participating in FRL training V $(V \leq m)$, and the communication rounds for federated aggregation P_f (Line 1). In each communication round, we introduce a new proximal term to attenuate the dispersion of local updates, because different smart communities commonly own personalized demands on QoS and system overheads. Specifically, the proximal term is added to the local training loss by calling *Algorithm 1*, allowing faster convergence to the global optimum. The process is defined as

$$\theta_i^{t+1} \approx \min_{\theta_i^t} h_i^t(\theta_i^t; \theta_f^r) = L(\theta_i^t) + \frac{\lambda_{\theta_i}}{2} ||\theta_i^t - \theta_f^r||^2, \tag{24}$$

where $\lambda_{\theta_i} = \{\lambda_{Q_{i,1}}, \lambda_{Q_{i,2}}, \lambda_{\mu_i}\}$ is the parameter set of the proximal term. We use an adaptive tuning of λ_{θ_i}, indicating the loss changes of local models, to solve the issue of model heterogeneity. This design limits the iterative trajectories of local agents to avoid the training dispersion caused by the deviation of local models. Specifically, when conducting federated aggregation, the training loss of improper local decisions might mislead global updates and cause global deviation. In this case, the parameters of the proximal term will be adjusted to be higher during local updates to weaken the negative impact of local divergence. Moreover, when the training loss of proper local decisions decreases, the parameters of the proximal term will be adjusted to be lower to avoid the excessive impact of the global model on the aggregation of local models. Besides, it is an average federated aggregation when the parameters of the proximal term are adjusted to 0.

Algorithm 2: The Proposed Personalized FL-Based Training for DRL.

1 **Initialize:** μ_f, $Q_{f,1}$, $Q_{f,2}$, V, and P_f.
2 **for** $r = 1, 2, ..., P_f$ **do**
3 **for** $i = 1, 2, ..., m$ **do**
4 Call **Algorithm 1** to obtain $\mu_i(s|\theta^{\mu_i})$, $Q_{i,1}(s, a|\theta^{Q_{i,1}})$, and $Q_{i,2}(s, a|\theta^{Q_{i,2}})$ and upload them to the C-BS;
5 **end**
6 Sort DRL models in R by ascending order of training loss: $R = env.sort(R)$;
7 **for** $v = 1, 2, ..., V$ **do**
8 **if** $random.uniform(0, 1) < \varepsilon$ **then**
9 Select one of the first V models from the sorted R to Z_r;
10 **else**
11 Randomly select a model from R to Z_r;
12 **end**
13 **end**
14 Conduct federated aggregation of V DRL models in Z_r according to Eq. (25);
15 Distribute the aggregated models to $R_i (R_i \in R)$;
16 **end**

In each communication round, each DRL model in $R_i (R_i \in R)$ uploads the actor's network $\mu_i(s|\theta^{\mu_i})$ and two critic's networks $Q_{i,1}(s, a|\theta^{Q_{i,1}})$ and $Q_{i,2}(s, a|\theta^{Q_{i,2}})$ to the C-BS (Lines 3∼5). Next, we sort DRL models by ascending order of their training loss (Line 6), and then we design a partial-greedy based participant selection mechanism, where the C-BS uses ε-greedy to select some DRL models for federated aggregation (Lines 7∼14). Specifically, to avoid the high complexity caused by aggregating all local DRL models, we introduce a partial-greedy mechanism, which selects V models with the lowest training loss by the probability of ε and adds them to Z_r, thereby accelerating the convergence of the global model. Meanwhile, this mechanism randomly selects models by the probability of 1-ε and also adds them to Z_r, which enables exploring a broader unknown solution space. Therefore, compared to the classic FRL that aggregates all local DRL models, the proposed partial-greedy mechanism can reduce the training complexity while giving the model sufficient exploration ability. The process of federated aggregation is defined as

$$\begin{aligned} Q_{f,1}(s, a|\theta^{Q_{f,1}}) &= \frac{1}{V} \sum\nolimits_{i \in Z_r} Q_{i,1}(s, a|\theta^{Q_{i,1}}), \\ Q_{f,2}(s, a|\theta^{Q_{f,2}}) &= \frac{1}{V} \sum\nolimits_{i \in Z_r} Q_{i,2}(s, a|\theta^{Q_{i,2}}), \\ \mu_f(s|\theta^{\mu_f}) &= \frac{1}{V} \sum\nolimits_{i \in Z_r} \mu_i(s|\theta^{\mu_i}), \end{aligned} \tag{25}$$

where Z_r indicates the set of local DRL models that participate in federated aggregation.

Finally, the C-BS distributes the aggregated DRL models to the DRL agents in each R_i $(R_i \in R)$ (Line 15) and waits for the next communication round of FRL training.

C. Complexity Analysis of PFR-OA

At the sub-slot t, the sizes of state and action spaces in a DRL agent are $|s_i(t)|$ and $|a_i(t)|$, respectively, and all m DRL agents own the same network structure. Therefore, the following two parts should be considered when calculating the complexity of the proposed PFR-OA.

