

Computation Offloading with Multiple Agents in Edge Computing-supported IoT

SHIHAO SHEN[†], Tianjin University, China
 YIWEN HAN[†], Tianjin University, China
 XIAOFEI WANG^{*†}, Tianjin University, China
 YAN WANG[‡], Harbin Engineering University, China

With the development of the Internet of Things (IoT) and the birth of various new IoT devices, the capacity of massive IoT devices is facing challenges. Fortunately, edge computing can optimize problems such as delay and connectivity by offloading part of the computational tasks to edge nodes close to the data source. Using this feature, IoT devices can save more resources while still maintaining the quality of service. However, since computation offloading decisions concern joint and complex resource management, we use multiple Deep Reinforcement Learning (DRL) agents deployed on IoT devices to guide their own decisions. Besides, Federated Learning (FL) is utilized to train DRL agents in a distributed fashion, aiming to make the DRL-based decision making practical and further decrease the transmission cost between IoT devices and Edge Nodes. In this paper, we first study the problem of computation offloading optimization and prove the problem is an NP-hard problem. Then, based on DRL and FL, we propose an offloading algorithm that is different from the traditional method. Finally, we studied the effects of various parameters on the performance of the algorithm and verified the effectiveness of both the DRL and FL in the IoT system.

CCS Concepts: • **Networks** → **Mobile networks**; • **Human-centered computing** → **Mobile computing**; • **Computing methodologies** → *Cooperation and coordination*.

Additional Key Words and Phrases: Federated learning, computation offloading, IoT, edge computing

ACM Reference Format:

Shihao Shen, Yiwen Han, Xiaofei Wang, and Yan Wang. 2019. Computation Offloading with Multiple Agents in Edge Computing-supported IoT. *ACM Trans. Sensor Netw.* X, XXX, Article 1 (September 2019), 28 pages. <https://doi.org/xx.xxxx/xxxxxxx.xxxxxxx>

*The corresponding author.

[†]Tianjin Key Laboratory of Advanced Networking (TANK), College of Intelligence and Computing, Tianjin University.

[‡]The National Key Laboratory of Underwater Acoustic Science and Technology, College of Underwater Acoustic Engineering, Harbin Engineering University.

This work is supported by the National Key R&D Program (2018YFC0809803, 2019YFB2101901) of China, China NSFC (Youth) through grant 61702364, Huawei Innovation Research Program (HO2018095224) and the Opening Fund of Acoustics Science and Technology Laboratory (Grant No.SSKF2018002). This paper is an extension of our conference paper published in the ACM TURC (DOI: 10.1145/3321408.3321586). The authors extend the previous work by improving the expression of parameters, adding complexity analysis to the problem, discussing the theoretical analysis of the proposed algorithm and adding multi-angle fine-grained related experiments. The authors would like to thank the anonymous referees for their valuable comments and helpful suggestions.

Authors' addresses: Shihao Shen, Tianjin University, Tianjin, China, shenshihao@tju.edu.cn; Yiwen Han, Tianjin University, Tianjin, China, hanyiwen@tju.edu.cn; Xiaofei Wang, Tianjin University, Tianjin, China, xiaofeiwang@tju.edu.cn; Yan Wang, Harbin Engineering University, Harbin, China, wangyan@hrbeu.edu.cn.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2019 Association for Computing Machinery.

1550-4859/2019/9-ART1

<https://doi.org/xx.xxxx/xxxxxxx.xxxxxxx>

1 INTRODUCTION

The Internet of Things(IoT) is an extension of the Internet. It can connect things to the network for communication by using sensors to realize intelligent management, positioning, identification, monitoring, and other functions. At present, IoT is rapidly emerging and has driven the vigorous development of various related application services. It has been applied in many fields such as smart home [19], human health detection [16], disaster management [21], building structure safety [15], person identification and [22] and so on.

As shown in Fig. 1, IoT has been closely integrated with various fields. However, various application services need to deploy a large number of IoT devices while providing rich functions, which puts heavy pressure on communication. Besides, various new IoT devices such as smart home devices and wearable devices are also emerging and they have extremely stringent requirements on the bandwidth, delay, and privacy of the network [29, 31], which pose challenges to the quality of the communication.

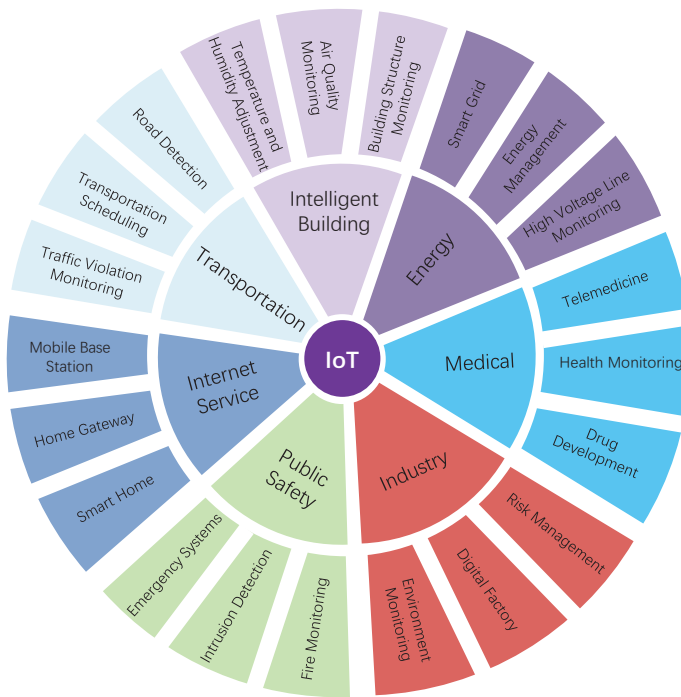


Fig. 1. The applications of IoT.

Catering for the increasing service requirement, massive IoT devices will be deployed in a pervasive fashion to carry out tasks like monitoring, sensing data collection and preprocessing and immediate decision-making. The above tasks usually require a large number of computing resources, while the ability of IoT devices is relatively weak to support. However, edge computing can offload tasks and is expected to solve this problem [25, 33]. In detail, edge computing can offload computing tasks submitted by IoT devices to similar edge nodes to provide rich computing resources. Besides, the edge node in edge computing systems is adopted as a coordinator among them and responsible for their communications and even load-balance [37].

In solving the resource allocation problem in task offloading, in addition to using convex optimization [6] and game theory [7], Deep Reinforcement Learning (DRL) is used in [8] to handle the comprehensive resource allocation in computation offloading. This approach can maximize the long-term benefits of energy consumption and execution latency and requires no prior knowledge of network statics and partial information. In detail, this kind of optimization has many advantages. First, the IoT device does not need to obtain global information, which is conducive to communication transmission and privacy protection. Secondly, it has the adaptability to the dynamic environment. Finally, this kind of optimization not only optimizes the system within a time segment but will consider long-term benefits. However, an assumption was made in [8]. They assume that IoT devices are computationally powerful enough to train their own DRL agents independently. However, IoT devices will not grow so powerful in the near future, and their computing resources can support lightweight neural networks at most.

At present, due to people pay more attention to data security and privacy, the protection of data and privacy has become an important issue that must be considered [38]. For example, the General Data Protection Regulation [28] implemented in the European Union is aimed at data security and privacy, and it gives users the right to delete or withdraw personal data. Therefore, the traditional way of transferring data to a data center for centralized analysis will face a privacy barrier in the future. In summary, how to protect data security and privacy while using large amounts of data will become an important challenge in the future.

Thus, we propose a distributed training scheme based on Federated Learning (FL) [5, 26] to alleviate the training burden on each device. Unlike the traditional distributed training in the data center with an excellent networking environment, this training is restrained by wireless communication and networking and shall be performed in an efficient manner of communication. In this vein, observation data sensed by each IoT device are not required to be transmitted frequently between it and the edge node. Observation data on a specific IoT device are used for local training, and only updated parameters of the DRL agent are uploaded to the edge node for further model aggregation.

Therefore, in this work, we use FL to conduct the training process of DRL agents for jointly allocating communication and computation resources. Specifically, our main contributions lie in three aspects.

- First, we studied a problem of computation offloading optimization in edge computing-supported IoT. On one hand, the computational tasks can be performed locally, while some energy needs to be allocated to the task processing components in the IoT device. On the other hand, it can also be performed by transmitting tasks to the edge nodes, while requiring some energy to be allocated to the data transmission components in the IoT device. Compared to local execution, this can be done with the richer computing resources of the edge nodes, but it also causes additional transmission delays due to data transmission. Moreover, we further analyzed the complexity of the problem and proved that the problem is NP-hard.
- Second, we designed an algorithm to make decisions about computation offloading and energy allocation, seeking to maximize the expected long-term utility. This algorithm can be trained based on federated learning, so data collected by each IoT device only needs to be stored locally for analysis. This approach avoids a lot of data transmission and achieves good data privacy protection.
- Finally, we simulate to evaluate the proposed algorithm and study the influence of various system parameters. Experimental results corroborate its effectiveness comparing to the centralized training method.

2 BACKGROUND

In this section, we introduce the background of this study. Since the technology that this research relies on mainly involves deep reinforcement learning, federated learning and edge computing, the principles and related research of these technologies are briefly introduced.

2.1 Deep Reinforcement Learning

Since reinforcement learning (RL) techniques are usually applied to small data spaces, it is difficult to perform data processing through RL when data with high dimensions. However, Deep reinforcement learning (DRL) solves this problem by combining the high-dimensional input of deep learning with RL.