- *Reward calculation:* The complexity of calculating rewards for all DRL agents is $Y_t^R = O(m \cdot |s_i(t)|)$.
- *Action selection:* The numbers of layers and neurons in the actor's network and two critic's networks are considered when calculating the complexity of selecting actions. For the actor's network, the number of neurons in the layer i is denoted as U_i^A, and the number of layers is denoted as M^A. Therefore, the complexity of the layer i is $O(U_{i-1}^A U_i^A + U_i^A U_{i+1}^A)$, and the complexity of the actor's networks for all DRL agents is $Y_t^A = O(m \cdot (|s_i(t)| \cdot U_2^A + \sum_{i=3}^{M^A-2} (U_{i-1}^A U_i^A + U_i^A U_{i+1}^A) + U_{M^A-1}^A \cdot |a_t|))$. Similarly, the number of neurons in the layer j of the critic's network is denoted as U_j^C, and the number of layers is denoted as M^C. Thus, the complexity of the layer j of the critic's network is $O(U_{j-1}^C U_j^C + U_j^C U_{j+1}^C)$. Since there are two critic's networks with the same structure, the complexity of the critic's networks for all DRL agents is $Y_t^C = O(2\,m \cdot (|s(t)| \cdot U_2^C + \sum_{j=3}^{M^C-2} (U_{j-1}^C U_j^C + U_j^C U_{j+1}^C) + U_{M^C-1}^C))$. With the above considerations, the complexity of selecting actions for all DRL agents is $Y_t^{AC} = T_t^A + T_t^C$.

By combining the above two parts, the complexity of a DRL agent is $Y_t = Y_t^R + Y_t^{AC}$ at the sub-slot t, and thus the complexity of completely training a DRL agent is $O(H \cdot T \cdot Y_t)$. Considering the FRL training process, the complexity of the proposed PFR-OA is $O(P_f \cdot H \cdot T \cdot Y_t)$.

V. Performance Evaluation

In this section, we evaluate the proposed PFR-OA through extensive comparison experiments on both simulation and testbed environments.

A. Experiment Setup

1) Datasets and Parameter Settings: We refer to the architecture of the real-world edge computing platform (i.e., C-ESP) and adopt its running datasets [40]. The C-ESP builds an edge computing platform covering almost all regions in China, which hosts different types of service providers and records running datasets. Specifically, the datasets come from 2359 edge servers deployed in more than 1,000 locations and 96,209 users with 10,159,851 requests. From the datasets, we can obtain the fuzzy geographic locations of request senders (i.e., users) and receivers (i.e., edge servers) with unique identifiers and the generation time of requests. It is noted that the geographic locations are fuzzy based on IP addresses to protect user privacy. The simulation experiments are conducted on a workstation with an 8-core Intel(R) Xeon(R) Sliver 4208 CPU @ 3.2 GHz, two NVIDIA GeForce RTX 3090 GPUs, and 32 GB of RAM. Based on PyTorch, we implement the proposed system model and

PFR-OA. Specifically, the system model is built with three MEC servers, where each MEC server owns computing capability $F_i(t)$ of 20 GHz. Meanwhile, each MEC server is equipped with a BS with bandwidth $B_i(t)$ of 15 MHz, where the EDs within the communication coverage are connected to it via the wireless network. The tasks of EDs are generated with various demands for dynamic multi-edge communities based on the running datasets of C-ESP. To simulate personalized multi-edge environments, we adopt different values of $q_{i,t}$, $q_{i,e}$, and $q_{i,p}$ in various edge environments, and there exist distinct task attributes and service demands in each edge environment. Moreover, a complete training epoch contains 20 time-slots and each time-slot contains 4 sub-slots. As for the other parameters in the proposed model, we set $T_d = 1.5$ s, $D_{i,j}(h) \in [0.5, 1.5]$ MB, $C_{i,j}(h) \in [1, 1.2]$ GHz, $P_{i,j}(t) \in [100, 400]$ W, $f_{i,j} \in [1, 1.2]$ GHz, $P_{i,j}(t) \in [0.1, 0.4]$ MB/s, $P_m = 100$ W, $b_{\max} = 120$ J, $e_t = 40$ J, $k = \mathrm{e}^{-26}$, $\sigma = \mathrm{e}^{-3}$, $q_{i,t} \in [0.5, 1]$, $q_{i,e} \in [0.5, 1]$, and $q_{i,p} \in [10, 20]$, respectively. The task attributes including $D_{i,j}(h)$ and $C_{i,j}(h)$ follow the Poisson process within fixed intervals with random values. As for the parameters in the proposed PFR-OA, we set $\gamma = 0.995$, $\tau = 0.0001$, $f_p = 50$, $f_q = 0.3$, $N = 256$, $\rho = 0.1$, and $f_a = 2$, respectively.

2) Performance Indications: To comprehensively evaluate the proposed PFR-OA, we use the following performance indicators.

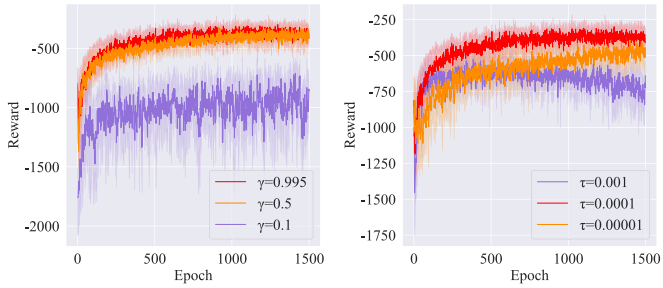
- Reward: Sum of instant rewards.
- Task success rate: Rate of completed tasks.
- Average energy consumption: Average energy consumption of all tasks, considering failed tasks.
- Average waiting time: Average waiting time of all tasks, considering failed tasks.

3) Benchmark Methods: To verify the superiority of the proposed PFR-OA, we compare it with the following benchmark methods.

- *MCF-TD3* [23]: The model is trained by the average federated aggregation of multiple TD3-based DRL agents.
- *TD3* [41]: The model is trained by the TD3 algorithm without multi-edge collaboration.
- *DDPG* [21]: The model is trained by the DDPG algorithm without multi-edge collaboration.
- *DQN* [26]: The model is trained by the DQN algorithm without multi-edge collaboration, where the continuous action space is discretized.
- *Greedy:* Tasks tend to be executed on EDs if the maximum tolerable delay can be satisfied.
- *Edge:* All tasks are offloaded to the MEC server for execution, where resources are evenly distributed.
- *Local:* All tasks are executed on EDs.

B. Experiment Results and Analysis

1) Hyperparameter Tuning: We analyze the impact of different hyperparameters (including the reward discount factor γ and learning rate τ) on the performance of the proposed PFR-OA. As shown in Fig. 5(a), when γ is small (e.g., 0.1), the PFR-OA focuses on the instant reward and thus adopts a short-sighted policy. However, it is commonly expected that the current decision can make a good balance between the instant and future rewards. As γ grows, the long-term rewards are fully considered in the current decision, thus improving model performance. It should be noted that $\gamma = 1$ indicates that the infinitely far future rewards will be considered in the current decision, which is unreasonable in practice. As shown in Fig. 5(b), the proposed PFR-OA can quickly converge to a steady status with different values of τ. As τ increases, there occurs a more significant influence of new environments on policies, accompanied by more pronounced model oscillations. Conversely, as τ decreases, the model becomes less exploratory, exhibiting slower convergence, and may not converge to the optimum. Based on the above hyperparameter tuning and analysis, the PFR-OA achieves excellent performance with $\gamma = 0.995$ and $\tau = 0.0001$, and thus we adopt this setting in the subsequent experiments.



(a) Discount factor γ. (b) Learning rate τ.

Fig. 5. Performance of PFR-OA with different hyperparameters.

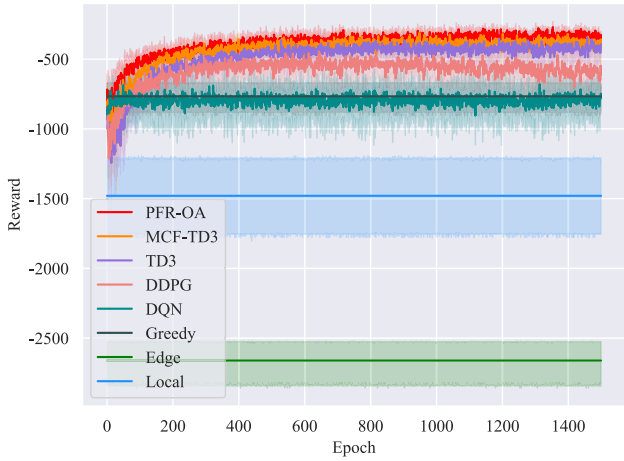
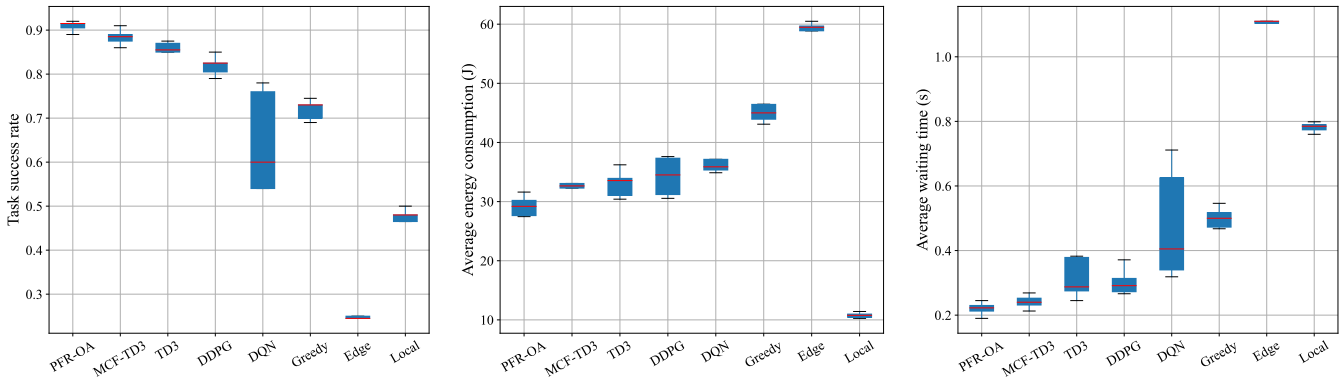


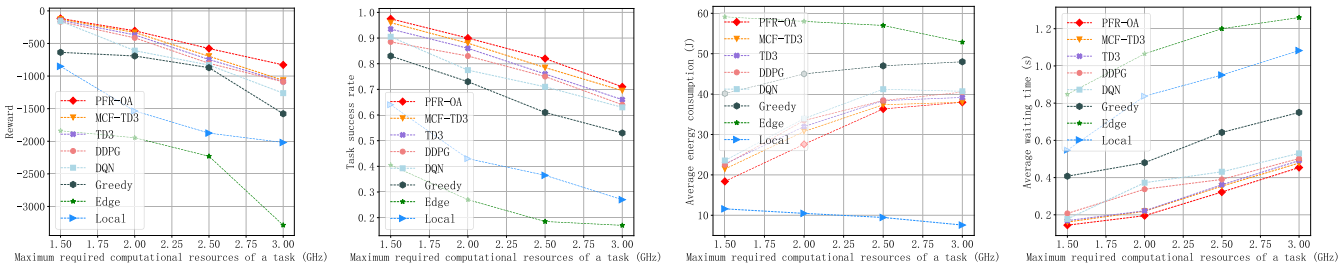
Fig. 6. Convergence comparison among different methods.