RL usually attempts to make action decisions a according to the environment state s , and the action proceeds to the environment to obtain the action reward r , and continuously adjusts and improves according to r [35]. While deep learning is a method of characterizing and learning data through a multi-layer neural network and learning the characteristic information of the data through the neural network. DRL combines deep learning and RL which not only retains the perception of deep learning but also can make decisions for RL, so it has better performance.

DRL has been well applied in many fields such as natural language processing [32], image recognition [30], and so on. Among them, AlphaGo [34], which defeated the human professional chess player in go and showed excellent decision-making ability. Besides, some researchers use DRL to play Atari. They train the model by using the joystick moving direction as the action space of DRL and using the score in the game as a reward. In the game, DRL surpassed the traditional method in six of the games, even in the performance of three games beyond the human level. Besides, various DRL libraries such as TensorFlow [12], Caffe [18] and Keras [4] are also emerging, which facilitates the application of DRL.

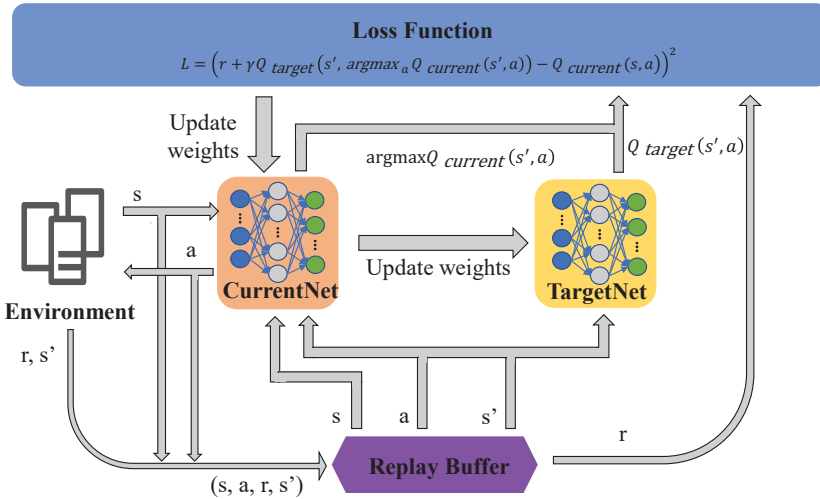


Fig. 2. Diagram of DDQN.

As shown in Fig. 2, Double Deep Q-Learning (DDQN) [36] is an excellent DRL algorithm. To solve the problem of the curse of dimensionality in reinforcement learning, DDQN uses a neural network to approximate some states that have not appeared before:

$$Q(s, a) \approx f(s, a, w) . \quad (1)$$

The training process of the neural network is an optimization problem, so the loss function is defined as:

$$L = (r + \gamma Q_{\text{target}}(s', \arg \max_a Q_{\text{current}}(s', a)) - Q_{\text{current}}(s, a))^2. \tag{2}$$

In addition, since the training of the neural network is supervised, the training data must satisfy the independent and identical distribution, otherwise, the network will be trapped in the local minimum. Therefore, a replay buffer \mathcal{B} is constructed to store the data sample $D_t = (s_t, a_t, r_t, s_{t+1})$ at each time step t , and randomly extract a mini-batch of samples during training.

2.2 Federated Learning

The traditional large-scale neural network training needs to concentrate data in one device, which puts great challenges on traffic load and data privacy. In this regard, Google has proposed Federated learning [13]. It allows multiple end devices to train on local data, and then only needs to upload updates to the cloud.

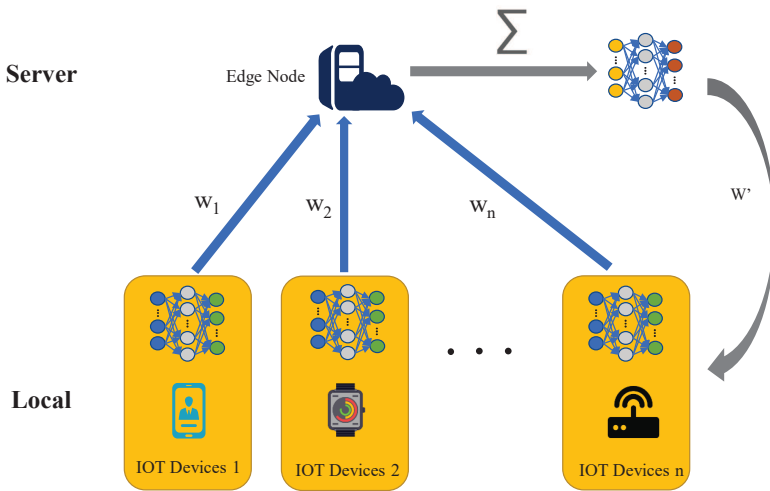


Fig. 3. The training process of federated learning.

The way of federated learning works is shown in Fig. 3. First, the end devices download the sharing model from the cloud, and then train the model according to the local data and transfer the update to the cloud with encrypted transmissions. Finally, the cloud integrates the sharing model according to the update from multiple end devices. Since user data is always stored locally on the end device throughout the process, large amounts of data can be avoided from being transmitted to the cloud, thereby reducing the pressure on data transmission and protecting data privacy.

In the practical application of federated learning, there are still some problems. On the one hand, sample data will be distributed in a large number of end devices in an extremely uneven manner. On the other hand, the transmission speed of end devices is slow, especially data uploading speed can limit overall performance. To solve these problems, Google developed an algorithm federated averaging to reduce the network requirements when training deep neural networks [26], and also compress updates by using random rotation and quantization to reduce the amount of data transferred [24]. In addition, a federate optimization algorithm was designed to optimize the high-dimensional sparse convex model [23].

2.3 Edge Computing

Edge computing provides network, computing, application, and storage services to close users through distributed edge nodes. Therefore, tasks can be performed on EN to avoid sending data to the cloud. In 2014, the European Telecommunications Standards Institute standardized the concept of edge computing [9], which also marked the standardization of edge computing technology.

The end devices in edge computing are diverse, such as connected vehicle [14], smart cameras [3], etc. and they are producers of data and tasks with data pre-processing and data transmission functions [1]. However, when an end device needs to handle a task with a very large computational resource, it is often difficult to rely on the computing capability of the device itself to meet the demand. Therefore, it can be solved by edge computing using the computing resources of the edge node. The edge node is geographically close to the end device and can provide high-quality network connection and computing services. Compared with the end device, the edge node has a more powerful computing capacity to process the task, and the edge node responds faster to the end device than the cloud. Therefore, by using edge nodes to perform some computational tasks, the response speed of the task can be improved while ensuring accuracy. In addition, the edge node also has a caching function [39], which can shorten the response time of re-access by caching objects with high access heat.

3 SYSTEM MODEL

Next, we will introduce the system model used in this study. First, the overall architecture of the edge computing-supported IoT system is presented and the relevant parameters involved in the system are introduced. After that, we introduce the changing ways of the related parameters and show the derivation process of the parameters.

3.1 Overview

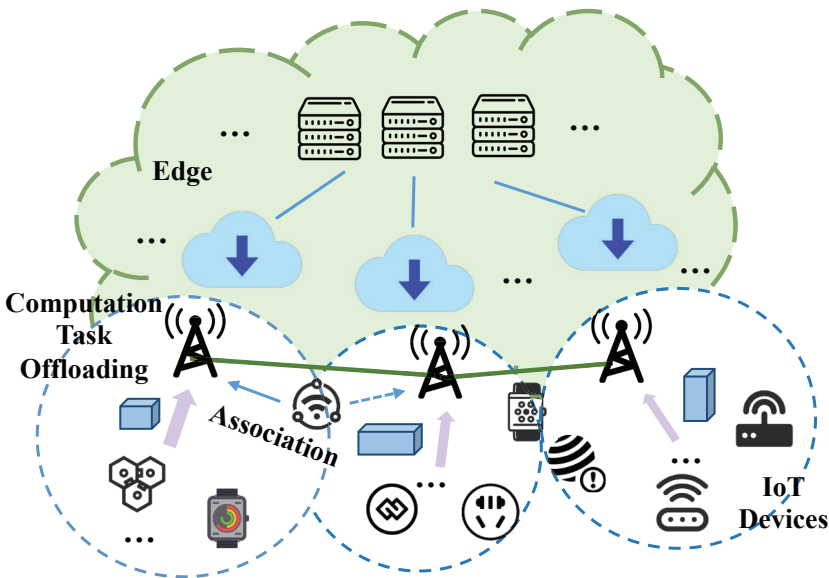


Fig. 4. Edge computing-supported IoT system.

As shown in Fig. 4, IoT devices, denoted as $\mathcal{D} = \{1, \dots, D\}$, are located in the service area of a set $\mathcal{N} = \{1, \dots, N\}$ of Edge Nodes (ENs). It is worth noting that the concept of time slices is used to divide the time into several epochs of δ (in seconds) indexed by i . The functions of transmission and calculation are supported by EN and the geographical location, task processing capability and data transmission capability of each EN are different. For IoT devices, there will be a task queue with a maximum length of q_{\max}^t to temporarily store tasks and these tasks will be executed in the order of first-in-first-out (FIFO). The IoT device will generate tasks according to the Bernoulli distribution and define a_i^t as the task arrival indicator. $a_i^t = 1$ indicates that the epoch i has a task generation, otherwise it indicates that no task is generated. When the task queue has reached the upper limit, the task queue will not store the newly generated task, which will cause the task to directly fail. In addition, the collection method of energy units is similar to that discussed in [2] and the IoT devices can collect energy units from the outside. In modeling, the IoT device has an energy queue with a maximum length of q_{\max}^e to store energy units, which will acquire the energy unit according to the Poisson distribution.

The calculation task generated by IoT devices is modeled as (μ, ν) , where μ (in bits) and ν respectively represent the transmission data size required for offloading a task and the needed number of CPU cycles for processing the task. Moreover, there are two ways to accomplish these calculation tasks, one is performed locally on the IoT device, and the other is offloading to an EN with channel bandwidth W Hz to perform. However, as shown in Fig. 5, the delay caused by data transmission can be avoided if the task is executed locally. For another, if the task is executed at EN, the task can be performed with the richer computing resources of EN. Both methods have their advantages and disadvantages and they need to be weighed according to the specific state.

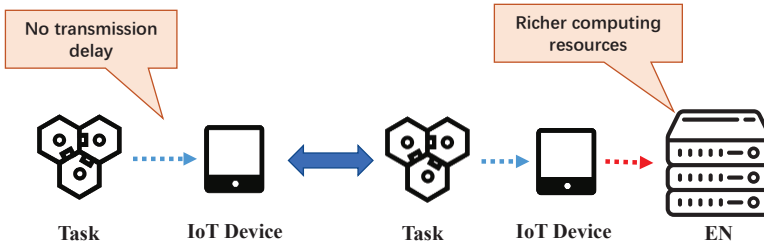


Fig. 5. Task execution method

3.2 Description of System Model

3.2.1 System model architecture. Since the execution mode of the task in the system model includes local execution and offloading to EN, the dynamic running process of the system is as shown in Fig. 6.

The IoT device needs to make a joint action (c_i, e_i) during the epoch i , where c_i represents the task offloading decision, and its specific definition is

$$c_i = \begin{cases} 0, & \text{if local execution,} \\ n \in \mathcal{N}, & \text{if offload to EN } n. \end{cases} \quad (3)$$

Besides, e_i represents the number of energy units allocated and affects the CPU frequency and data transmission rate of IoT devices. Moreover, e_i cannot exceed the number of energy units in the energy queue, and if $e_i = 0$, the task will not be executed and will still be saved in the task

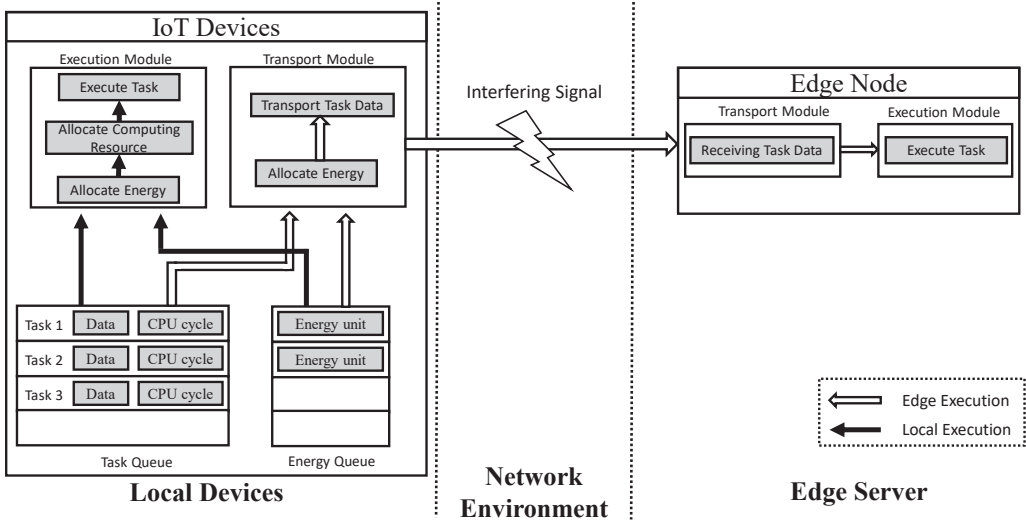


Fig. 6. Schematic diagram of the dynamic system model.

queue, and only $e_i > 0$, the task will be executed. In addition, IoT devices also have a task queue that will not be able to save newly created tasks when the task queue is full.

3.2.2 Local execution. When the task is executed locally, the time consumption satisfies the following constraints

$$d_i^m = e_i / p_i^{exe}, \quad (4)$$

where p_i^{exe} is the power of the IoT device to perform a task. Meanwhile, as in [11], p_i^{exe} can be written as

$$p_i^{exe} = \tau \cdot f_i^\zeta, \quad (5)$$

where τ is a constant that depends on the average switched capacitance and ζ is the average activity factor which usually close to 3. Besides, the time consumption also satisfies the following constraints

$$d_i^m = v / f_i. \quad (6)$$

From the above, we can get the time consumption d_i^m for local execution by solving equation (4), (5) and (6), which can be written as

$$d_i^m = \left(\frac{v^\zeta \cdot \tau}{e_i} \right)^{\frac{1}{\zeta-1}}. \quad (7)$$

3.2.3 Executed at EN. If the IoT device decides to offload the task to EN, then the relevant data of the task needs to be transferred to EN. Therefore, an association between the IoT device and the EN is required. If the required association is different from the previous, a handover will occur and cause additional delays. Let s_i denote the association between IoT device and EN at the beginning of epoch i , which can be written as

$$s_i = \begin{cases} s_{i-1}, & \text{if executed locally in the previous epoch } (c_{i-1} = 0), \\ c_{i-1}, & \text{if offload to EN in the previous epoch } (c_{i-1} \neq 0). \end{cases} \quad (8)$$

Moreover, the handover delay resulting from altering EN association should be considered. We assume that there will be σ seconds delay when handoff occurs, so the handover delay h_i can be represented as

$$h_i = \begin{cases} 0, & \text{if no altering EN association } (c_i = s_i), \\ \sigma, & \text{if altering EN association } (c_i \neq s_i). \end{cases} \quad (9)$$

In addition, it is necessary to model the rate of transfer between the IoT device and EN when data transfer occurs. We denote g_u^n as the channel gain of the IoT device u between the IoT device and an EN $n \in \mathcal{N}$, which is assumed static and independently taken from a finite state space \mathcal{G}_n . Let L_c denote all sets of IoT devices that use the same channel as the target IoT device u_t . When a radio link is established for them, the achievable data rate can be calculated as

$$r_i = W \cdot \log_2 \left(1 + \frac{g_{u_t}^n \cdot p_{u_t}^{\text{tr}}}{\sum_{u \in L_c} g_u^n \cdot p_u^{\text{tr}} - g_{u_t}^n \cdot p_{u_t}^{\text{tr}}} \right), \quad (10)$$

where p_u^{tr} represent the transmit power of the IoT device u and p_u^{tr} satisfies

$$p_u^{\text{tr}} = e_i / d_i^{\text{tr}} \leq p_{\max}^{\text{tr}}. \quad (11)$$

Thus, the time consumption on data transmission can be expressed as

$$d_i^{\text{tr}} = \mu / r_i. \quad (12)$$

According to the proof in [8], given the association $s_i \in \mathcal{N}$ and the allocated energy units $e_i > 0$ at a epoch i , the transmitting rate should remain a constant for achieving the minimum transmission time, which is preferred in practical. Therefore, the minimum transmission time can be solved by equations of (10), (11) and (12) as

$$\log_2 \left(1 + \frac{g_i^{c_i} \cdot e_i}{d_i^{\text{tr}} \cdot \sum_{u \in L_c} g_u^n \cdot p_u^{\text{tr}} - g_{u_t}^n \cdot p_{u_t}^{\text{tr}}} \right) = \frac{\mu}{W \cdot d_i^{\text{tr}}}. \quad (13)$$

After the task is offloaded to an EN, the task will be completed by this edge node. While the execution delay d^s of a task in EN is much less than the transmission delay d_i^{tr} , so d^s is set to a small constant. In addition, the payment ϕ_i of occupying the EN is set to avoid excessive use of EN resources in actual operation. With defining $\pi \in \mathbb{R}_+$ as the price per unit of time, the payment expression can be written as

$$\phi_i = \pi \cdot (\min \{h_i + d_i^{\text{tr}} + d^s, \delta\} - h_i). \quad (14)$$

3.2.4 Update system model parameters. Through the above modeling of different treatments of tasks, the task execution delay can be summarized as follows:

$$d_i = \begin{cases} d_i^{\text{m}}, & \text{if local execution } (e_i > 0 \text{ and } c_i = 0), \\ h_i + d_i^{\text{tr}} + d^s, & \text{if offload to EN to execute } (e_i > 0 \text{ and } c_i \in \mathcal{N}), \\ 0, & \text{if not executed } (e_i = 0). \end{cases} \quad (15)$$

In addition, it is also considered that not all tasks can be executed immediately after they are generated, so let ρ_i denote the queuing delay in the task queue at epoch i , which can be described as

$$\rho_i = q_i^{\text{t}} - 1_{\{d_i > 0\}}. \quad (16)$$

In each epoch, we need to pay attention to the dynamic changes of the task queue and energy queue of the IoT device. In the details, the change in the length q_i^e of the energy queue can be described as

$$q_{i+1}^e = \min \{q_i^e - e_i + a_i^e, q_{\max}^e\}, \quad (17)$$

where $a_i^e \in \mathbb{N}_+$ is the number of energy units acquired by the IoT device in epoch i . For task queues, we need to consider the generation and completion of tasks for each epoch. Similarly, let a_i^t denote the number of tasks generated in epoch i . Then dynamics of the task queue length can be calculated as

$$q_{i+1}^t = \min \{q_i^t - \mathbf{1}_{\{0 < d_i \leq \delta\}} + a_i^t, q_{\max}^t\}. \quad (18)$$

However, newly generated tasks cannot be stored and fail directly when the task queue is full. So let η_i denote the number of computation task drop in an epoch i , which can be described as

$$\eta_i = \max \{q_i^t - \mathbf{1}_{\{0 < d_i \leq \delta\}} + a_i^t - q_{\max}^t, 0\}. \quad (19)$$

In order to express this model more clearly, the values and definitions of the parameters are shown in table 1.