2) Convergence Comparison: As shown in Fig. 6, we compare the convergence of the different methods. The *Greedy*, *Edge*, and *Local* are single-step decision-making methods, and thus there is no learning and optimization process. Compared to the DRL-based methods, the *Greedy*, *Edge*, and *Local* perform worse. This is because their policies of computation offloading and resource allocation are relatively blind without fully considering system states and task characteristics, resulting in massive failed tasks due to violating the constraints on the maximum



(a) Task success rate. (b) Average energy consumption. (c) Average waiting time.

Fig. 7. Comparison of task success rate, average energy consumption, and average waiting time among different methods.



(a) Reward. (b) Task success rate. (c) Average energy consumption. (d) Average waiting time.

Fig. 8. Performance comparison among different methods with various required computational resources of tasks.

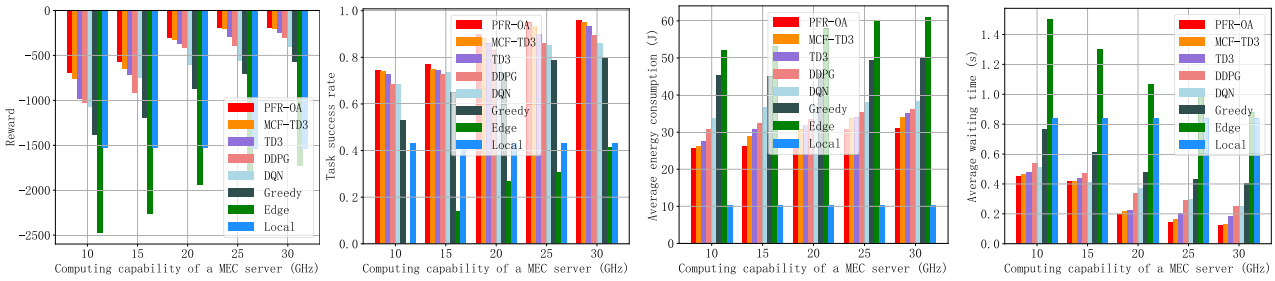
tolerable delay or battery power. Compared to the advanced *MCF-TD3* that performs best among other DRL-based methods, the proposed PFR-OA converges to better performance faster. This is because the PFR-OA improves the loss function in the classic TD3 by designing a new proximal term while considering personalized demands in multi-edge smart communities. Specifically, as shown in Fig. 7(a), we compare the task success rates of different methods after convergence. The PFR-OA can converge to better performance and complete more tasks than other methods under delay and power constraints. Moreover, Fig. 7(b) and (c) illustrate the average energy consumption and average waiting time of different methods after convergence. The *Local* consumes the least energy by executing all tasks on EDs because the power of EDs is much lower than MEC servers. However, limited by the computing capabilities of EDs, more tasks may fail due to exceeding the maximum tolerable delay, also resulting in longer average waiting time. The *Edge* offloads all tasks to MEC servers for execution, leading to excessive waiting time and energy consumption. Compared to other advanced DRL-based methods, the proposed PFR-OA achieves better performance in terms of task success rate, average energy consumption, and average waiting time with the improved twin-delayed DRL and well-designed personalized FL-based framework.

3) Performance Comparison With Various Required Computational Resources of Tasks: As illustrated in Fig. 8(a), we evaluate the overall performance of different methods with various required computational resources of tasks in terms of reward. When tasks require a smaller amount of computational resources, they consume less delay and energy while more tasks can be completed under constraints, leading to higher rewards. As this variable increases, more tasks may fail because they cannot satisfy the delay and energy constraints, causing decreased rewards. Specifically, as depicted in Fig. 8(b), the proposed PFR-OA obtains higher task success rates than the advanced *MCF-TD3* and *TD3* by introducing a personalized FL-based framework. Compared to the *DQN*, the PFR-OA can better handle the continuous problem of resource allocation and make proper policies according to diverse task characteristics, ensuring more completed tasks. Fig. 8(c) and (d) compare different methods from the perspectives of average energy consumption and average waiting time. The *Local* executes all tasks on EDs, resulting in lower energy consumption. However, due to the limited computational capabilities of EDs, a lower task success rate happens with the increasing difficulty of executing tasks. The *Edge* offloads all tasks to MEC servers for execution, leading to higher energy consumption and transmission delay. It is worth noting that the proposed PFR-OA displays better performance than other advanced DRL-based methods regarding task success rate, average energy consumption, and average waiting time.

4) Performance Comparison With Various Network Bandwidths of a BS: We test the influence of various network bandwidths on the performance of different methods. As shown in

TABLE III
PERFORMANCE COMPARISON OF DIFFERENT METHODS WITH VARIOUS NETWORK BANDWIDTHS OF A BS

Scenario	Method	Reward	Task success rate	Average energy consumption (J)	Average waiting time (s)
$B_i(t)$=5 MHz	**PFR-OA**	**-446.23**	**0.87**	31.47	**0.26**
	MCF-TD3	-477.34	0.85	34.00	0.30
	TD3	-489.75	0.83	32.51	0.32
	DDPG	-522.52	0.81	35.20	0.36
	DQN	-675.71	0.77	39.15	0.39
	Greedy	-940.16	0.65	45.56	0.60
	Edge	-2918.37	0.19	59.12	1.21
	Local	-1535.33	0.43	**10.47**	0.84
$B_i(t)$=15 MHz	**PFR-OA**	**-305.82**	**0.90**	27.58	**0.19**
	MCF-TD3	-331.53	0.88	30.74	0.22
	TD3	-369.24	0.86	31.96	0.22
	DDPG	-414.15	0.83	33.45	0.34
	DQN	-610.26	0.78	37.93	0.37
	Greedy	-871.12	0.73	45.00	0.48
	Edge	-1946.38	0.27	58.36	1.07
	Local	-1535.35	0.43	**10.47**	0.84
$B_i(t)$=25 MHz	**PFR-OA**	**-285.69**	**0.92**	26.62	**0.19**
	MCF-TD3	-312.83	0.89	29.35	0.20
	TD3	-333.45	0.89	29.00	0.20
	DDPG	-374.31	0.87	30.81	0.31
	DQN	-533.62	0.83	36.63	0.35
	Greedy	-680.04	0.78	42.04	0.43
	Edge	-1645.64	0.38	55.24	0.90
	Local	-1535.32	0.43	**10.47**	0.84