4 POLICY TRAINING COORDINATED BY FEDERATED LEARNING

In this section, we first explain the optimization problem to be solved and formulate the problem. In addition, we further analyze the complexity of the problem and prove that the optimization problem is NP-hard. After that, we analyze the merits of federated learning in edge computing and propose an algorithm of federated learning-based policy training. Finally, we carry out a theoretical analysis of the proposed algorithm.

4.1 Problem Formulation

Based on the system model described in Sec. 3.1 and Sec. 3.2, we will discuss the optimization problem next. First we define \mathcal{X}_i to represent the network environment of the IoT device during the epoch i .

$$\begin{aligned} \mathcal{X}_i &= (q_i^t, q_i^e, s_i, \mathbf{g}_i) \in \mathcal{X} \\ &\stackrel{\text{def}}{=} \{0, 1, \dots, q_{\max}^t\} \times \{0, 1, \dots, q_{\max}^e\} \times \mathcal{N} \times \{\times_{n \in \mathcal{N}} \mathcal{G}_n\}, \end{aligned} \quad (20)$$

where $\mathbf{g}_i = (g_i^n : n \in \mathcal{N})$. The IoT device will make the offloading decision and determine the number of energy units allocated during the initial period of epoch i , i.e.,

$$(c_i, e_i) \in \mathcal{J} \stackrel{\text{def}}{=} \{\{0\} \cup \mathcal{N}\} \times \{0, 1, \dots, q_{\max}^e\}. \quad (21)$$

The policy for making the above actions is defined as Φ and the expected long-term utility is defined as

$$U(\mathcal{X}, \Phi) = \mathbb{E}_{\Phi} \left[\lim_{I \rightarrow \infty} \frac{1}{I} \cdot \sum_{i=1}^I u(\mathcal{X}_i, \Phi(\mathcal{X}_i)) \mid \mathcal{X}_1 = \mathcal{X} \right], \quad (22)$$

where \mathcal{X}_1 represents the initial network environment and $u(\cdot)$ represents the short-term utility in the epoch i , which is determined by the task execution delay d_i , the number of task drop η_i , the task queuing delay ρ_i and the payment ϕ_i . Besides, it is worth mentioning that this optimization policy can be personalized according to the target. For instance, if low delay is the most important indicator in a system, the weighting of the task execution delay d_i and task queuing delay ρ_i can be adjusted to change the proportion of delay in the whole utility.

Table 1. The values and definitions of main parameters

Parameters	Definitions	Values
\mathcal{D}	The set of IoT devices.	/
\mathcal{N}	The set of Edge Nodes.	/
W	Channel bandwidth.	6.0×10^5 Hz
δ	Duration of each epoch.	5.0×10^{-3} s
a_i^t	Task arrival indicator.	/
μ	The transmission data size required for offloading a task.	3.0×10^4 bit
ν	The needed number of CPU cycles for processing a task.	8.375×10^6 cycle
c_i	Task offload decision.	/
e_i	The number of energy units allocated.	/
τ	A constant about the average switched capacitance.	1.0×10^{-28}
f_i	Allocated CPU frequency of the IoT device.	/
f_{\max}^c	The maximum CPU frequency of the IoT device.	2.0×10^9 Hz
d_i^m	Time consumption for the local task execution.	/
r_i	The achievable data rate of IoT device.	/
P_{\max}^{tr}	The maximum transmit power.	2W
p_i^{tr}	The maximum transmit power at epoch i .	/
d_i^{tr}	The time for transmitting task data.	/
q_{i+1}^e	The energy queue length insider the IoT device at epoch i .	/
d^s	The delay of server-side execution.	1.0×10^{-6} s
q_{\max}^t	The maximum length of the local task queue.	4
q_{\max}^e	The maximum length of the local energy queue.	4
d_i^{tr}	The time for transmitting task data.	/
σ	Delay of one handover.	2.0×10^{-3} s
h_i	The handover delay resulting from altering EN association.	/
q_i^t	Task queue length at epoch i .	/
η_i	The number of computation task drop at epoch i .	/
ρ_i	Queuing delay during the epoch i .	/
φ_i	The penalty of a computation task fails.	/
ϕ_i	The payment of occupying the EN.	/

4.2 Complexity Analysis

Here, we can first consider a special case of this problem. Suppose that at epoch t , there are $N'(q_{\max}^t \geq N' > 0)$ tasks in the task queue and $M'(q_{\max}^e \geq M' > 0)$ energy units in the energy queue. Besides, no energy unit and task are generated after epoch t . In addition, each task is executed on the local device and executed at the maximum CPU frequency f_{\max}^c . Therefore, the energy unit e'_k required to perform task k and the utility u'_k obtained by completing task k will be two certain values. Furthermore, $d'_k \in \{0, 1\}$ is defined as the task execution indicator, that is, $d'_k = 0$ means that the task k is not executed, and $d'_k = 1$ means that the task k is executed. In this case, the problem is changed to:

$$\max \sum_{k \in N'} d'_k \cdot u'_k, \quad (23)$$

subject to

$$\sum_{k \in N'} d'_k \cdot e'_k \leq M'. \quad (24)$$

For this situation, we can regard the energy units M' as the capacity of a knapsack, N' tasks as the items, and the energy unit e'_k required for each task and the utility u'_k obtained by completing a task are regarded as the weight and value of items respectively. Then the special case can be regarded as 0/1 knapsack problem. Since [20] has proven that the 0/1 knapsack problem is NP-hard. Therefore, the special case is also NP-hard. According to [10], since the problem in the special case is NP-hard, the problem is also NP-hard in non-special case.

4.3 Reasons for Using Federated Learning in Edge Computing

In the above, we have introduced the computation offloading problem. Fortunately, DRL can deal with this kind of problem well, thus we use Double Deep Q Learning (DDQN) [27, 36] to maximize the long-term utility. In addition, we also use the FL in our policy, and then we will explain why FL is used.

Although DRL can make decisions efficiently, it consumes a lot of computing resources. Therefore, how to provide the computing resources needs to be considered. On one hand, if the DRL agent is trained on the EN, it will bring about three disadvantages: 1) it will cause a large amount of data to be transmitted between the IoT device and EN, thereby increasing the transmission pressure of the wireless channel; 2) the transmitted data may contain private information, which is not conducive to the privacy protection of the data; 3) although the privacy information in the data can be removed in some ways, this will destroy the integrity of the data and affect the training effect.

On the other hand, if the DRL agent is trained on the IoT device individually, two deficiencies remain: 1) this will take too long to train each DRL agent from the beginning; 2) if each DRL agent will be trained independently, it will cause more energy consumption.

Therefore, the DRL will be trained in a distributed manner as shown in Fig. 7. However, due to network limitations and the challenge of protecting data privacy, most distributed deep learning technologies [26] are not feasible. For the above reasons, FL is introduced into the proposed policy for distributed training DRL agents.

4.4 Federated Learning-based DRL Training about Computation Offloading

Because each IoT device needs to make decisions based on its own network environment in computation offloading and it will also have an impact on the network environment. Therefore, we need EN to coordinate various IoT devices to optimize the overall network environment.

As shown in Algorithm 1, some IoT devices will be randomly selected during each iteration to perform the following operations: 1) download DRL agent parameters from EN and load them; 2) training DRL agent with the data obtained by itself; 3) upload update parameters to EN for model aggregation. In the envisioned IoT system, the IoT device needs to train the DRL agent based on the local data acquired by itself, including the number of unfinished tasks, the remaining energy units, and their own network connection status, etc. Therefore, the IoT device does not need to upload these local data, just upload the update parameters to the EN for aggregation, and then download the aggregated model parameters from EN. In addition, some IoT devices with insufficient training data can share the training of the DRL agent.

4.5 Theoretical Analysis of Federated Learning-based Policy

Recalling the federal learning-based policy training above, for each agent running on the IoT device, the network state $X_i = (q_i^t, q_i^e, s_i, g_i) \in \mathcal{X}$ will be obtained first, then the action pair (c_i, e_i) will

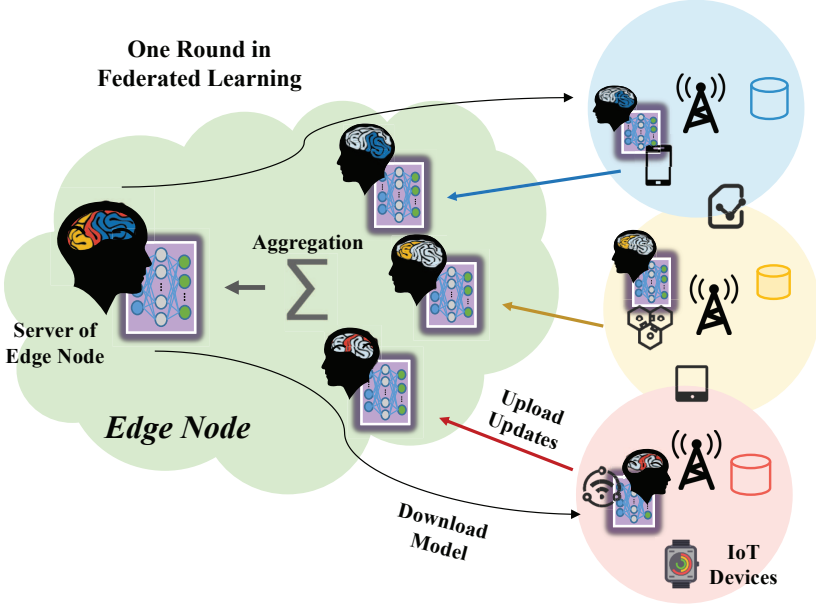


Fig. 7. Federated Learning-based DRL Training.