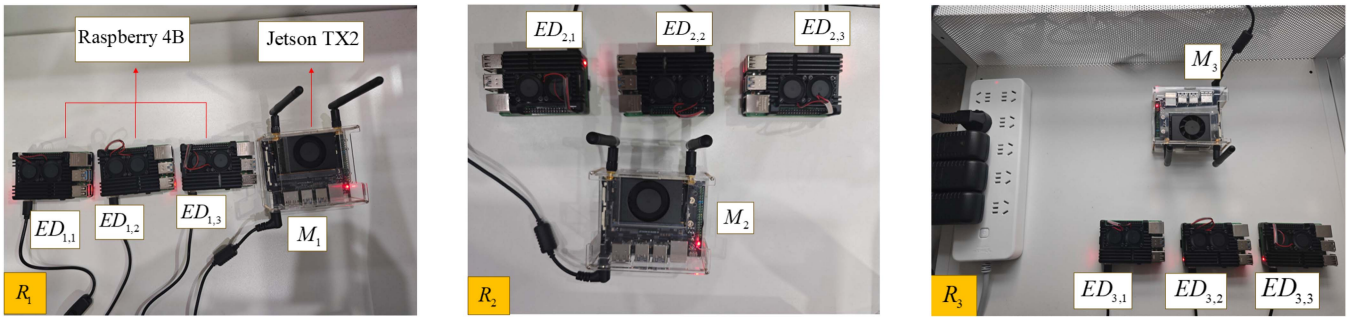


(a) Reward. (b) Task success rate. (c) Average energy consumption. (d) Average waiting time.

Fig. 9. Performance comparison among different methods with various computing capabilities of a MEC server.

Table III, the changes in network bandwidths do not affect the performance of the *Local* because it does not contain the offloading process. As the network bandwidth increases, the performance of the *Edge* enhances most significantly. This is because the *Edge* offloads all tasks to MEC servers for execution, and thus the increasing network bandwidths can reduce the task transmission delay, also considerably improving reward and task success rate. With the increase in network bandwidths, the performance of different methods tends to stabilize. This is because fewer tasks fail due to exceeding the delay constraint during the offloading process. Since the energy consumption of computation offloading is much higher than the local execution, EDs may not support too many tasks for offloading under the constraint of battery power. In this case, the performance cannot be further improved. It is noted that the proposed PFR-OA outperforms other advanced DRL-based methods regarding different performance indicators, verifying its superiority in handling the complex issue of computation offloading and resource allocation in dynamic multi-edge environments.

5) Performance Comparison With Various Computing Capabilities of a MEC Server: We evaluate the performance of different methods with various computing capabilities of MEC servers. As depicted in Fig. 9(a), there is no offloading process in the *Local*, and thus the increase of computing capabilities of MEC servers does not affect its performance. With the rise of this variable, both the reward and task success rate of all methods exhibit a growing trend. Compared to other advanced DRL-based methods, the proposed PFR-OA performs better because it can effectively ameliorate the negative impact of local updates on model convergence in classic distributed training. As shown in Fig. 9(b), the performance of all methods (except the *Local*) tends to be stable as the computing capabilities of MEC servers rise because fewer offloaded tasks fail due to exceeding the delay constraint. However, EDs are constrained by battery power and cannot support offloading too many tasks, and thus the performance of these methods cannot be further improved. As shown in Fig. 9(c) and (d), more tasks can be offloaded to MEC servers for execution as their computing capabilities



(a) Edge environment of R_1. (b) Edge environment of R_2. (c) Edge environment of R_3.

Fig. 10. Construction of the real-world testbed with hardware devices.

expand. Therefore, the average energy consumption of all methods (except the *Local*) reveals an increasing tendency. Meanwhile, by offloading tasks to MEC servers, the task execution delay can be lessened, and thus the average waiting time exhibits a decreasing tendency. Among all these methods, the proposed PFR-OA can always keep more excellent overall performance.

C. Real-World Testbed Validation

To further verify the practicality and superiority of the proposed PFR-OA, we construct a real-world testbed with hardware devices to evaluate the performance. As illustrated in Fig. 10, the testbed consists of three groups of devices that are located in the lab, where each group contains an MEC server and three EDs. Specifically, each ED is acted by a Raspberry 4B that is equipped with a Broadcom BCM2711 SoC @ 1.5 GHz, 4 GB of RAM, and a Raspbian GNU/Linux 11 OS, and each MEC server is acted by a Jetson TX2 that is equipped with a 4-core Arm CortexA57 MP-Core processor, a 256-core NVIDIA Pascal GPU, 8 GB of RAM, and Ubuntu 18.04.6 LTS. All these devices are connected to a 5 GHz router where the communication platform is built based on the Flask framework. In the testbed environment, each MEC server owns comparable bandwidth to the simulation environment but its computing capability is different. We adopt image classification as a service instance for computation offloading, where EDs generate tasks of image classification with varying data volumes and resource demands at different time-slots and send offloading requests. If the requests are accepted, the tasks will be uploaded to the corresponding MEC servers for execution. Otherwise, the tasks will be executed locally. Since the data transmission delay might be affected by unstable channels in real-world environments, the errors between the theoretical and actual values of the data transmission time are taken into account. Moreover, we consider the diversity of task attributes and service demands in edge environments of R_1, R_2, and R_3 to simulate personalized multi-edge environments.

As illustrated in Fig. 11, we compare the performance of the different methods on the real-world testbed. It should be noted that because the hardware devices and tasks in testbed experiments differ from the parameter settings in simulation experiments, the reward values of different methods are not in the same range as simulation results. The *Local* executes all tasks

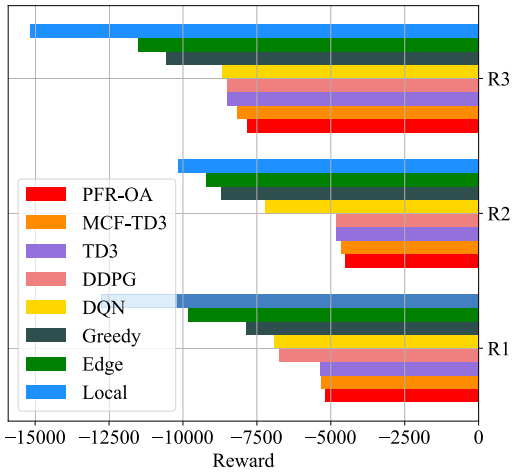


Fig. 11. Performance comparison of different methods on the testbed.

locally, which cannot handle the cases of task failing when the task execution time exceeds the maximum tolerable delay. The *Edge* offloads all tasks to MEC servers for execution, leading to the under-performance of some offloaded tasks due to the insufficient allocation of computational resources. The advanced DRL-based methods can make appropriate offloading decisions based on system states and task attributes. Therefore, the DRL-based methods achieve higher rewards than other heuristics under different edge environments. Among all the DRL-based methods, the proposed PFR-OA reaches the best performance. This is because the PFR-OA considers the demand diversity in different edge environments and improves the efficiency of edge cooperative training through a new personalized FL-based framework, avoiding performance degradation due to local training dispersion. The above results verify the effectiveness of the proposed PFR-OA in real-world scenarios.

VI. Conclusion

In this article, we first formulate the computation offloading and resource allocation in dynamic multi-edge smart community systems with personalized demands as a model-free DRL problem with multiple constraints. Next, we propose a novel PFR-OA that combines an improved twin-delayed DRL-based

algorithm and a new personalized FL-based training framework to address the issues of action dispersion and inefficient model updates. Using real-world system settings and testbed, extensive experiments demonstrate the effectiveness of the proposed PFR-OA. Compared to the other seven benchmark methods (i.e., *MCF-TD3*, *TD3*, *DDPG*, *DQN*, *Greedy*, *Edge*, and *Local*), the PFR-OA shows superiority in improving the task success rate, average energy consumption, and average waiting time. Specifically, the PFR-OA outperforms other benchmark methods in different scenarios with various required computational resources of tasks, network bandwidths of BSs, and computing capabilities of MEC servers. Notably, we validate the practicality of the PFR-OA on the real-world testbed. When facing heterogeneous devices and diverse demands in different edge environments, the PFR-OA is able to maintain the best performance among all methods. In our future work, we will extend this work to the scenario of space-air-ground networks and explore the feasibility of the proposed personalized FDRL framework for improving load balancing among heterogeneous devices.

REFERENCES

[1] S. H. Alsamhi, F. A. Almalki, O. Ma, M. S. Ansari, and B. Lee, "Predictive estimation of optimal signal strength from drones over IoT frameworks in smart cities," *IEEE Trans. Mobile Comput.*, vol. 22, no. 1, pp. 402–416, Jan. 2023.

[2] C. Cabrera and S. Clarke, "A self-adaptive service discovery model for smart cities," *IEEE Trans. Services Comput.*, vol. 15, no. 1, pp. 386–399, Jan./Feb. 2022.

[3] M. Goudarzi, M. Palaniswami, and R. Buyya, "A distributed deep reinforcement learning technique for application placement in edge and fog computing environments," *IEEE Trans. Mobile Comput.*, vol. 22, no. 5, pp. 2491–2505, May 2023.

[4] Z. Chen, J. Hu, G. Min, C. Luo, and T. El-Ghazawi, "Adaptive and efficient resource allocation in cloud datacenters using actor-critic deep reinforcement learning," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 8, pp. 1911–1923, Aug. 2022.

[5] H. Jin, P. Zhang, H. Dong, X. Wei, Y. Zhu, and T. Gu, "Mobility-aware and privacy-protecting QoS optimization in mobile edge networks," *IEEE Trans. Mobile Comput.*, vol. 23, no. 2, pp. 1169–1185, Feb. 2024.

[6] X. Wang et al., "Wireless powered mobile edge computing networks: A survey," *ACM Comput. Surv.*, vol. 55, 2023, Art. no. 263.

[7] Z. Chen and Z. Yu, "Intelligent offloading in blockchain-based mobile crowdsensing using deep reinforcement learning," *IEEE Commun. Mag.*, vol. 61, no. 6, pp. 118–123, Jun. 2023.

[8] H. Jiang, X. Dai, Z. Xiao, and A. Iyengar, "Joint task offloading and resource allocation for energy-constrained mobile edge computing," *IEEE Trans. Mobile Comput.*, vol. 22, no. 7, pp. 4000–4015, Jul. 2023.

[9] L. Ma, X. Wang, X. Wang, L. Wang, Y. Shi, and M. Huang, "TCDA: Truthful combinatorial double auctions for mobile edge computing in industrial Internet of Things," *IEEE Trans. Mobile Comput.*, vol. 21, no. 11, pp. 4125–4138, Nov. 2022.