Algorithm 1: Federated Learning-based Policy Training

```

1 Initialization:
2 Init the set of ENs  $\mathcal{N}$ , the set of IoT devices  $\mathcal{D}$ ;
3 Init  $\{\theta_n, \theta_d, C_d | n \in \mathcal{N}, d \in \mathcal{D}\}$ 
4 Iteration:
5 for  $t = 1, 2, \dots, T$  do
6   Randomly select  $m$  IoT devices  $\mathcal{D}^t \subseteq \mathcal{D}$ ;
7   for each  $d \in \mathcal{D}^t$  do
8     Gets weights  $\theta_n^{t-1}$  of the associated EN;
9     Update local weights  $\theta_d^{t-1} \leftarrow \theta_n^{t-1}$ ;
10    Get local data  $\mathcal{D}_d^t$ ;
11    Get weights and training times  $(\theta_d^t, C_d^t) \leftarrow \text{Train}(\theta_d^{t-1}, \mathcal{D}_d^t)$ ;
12    Upload  $\theta_d^t$  and  $C_d^t$  to EN;
13  end
14  for each  $e \in \mathcal{N}$  do
15    Receive updates from  $\mathcal{D}_n^t \subseteq \mathcal{D}$ ;
16     $\theta_n^t \leftarrow \sum_{d \in \mathcal{D}_n^t} (C_d^t / \sum_{d \in \mathcal{D}_n^t} C_d^t) \cdot \theta_d^t$ ;
17  end
18 end
    
```

be selected according to the policy Φ , after that, a new network state X_{i+1} and a reward r_i will be generated. The process can be described as a Markov decision process.

The more important it is to be closer to the current reward when calculating reward feedback. Therefore, the method of discounted future reward is adopted

$$R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots + \gamma^{n-t} r_n = r_t + \gamma R_{t+1}, \quad (25)$$

where $\gamma = 0.9$ is the discount factor. After that, the basic form of the bellman equation can be obtained by combining equation(25)

$$v(\mathcal{X}_t) = \mathbb{E} [r_t + \gamma v(\mathcal{X}_{t+1}) | \mathcal{X}_t = \mathcal{X}_t], \quad (26)$$

Similar, Q-function is introduced to describe the value of different actions in a certain state. Therefore, $Q(\mathcal{X}^j, (c^j, e^j))$ defined to represent the value of the action pair (c^j, e^j) when it is in state \mathcal{X}^j .

$$Q(\mathcal{X}^{j+1}, (c^{j+1}, e^{j+1})) = \mathbb{E} [r + \lambda Q(\mathcal{X}^j, (c^j, e^j))] \quad (27)$$

However, because the greedy method is used to calculate the Q-function, the calculation of the Q-function may be too close to an earlier calculated local optimal Q-function, resulting in a large deviation, also called the overestimation. To solve overestimation, DDQN decouples the selection of actions and calculates the Q-function [36]. First, it will find the action pair (c^{\max}, e^{\max}) corresponding to the maximum Q-function through Current Network, and then calculate Q-function through Target Network. In addition, for the theoretical proof, the deviation caused by the use of neural network approximation is neglected here, namely

$$Q(\mathcal{X}', (c', e')) = Q(\mathcal{X}^j, (c^j, e^j)) + \alpha (u(\mathcal{X}_j, \Phi(\mathcal{X}_j)) + \gamma \cdot Q(\mathcal{X}', ((c^{\max}, e^{\max}))) - Q(\mathcal{X}^j, (c^j, e^j))), \quad (28)$$

where $\alpha = 0.005$ is the learning rate and \mathcal{X}', c' and e' represent \mathcal{X}^j, c^j and e^j of the next cycle, respectively. Based on the above reasoning, the convergence of the algorithm can be further analyzed, namely:

THEOREM 4.1. *If the following conditions are met, each agent running on the IoT device in the Algorithm 1 will converge to the optimal $Q(\mathcal{X}^*, (c^*, e^*))$ with probability one (w.p.1) when it is updated according to equation (28).*

- 1) *The state and action spaces are finite;*
- 2) $\sum_{t=0}^{+\infty} \alpha = \infty, \sum_{t=0}^{+\infty} (\alpha)^2 < \infty;$
- 3) $\mathbf{Var}\{u(\mathcal{X}_j, \Phi(\mathcal{X}_j))\}$ *is bounded.*

PROOF. In the proof, a result of stochastic approximation given in [17] is first used.

LEMMA 4.2. *A random iterative process $\Delta^{j+1}(x)$, which is defined as*

$$\Delta^{j+1}(x) = (1 - \alpha^j(x)) \Delta^j(x) + \beta^j(x) \Psi^j(x), \quad (29)$$

converges to zero with probability one (w.p.1) if and only if the following conditions are satisfied.

- 1) *The state space is finite;*
- 2) $\sum_{j=0}^{+\infty} \alpha^j = \infty, \sum_{j=0}^{+\infty} (\alpha^j)^2 < \infty, \sum_{j=0}^{+\infty} \beta^j = \infty, \sum_{j=0}^{+\infty} (\beta^j)^2 < \infty,$ *and*
 $\mathbb{E} \{\beta^j(x) | \Lambda^j\} \leq \mathbb{E} \{\alpha^j(x) | \Lambda^j\}$ *uniformly w.p. 1;*
- 3) $\|\mathbb{E} \{\Psi^j(x) | \Lambda^j\}\|_W \leq \varrho \|\Delta^j\|_W,$ *where $\varrho \in (0, 1)$*
- 4) $\mathbf{Var} \{\Psi^j(x) | \Lambda^j\} \leq C \left(1 + \|\Delta^j\|_W\right)^2,$ *where $C > 0$ is a constant.*

Note that $\Lambda^j = \{\Delta^j, \Delta^{j-1}, \dots, \Psi^{j-1}, \dots, \alpha^{j-1}, \dots, \beta^{j-1}\}$ denotes the past at time slot j . $\|\cdot\|_W + W$ denotes some weighted maximum norm.

Subsequent proofs can be made around lemma 4.2, including the transformation of the form into the equation (29) and the four conditions in the lemma 4.2.

a. Transformation equation form. First of all, we can rewrite equation (28) to the following

$$Q(\mathcal{X}', (c', e')) = (1 - \alpha) \cdot Q(\mathcal{X}^j, (c^j, e^j)) + \alpha \cdot (u(\mathcal{X}_j, \Phi(\mathcal{X}_j)) + \gamma \cdot Q(\mathcal{X}', (c^{\max}, e^{\max}))) . \quad (30)$$

By subtracting the optimal $Q(\mathcal{X}^*, (c^*, e^*))$ from both sides of equation (30), it can be rewritten to the following form

$$\Delta^j(\mathcal{X}, (c, e)) = (1 - \alpha^j) \Delta^j(\mathcal{X}, (c, e)) + \alpha^j \Psi^j(\mathcal{X}, (c, e)) , \quad (31)$$

where

$$\Delta^j(\mathcal{X}, (c, e)) = Q(\mathcal{X}', (c', e')) - Q(\mathcal{X}^*, (c^*, e^*)) , \quad (32)$$

$$\Psi^j(\mathcal{X}, (c, e)) = u(\mathcal{X}_j, \Phi(\mathcal{X}_j)) + \gamma \cdot Q(\mathcal{X}', (c^{\max}, e^{\max})) - Q(\mathcal{X}^*, (c^*, e^*)) . \quad (33)$$

Therefore, equation (31) can be seen as equation (29) in lemma 4.2 with $\alpha^j(x) = \beta^j(x)$ and then the theorem 4.1 will be proved by proving that all four conditions of lemma 4.2 have been met.

b. Condition 1) in lemma 4.2. For condition 1) in lemma 4.2, it can be proved by condition 1) in theorem 4.1.

c. Condition 2) in lemma 4.2. The condition 2) in lemma 4.2 also can be easily proved. Since α and β in the equation (29) correspond to the learning rate $\alpha \in (0, 1)$ in the equation (31), the condition is satisfied.

d. Condition 3) in lemma 4.2. To prove condition 3) in lemma 4.2, the concept of contraction mapping is introduced and then prove that $\Psi^j(\mathcal{X}, (c, e))$ is a contraction mapping.

Definition 4.3. For a map $\mathbf{H} : \mathcal{X} \rightarrow \mathcal{X}$ and any $x_1, x_2 \in \mathcal{X}$, if there exists a constant $\delta \in (0, 1)$ satisfies the following equation, then the map \mathbf{H} is a contraction mapping.

$$\|\mathbf{H}x_1 - \mathbf{H}x_2\| \leq \delta \|x_1 - x_2\| \quad (34)$$

Combined with equation (27), the optimal $Q(\mathcal{X}^*, (c^*, e^*))$ can be expressed as

$$Q(\mathcal{X}^*, (c^*, e^*)) = \mathbf{E} [u(\mathcal{X}_*, \Phi(\mathcal{X}_*)) + \lambda Q(\mathcal{X}', (c^{\max}, e^{\max}))] . \quad (35)$$

Then further define the mapping \mathbf{H} as

$$\mathbf{H}_{q(\mathcal{X}', (c', e'))} = \mathbf{E} [u(\mathcal{X}_j, \Phi(\mathcal{X}_j)) + \lambda q(\mathcal{X}', (c^{\max}, e^{\max}))] . \quad (36)$$

After that, combined with the properties of absolute value inequalities and the definition of infinity norm, the following calculations can prove that \mathbf{H} is a contraction mapping.

$$\begin{aligned} \|\mathbf{H}_{q_1(\mathcal{X}', (c', e'))} - \mathbf{H}_{q_2(\mathcal{X}', (c', e'))}\|_{\infty} &= \mathbf{E} [\lambda q_1(\mathcal{X}', (c^{\max}, e^{\max})) - \lambda q_2(\mathcal{X}', (c^{\max}, e^{\max}))] \\ &\leq \mathbf{E} [\lambda |q_1(\mathcal{X}', (c^{\max}, e^{\max})) - q_2(\mathcal{X}', (c^{\max}, e^{\max}))|] \\ &\leq \mathbf{E} \left[\lambda \arg \max_{(c', e')} |q_1(\mathcal{X}', (c', e')) - q_2(\mathcal{X}', (c', e'))| \right] \\ &= \lambda \|q_1(\mathcal{X}', (c', e')) - q_2(\mathcal{X}', (c', e'))\|_{\infty} \end{aligned} \quad (37)$$

According to equations (36) and (33), $\mathbf{E} \{\Psi^j(x)\}$ can be expressed as

$$\begin{aligned} \mathbf{E} \{\Psi^j(x)\} &= \mathbf{E} \{u(\mathcal{X}_j, \Phi(\mathcal{X}_j)) + \gamma \cdot Q(\mathcal{X}', (c^{\max}, e^{\max})) - Q(\mathcal{X}^*, (c^*, e^*))\} \\ &= \mathbf{H}_{Q(\mathcal{X}', (c', e'))} - Q(\mathcal{X}', (c', e')) \\ &= \mathbf{H}_{Q(\mathcal{X}', (c', e'))} - \mathbf{H}_{Q(\mathcal{X}^*, (c^*, e^*))} . \end{aligned} \quad (38)$$

The last step in equation (38) is because $Q(\mathcal{X}', (c', e'))$ is a constant. Finally, we can prove that Condition (3) in Lemma 4.2 satisfies through Equation (39). In the calculation process of Equation

(39), the first step can be derived based on Equation (38), the second step can be derived based on Equation (34), and the last step can be derived based on Equation (32).

$$\begin{aligned} \|\mathbf{E}\{\Psi^j(x)\}\|_\infty &= \|\mathbf{H}_{Q(\mathcal{X}',(c',e'))} - \mathbf{H}_{Q(\mathcal{X}^*,(c^*,e^*))}\|_\infty \\ &\leq \delta \|Q(\mathcal{X}',(c',e')) - Q(\mathcal{X}^*,(c^*,e^*))\|_\infty \\ &= \delta \|\Delta^j(\mathcal{X},(c,e))\|_\infty \end{aligned} \quad (39)$$

e. Condition 4) in lemma 4.2. Next is the proof of the condition 4) in lemma 4.2. The following calculation uses equation (33), equation (36) and the properties of variance.

$$\begin{aligned} \text{Var}\{\Psi^j(\mathcal{X},(c,e))\} &= \mathbf{E}\{u(\mathcal{X}_j, \Phi(\mathcal{X}_j)) + \gamma \cdot Q(\mathcal{X}',(c^{\max},e^{\max})) - Q(\mathcal{X}^*,(c^*,e^*)) \\ &\quad - \mathbf{H}_{Q(\mathcal{X}',(c',e'))} + Q(\mathcal{X}^*,(c^*,e^*))\} \\ &= \mathbf{E}\{u(\mathcal{X}_j, \Phi(\mathcal{X}_j)) + \gamma \cdot Q(\mathcal{X}',(c^{\max},e^{\max})) - \mathbf{H}_{Q(\mathcal{X}',(c',e'))}\} \\ &= \mathbf{E}\{u(\mathcal{X}_j, \Phi(\mathcal{X}_j)) + \gamma \cdot Q(\mathcal{X}',(c^{\max},e^{\max})) \\ &\quad - \mathbf{E}[u(\mathcal{X}_j, \Phi(\mathcal{X}_j)) + \gamma \cdot Q(\mathcal{X}',(c^{\max},e^{\max}))]\} \\ &= \text{Var}\{u(\mathcal{X}_j, \Phi(\mathcal{X}_j)) + \gamma \cdot Q(\mathcal{X}',(c^{\max},e^{\max}))\} \\ &\leq C(1 + \|\Delta^j(\mathcal{X},(c,e))\|_\infty), \end{aligned} \quad (40)$$

where C is a constant and the last step in the above calculation is because of $u(\mathcal{X}_j, \Phi(\mathcal{X}_j))$ is bounded and $Q(\mathcal{X}',(c^{\max},e^{\max}))$ at most linearly. \square

The above proof process has proved that the four conditions in lemma 4.2 are satisfied, and based on this, theorem 4.1 can be proved.

5 PERFORMANCE EVALUATION

In this section, we conduct some simulation experiments to explore the impact of various parameters on FL-based DRL training and the performance of FL-based DRL training and centralized DRL training.

5.1 Experiment Settings

To evaluate the capability of our proposed method, we simulate an edge computing system. The channel gain of the network connection between the IoT device and EN is defined as 6 levels. Besides, the settings for the DRL agent are shown in table 2.

Table 2. The settings for the DRL agent

Parameters	Values
The number of full connected layers	2
The number of neurons	200
Activation function	tanh
Discount factor	0.9
Replay memory capacity	5000
Update interval of the target Q network	250

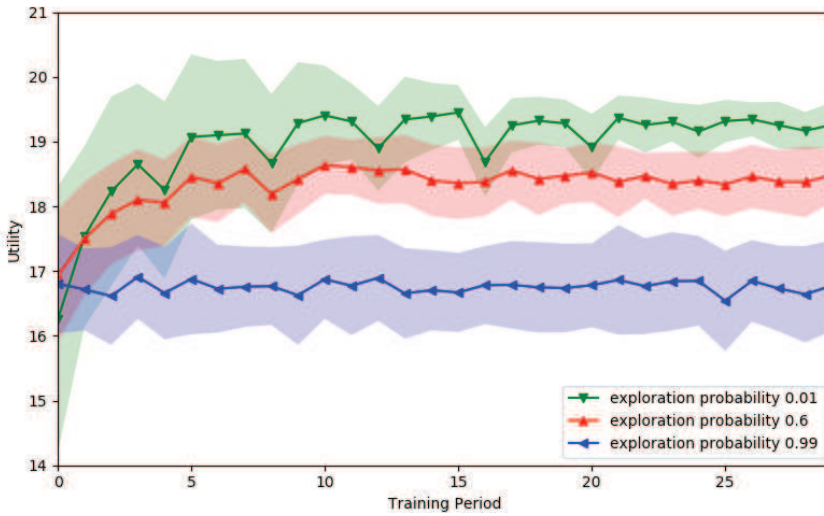


Fig. 8. Exploration effect.

5.2 Analysis of Exploration Probability

A strategy called ϵ -greedy is used in DDQN, which is a similar approach to greedy strategy. Since there is no corresponding Q value for the state-action pair that has not appeared and these combinations will never be tried if the greedy strategy is used directly, so the concept of exploration and utilization is integrated. On the one hand, we must seek to maximize the utility from the information already known, and on the other hand, we must explore what is not known in the environment. The ϵ -greedy is to add exploration rates based on a greedy strategy. For each decision, there will be a probability to randomly select actions to explore, and there is a probability to use a greedy strategy to maximize the utility. Therefore, we first explored the effect of exploration probability on utility in the experiment.

We selected three values for exploration probability to test and one hundred times of experiments are taken for collecting statistics. In Fig. 8, solid lines and the shallow area around them represent mean values and the standard deviation, respectively. It can be found that the average value of utility is the highest when exploration probability = 0.01, and the standard deviation is the smallest. When exploration probability = 0.6, the average value of utility will decrease slightly and the standard deviation will increase. When exploration probability = 0.99, the average value of utility will be greatly reduced and the range of standard deviation will be further increased. It can be found that a higher exploration probability will have a negative impact on performance of IoT devices. This is due to the fact that IoT devices in most cases choose to explore and do not fully utilize the previous training results, resulting in poor performance. Based on the results, accordingly, we choose the final exploration possibility as 0.01 in the following experiments.

5.3 Analysis of Task Generate Probability

When the IoT device is applied, the workload of some IoT devices is heavy, and some IoT devices are relatively idle. Even for an IoT device, the workload will change greatly during different time periods. Therefore, we further explore the effect of workload.