[10] Y. Chen, N. Zhang, Y. Zhang, X. Chen, W. Wu, and X. S. Shen, "TOFFEE: Task offloading and frequency scaling for energy efficiency of mobile devices in mobile edge computing," *IEEE Trans. Cloud Comput.*, vol. 9, no. 4, pp. 1634–1644, Fourth Quarter 2021.

[11] Z. Ning et al., "5G-enabled UAV-to-community offloading: Joint trajectory design and task scheduling," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 11, pp. 3306–3320, Nov. 2021.

[12] Z. Ding, D. Xu, R. Schober, and H. V. Poor, "Hybrid NOMA offloading in multi-user MEC networks," *IEEE Trans. Wireless Commun.*, vol. 21, no. 7, pp. 5377–5391, Jul. 2022.

[13] M. Ma, J. Zhang, and P. Wang, "DePo: Dynamically offload expensive event processing to the edge of cyber-physical systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 9, pp. 2120–2132, Sep. 2022.

[14] H. Tran-Dang and D.-S. Kim, "FRATO: Fog resource based adaptive task offloading for delay-minimizing IoT service provisioning," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 10, pp. 2491–2508, Oct. 2021.

[15] Z. Song, Y. Liu, and X. Sun, "Joint task offloading and resource allocation for NOMA-enabled multi-access mobile edge computing," *IEEE Trans. Commun.*, vol. 69, no. 3, pp. 1548–1564, Mar. 2021.

[16] A. Feriani and E. Hossain, "Single and multi-agent deep reinforcement learning for AI-enabled wireless networks: A tutorial," *IEEE Commun. Surveys Tuts.*, vol. 23, no. 2, pp. 1226–1252, Second Quarter 2021.

[17] Y. Wu, T. Q. Dinh, Y. Fu, C. Lin, and T. Q. S. Quek, "A hybrid DQN and optimization approach for strategy and resource allocation in MEC networks," *IEEE Trans. Wireless Commun.*, vol. 20, no. 7, pp. 4282–4295, Jul. 2021.

[18] S. Bi, L. Huang, H. Wang, and Y.-J. A. Zhang, "Lyapunov-guided deep reinforcement learning for stable online computation offloading in mobile-edge computing networks," *IEEE Trans. Wireless Commun.*, vol. 20, no. 11, pp. 7519–7537, Nov. 2021.

[19] M. Tang and V. W. Wong, "Deep reinforcement learning for task offloading in mobile edge computing systems," *IEEE Trans. Mobile Comput.*, vol. 21, no. 6, pp. 1985–1997, Jun. 2022.

[20] W. Fan et al., "DNN deployment, task offloading, and resource allocation for joint task inference in IIoT," *IEEE Trans. Ind. Inform.*, vol. 19, no. 2, pp. 1634–1646, Feb. 2023.

[21] W. Jiang, D. Feng, Y. Sun, G. Feng, Z. Wang, and X. Xia, "Joint computation offloading and resource allocation for D2D-assisted mobile edge computing," *IEEE Trans. Services Comput.*, vol. 16, no. 3, pp. 1949–1963, May/Jun. 2023.

[22] C. Liu, F. Tang, Y. Hu, K. Li, Z. Tang, and K. Li, "Distributed task migration optimization in MEC by extending multi-agent deep reinforcement learning approach," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 7, pp. 1603–1614, Jul. 2021.

[23] T. Zhao, F. Li, and L. He, "Secure video offloading in MEC-enabled IIoT networks: A multi-cell federated deep reinforcement learning approach," *IEEE Trans. Ind. Inform.*, vol. 20, no. 2, pp. 1618–1629, Feb. 2024.

[24] S. Xia, Z. Yao, Y. Li, and S. Mao, "Online distributed offloading and computing resource management with energy harvesting for heterogeneous MEC-enabled IoT," *IEEE Trans. Wireless Commun.*, vol. 20, no. 10, pp. 6743–6757, Oct. 2021.

[25] Y. Mao, W. Hong, H. Wang, Q. Li, and S. Zhong, "Privacy-preserving computation offloading for parallel deep neural networks training," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 7, pp. 1777–1788, Jul. 2021.

[26] L. Hsieh, H. Liu, Y. Guo, and R. Gazda, "Deep reinforcement learning-based task assignment for cooperative mobile edge computing," *IEEE Trans. Mobile Comput.*, vol. 23, no. 4, pp. 3156–3171, Apr. 2024.

[27] S. Zhang et al., "Blockchain and federated deep reinforcement learning based secure cloud-edge-end collaboration in power IoT," *IEEE Wireless Commun.*, vol. 29, no. 2, pp. 84–91, Apr. 2022.

[28] T. M. Ho and K. Nguyen, "Joint server selection, cooperative offloading and handover in multi-access edge computing wireless network: A deep reinforcement learning approach," *IEEE Trans. Mobile Comput.*, vol. 21, no. 7, pp. 2421–2435, Jul. 2022.

[29] Z. Hu et al., "An efficient online computation offloading approach for large-scale mobile edge computing via deep reinforcement learning," *IEEE Trans. Services Comput.*, vol. 15, no. 2, pp. 669–683, Mar./Apr. 2022.

[30] X. Liu, J. Yu, J. Wang, and Y. Gao, "Resource allocation with edge computing in IoT networks via machine learning," *IEEE Internet Things J.*, vol. 7, no. 4, pp. 3415–3426, Apr. 2020.

[31] L. Yuan et al., "CSEdge: Enabling collaborative edge storage for multi-access edge computing based on blockchain," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 8, pp. 1873–1887, Aug. 2022.

[32] X. Deng, J. Li, L. Shi, Z. Wei, X. Zhou, and J. Yuan, "Wireless powered mobile edge computing: Dynamic resource allocation and throughput maximization," *IEEE Trans. Mobile Comput.*, vol. 21, no. 6, pp. 2271–2288, Jun. 2022.

[33] Z. Gao, L. Yang, and Y. Dai, "Large-scale computation offloading using a multi-agent reinforcement learning in heterogeneous multi-access edge computing," *IEEE Trans. Mobile Comput.*, vol. 22, no. 6, pp. 3425–3443, Jun. 2023.

[34] X. Dai et al., "Task co-offloading for D2D-assisted mobile edge computing in industrial Internet of Things," *IEEE Trans. Ind. Inform.*, vol. 19, no. 1, pp. 480–490, Jan. 2023.

[35] D. C. Nguyen, M. Ding, P. N. Pathirana, A. Seneviratne, J. Li, and H. V. Poor, "Cooperative task offloading and block mining in blockchain-based edge computing with multi-agent deep reinforcement learning," *IEEE Trans. Mobile Comput.*, vol. 22, no. 4, pp. 2021–2037, Apr. 2023.

[36] Q. Zhang, H. Wen, Y. Liu, S. Chang, and Z. Han, "Federated-reinforcement-learning-enabled joint communication, sensing, and computing resources allocation in connected automated vehicles networks," *IEEE Internet Things J.*, vol. 9, no. 22, pp. 23224–23240, Nov. 2022.

[37] T. Zhu, W. Zhou, D. Ye, Z. Cheng, and J. Li, "Resource allocation in IoT edge computing via concurrent federated reinforcement learning," *IEEE Internet Things J.*, vol. 9, no. 2, pp. 1414–1426, Jan. 2022.
[38] T. Zhao, F. Li, and L. He, "DRL-based joint resource allocation and device orchestration for hierarchical federated learning in noma-enabled industrial IoT," *IEEE Trans. Ind. Inform.*, vol. 19, no. 6, pp. 7468–7479, Jun. 2023.
[39] K. Tammer, "The application of parametric optimization and imbedding to the foundation and realization of a generalized primal decomposition approach," *Math. Res.*, vol. 35, pp. 376–386, 1987.
[40] S. Shen et al., "A holistic QoS view of crowdsourced edge cloud platform," in *Proc. IEEE/ACM 31st Int. Symp. Qual. Serv.*, 2023, pp. 01–10.
[41] B. Hazarika, K. Singh, S. Biswas, and C. Li, "DRL-based resource allocation for computation offloading in IoV networks," *IEEE Trans. Ind. Inform.*, vol. 18, no. 11, pp. 8027–8038, Nov. 2022.

Zheyi Chen (Member, IEEE) received the MSc degree in computer science and technology from Tsinghua University, China, in 2017, and the PhD degree in computer science from the University of Exeter, U.K., in 2021. He is a professor and Qishan scholar with the College of Computer and Data Science, Fuzhou University, China. His research interests include cloud-edge computing, resource optimization, deep learning, and reinforcement learning. He has published more than 30 research papers in reputable international journals and conferences such as the *IEEE Transactions on Parallel and Distributed Systems*, IEEE INFOCOM, *IEEE Transactions on Industrial Informatics*, *IEEE Communications Magazine*, *IEEE Transactions on Cloud Computing*, *IEEE Internet of Things Journal*, and IEEE ICC.

Bing Xiong received the BS degree in computer science and technology from Fujian Normal University, China, in 2021. He is currently working toward the MS degree in computer science with the College of Computer and Data Science, Fuzhou University. His current research interests include cloud/edge computing and deep reinforcement learning.

Xing Chen (Member, IEEE) received the BS and PhD degrees from Peking University, in 2008 and 2013, respectively. Upon completion of the PhD degree, he joined Fuzhou University and has held the rank of professor since 2020. Now, he is the deputy director with the Fujian Provincial Key Laboratory of Network Computing and Intelligent Information Processing (Fuzhou University), and leads the Systems Research Group. His research focuses on the software systems and engineering approaches for cloud and mobility. His current projects cover the topics from self-adaptive software, computation offloading, and model driven approach. He has published more than 50 journal and conference articles, and was awarded three First Class Prizes for Provincial Scientific and Technological Progress, in 2018, 2020, and 2021.

Geyong Min (Member, IEEE) received the BSc degree in computer science from the Huazhong University of Science and Technology, China, in 1995, and the PhD degree in computing science from the University of Glasgow, U.K., in 2003. He is a professor of high performance computing and networking with the Department of Computer Science, Faculty of Environment, Science and Economy, University of Exeter, U.K. His research interests include future Internet, computer networks, wireless communications, multimedia systems, information security, high-performance computing, ubiquitous computing, modelling, and performance engineering.

Jie Li (Fellow, IEEE) received the BE degree in computer science from Zhejiang University, Hangzhou, China, the ME degree in electronic engineering and communication systems from the China Academy of Posts and Telecommunications, Beijing, China, and the DrEng degree from the University of Electro-Communications, Tokyo, Japan. He is with the Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China where he is a chair professor. His current research interests include Big Data and AI, blockchain, edge computing, networking and security, OS, information system architecture. He serves as the director of Shanghai Jiao Tong University Blockchain Research Centre. He was a professor with the Department of Computer Science, University of Tsukuba, Japan. He was a visiting professor with Yale University, USA, Inria Sophia Antipolis and Inria Grenoble-Rhone-Aples, France. He is the co-chair of IEEE Technical Community on Big Data and the founding chair of IEEE ComSoc Technical Committee on Big Data. He serves as an associated editor for many IEEE journals and transactions. He has also served on the program committees for several international conferences.