Under different probabilities of task generation, the performance of the proposed algorithm is showed in Figure 9. When the task generates probability = 0.1, the IoT device can run at a higher

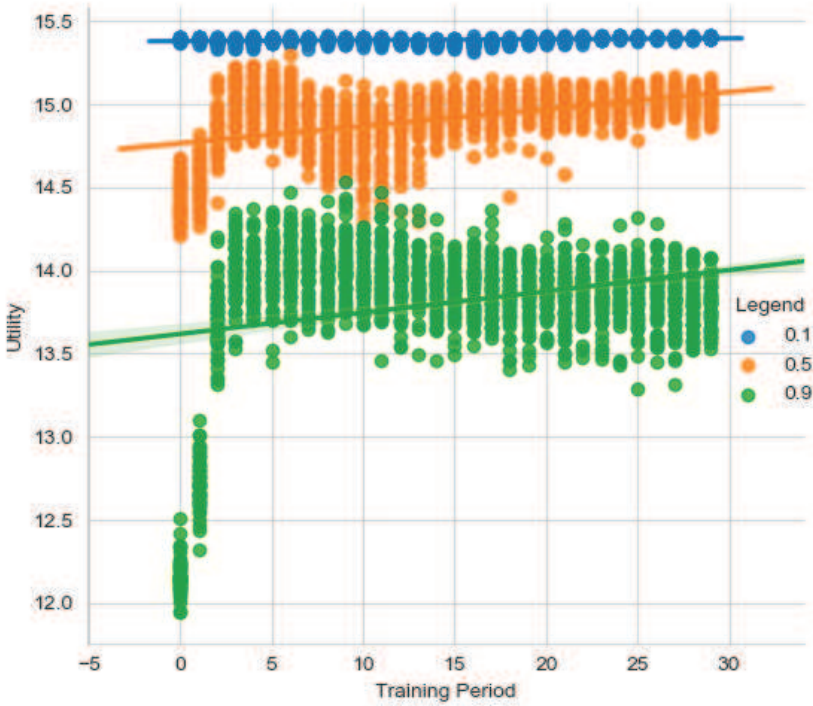


Fig. 9. Utility varies with workload.

utility and the standard deviation of utility is also small, namely, the performance is very good and stable. When the task generates probability = 0.5, the workload of the IoT device will be relatively heavy. In the initial stage of training, the utility is lower and the standard deviation is large, that is, the performance is poor and unstable. However, as the training progressed, the utility stabilized at a higher level and the standard deviation is greatly reduced. When the task generate probability = 0.9, the workload of the IoT device is extremely heavy and the overall utility value is very low. It can be found that, when the exploration probability is higher, the performance of the proposed algorithm will be worse and more unstable. In other words, a higher exploration probability will have a negative impact on the performance. This may be due to the limitation of IoT devices' performance, because a high exploration probability will produce a large number of tasks that IoT devices cannot cope with, resulting in high queuing delay and a large number of task failures. In addition, due to the limited energy of IoT devices, heavy tasks will result in low or no energy allocated for each task, thus limiting the performance of IoT devices. However, after a period of training, the IoT device's utility value is improved and stabilized at a relatively high level, which proves the effectiveness of the algorithm under a heavy workload.

To further explore the performance of our proposed algorithm under different workloads, we selected four key parameters queuing delay, offload payment, task execution delay, and task drop number for comparison. The performance of the algorithm presented under different workloads is shown in Fig. 10. When the task generates probability = 0.1, the IoT device needs to handle few tasks and is very good at all key parameters. When the task generates probability = 0.5, the workload of the IoT device is increased, and various key parameters are beginning to increase.

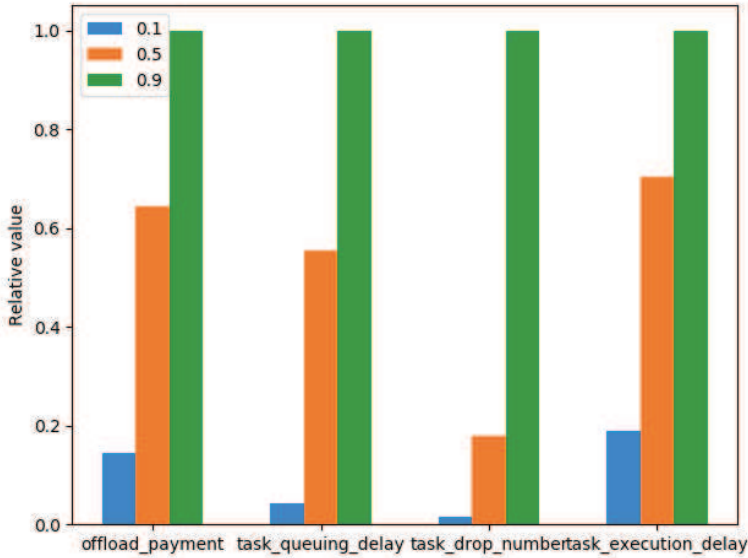


Fig. 10. Key parameters that vary with workload.

However, the increase of the task drop number is small, which means that the IoT device can make most tasks complete normally under the current workload. When the task generate probability = 0.9, the workload of the IoT device is further increased, which will make the key parameters perform worse.

In summary, Fig. 9 and Fig. 10 show the performance of one IoT device with varying the workload on itself, which corroborates that our work can adapt to the workload variation and be converged.

5.4 Analysis of Energy Generate Probability

Differences in the deployment conditions of IoT devices may cause them to face different probabilities of energy generation. Therefore, in order to test the performance of the algorithm under different energy generation, we conducted the following experiments.

Under different probabilities of energy generation, the performance of the proposed algorithm is showed in Figure 11. When energy generation probability = 0.1, IoT devices obtain energy at a very slow speed, resulting in energy shortage which leads to a lower utility level. When energy generation probability = 0.5, the speed of obtaining the energy of IoT equipment increases obviously. Besides, the utility is improved significantly overall and shows a tendency to gradually increase and eventually stabilize. When energy activates probability = 0.9, the energy of the IoT device is more abundant. The utility is further improved, and the standard deviation is reduced, that is, the IoT device performs better and more stable. It can be found that the higher the energy generate probability is, the more sufficient the energy of IoT equipment is, so higher power can be used to perform tasks and the overall performance is better.

To more closely compare the performance of the proposed algorithm in different energy environments, we selected some key parameters for comparison. The algorithm proposed in different energy environments is shown in Fig. 12. When energy generation probability = 0.1, the queuing delay, task drop number, and task execution delay is very high which may be due to insufficient

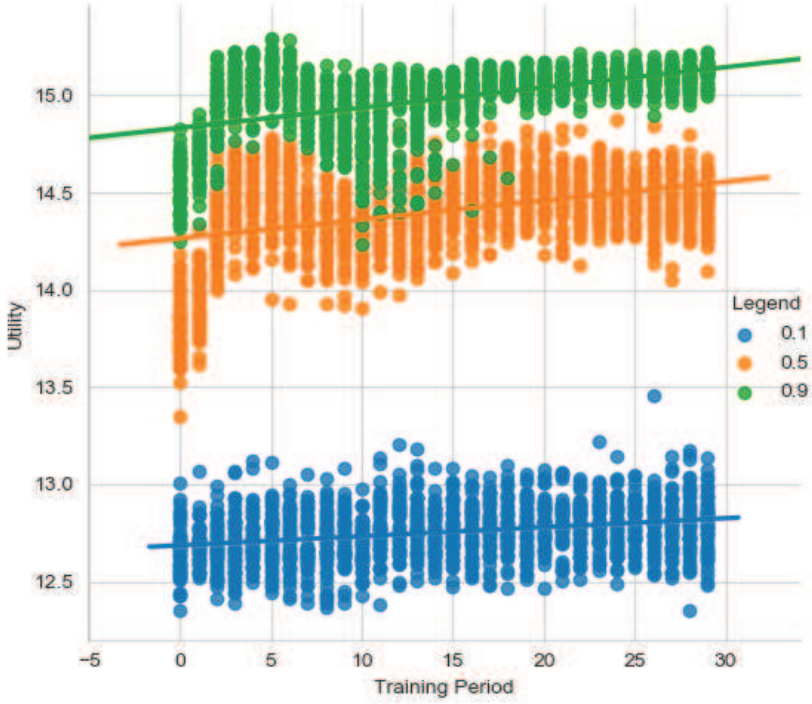


Fig. 11. Utility varies with energy generate probability.

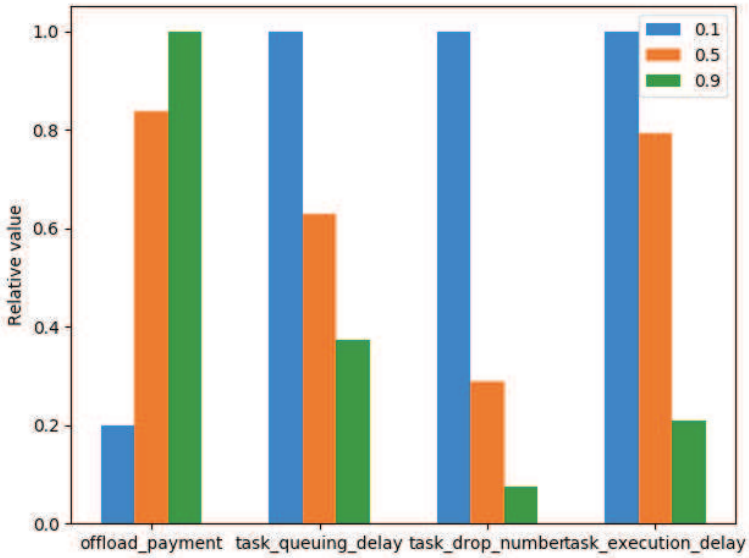


Fig. 12. Key parameters that vary with energy generate probability.

energy, resulting in the task can not be completed in time. However, the offload payment at this time is very low, which means that most tasks are executed locally rather than being transferred to EN. When energy generation probability = 0.5, the IoT device has more energy available, and the offload payment is further improved but other parameters are reduced. This indicates that more tasks are transferred to EN and the task execution is faster. When energy generation probability = 0.9, the energy of the IoT device is sufficient. In this case, more tasks are transferred to EN for execution and the proposed algorithm performs better. Therefore, it can be seen that the higher the energy generate probability is, the more tasks submitted to EN for running. That is to say, compared with local execution of IoT devices, transferring tasks to EN will consume more energy but obtain more powerful task processing capability.

Therefore, experiments show that the proposed algorithm can adapt to different energy to generate probability and perform convergence.

5.5 Analysis of the Number of IoT Devices

When the FL-based DRL training is running, it will be trained using many IoT devices in the area. However, the number of IoT devices used is difficult to determine, so the impact of the number of different IoT devices will be explored next.

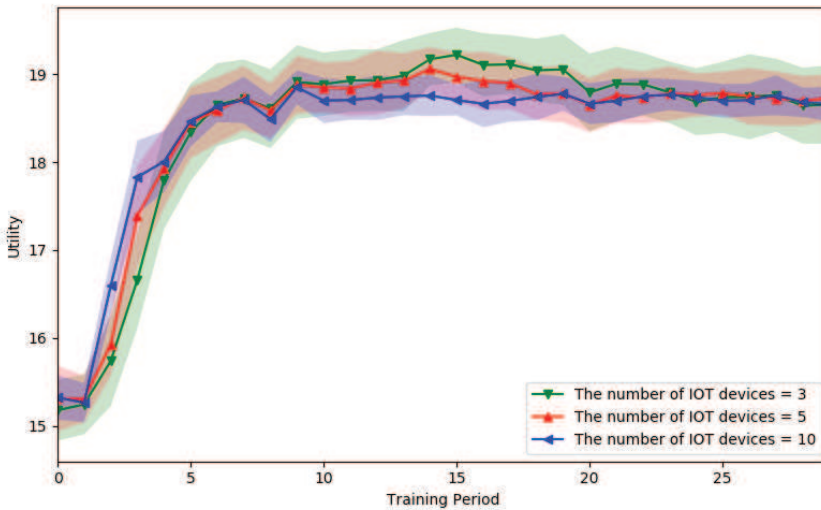


Fig. 13. Performance under different numbers of IoT devices.

The experiment selected different numbers of IoT devices for testing and collected their utilities as a comparison. As shown in Fig. 13, all their utilities show an upward trend at the early stage and remain stable at the later stage. However, the number of different IoT devices also has an impact on their utilities. On the one hand, when the number of IoT devices is small, its utility increases relatively slowly in the early days. On the other hand, for different IoT devices, all the utility will be stable at the same level in the later stage, but the number of IoT devices will have an impact on the standard deviation of the utility. When the number of IoT devices is small, the standard deviation of the utility will be large, that is, more IoT devices will make the performance more stable. We think that when the number of IoT devices is large, more environmental states can be learned in the same training cycle, thus contributing to the training progress of IoT devices in the early stage. Therefore, the more IoT devices in the early stage, the performance is better.

However, after training convergence, the more IoT devices there are, the worse their performance will be. This may be because different IoT devices face different environmental states, so their model parameters are not applicable to all IoT devices, that is, aggregating model parameters of IoT devices under different environmental states cannot produce optimal results.

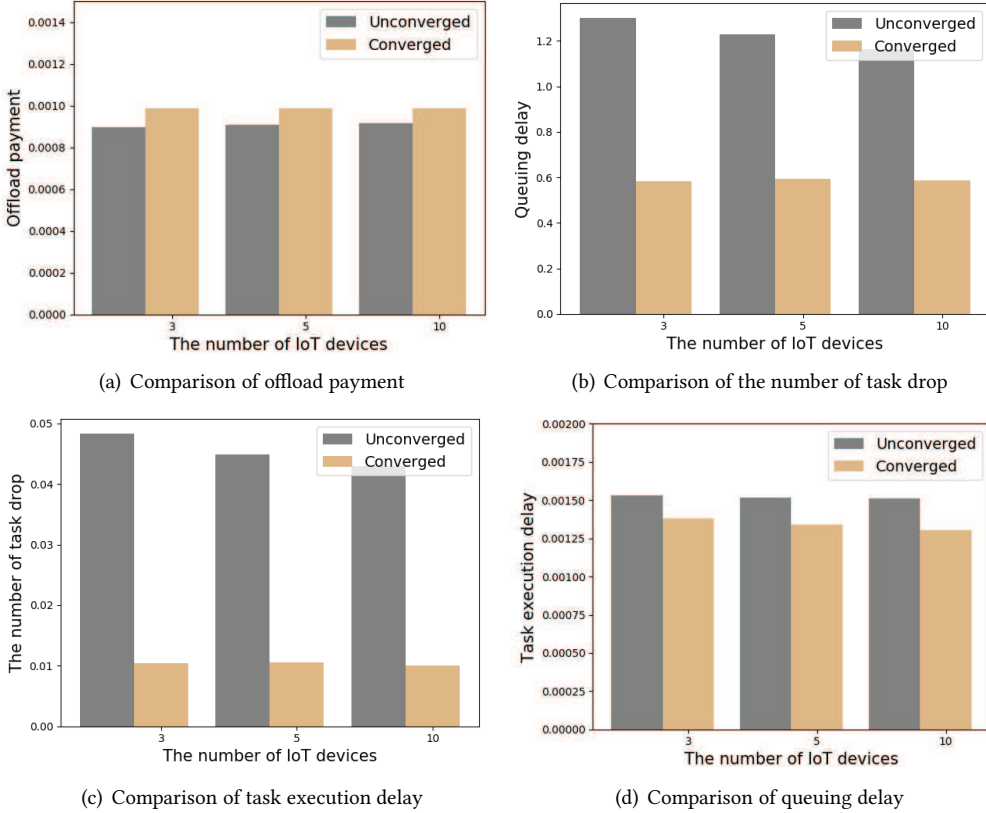


Fig. 14. Comparison of parameters before and after convergence for different number of IoT devices.

To delve deeper into the impact of the number of different IoT devices, we conducted experiments on more detailed parameters. As shown in Fig. 14, when comparing the convergence and non-convergence of a certain IoT device, it can be found that offload payment increases slightly after training convergence, but all other parameters decrease. The reason for this situation may be the increased use of EN after training convergence. Therefore, it costs more to use EN while other parameters are optimized. In addition, for the case of the number of different IoT devices, they are at the same level after the training convergence.

In summary, the more IoT devices, the faster the convergence rate during training, that is, the better the performance in the early stage. However, after training convergence, the performance of different IoT devices is at the same level.

5.6 Analysis of Energy Consumption

For IoT devices, many devices use batteries as a power source, so they usually have some limitations in terms of energy. Therefore, energy consumption is a parameter that needs to be considered.

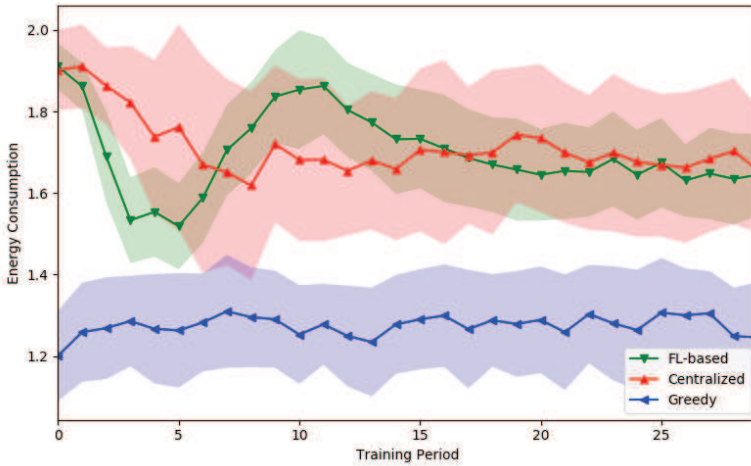


Fig. 15. Comparison of energy consumption.

As shown in Fig. 15, the energy consumption of both FL-based DRL training and centralized DRL training is higher than the greedy strategy. The reason may be that more energy has to be used to reduce the number of task drop and task execution delay. In addition, the average energy consumption of FL-based DRL training is comparable to that of centralized DRL training, but the standard deviation is relatively small.

In summary, our proposed algorithm may result in higher energy consumption, which may be due to higher power for local execution or data transmission. In this case, although the energy consumption may be at a high level, it is improved in terms of time delay, the number of task drop, etc. In addition, it is worth mentioning that the energy consumption of FL-based DRL training and centralized DRL training are at the same level.

5.7 Analysis of Key Parameters in the System Model

In order to deeply compare the characteristics of FL-based DRL training and centralized DRL training, some key parameters in the system model are selected for comparison and the first is the task execution delay. Task execution delay is the time from when the task leaves the task queue to completion, including the time of establishing a connection, the time of transferring task data and the time of executing the task. The task execution delay of the FL-based DRL training and centralized DRL training is shown in Fig. 16.

It can be seen that the FL-based DRL training has a higher task execution delay than the centralized DRL training task execution delay, and both of them stay stable after falling in the early stage. It is worth noting that the task execution delay of the FL-based DRL training has decreased more in the early stage and reached a steady-state faster, that is, the L-based DRL training has a faster convergence speed.

In addition, the queuing delay of FL-based DRL training and centralized DRL training is compared. The queuing delay is the time it takes for a task to go from being placed in the task queue to being taken out of the task queue. The queuing delay of FL-based DRL training and centralized DRL training is shown in Fig. 17.

The queuing delay of the FL-based DRL training is generally lower than that of the centralized DRL training, but the queuing delay of the FL-based DRL training is higher overall.

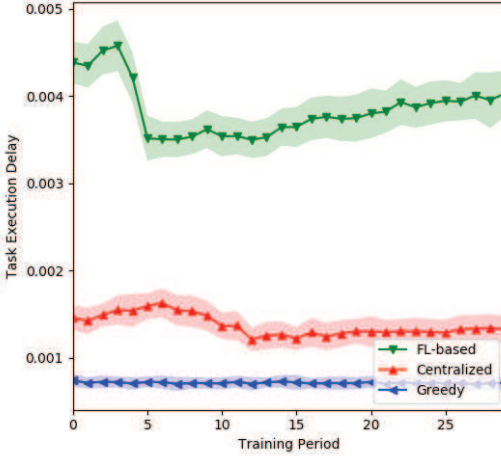


Fig. 16. Comparison of task execution delay.

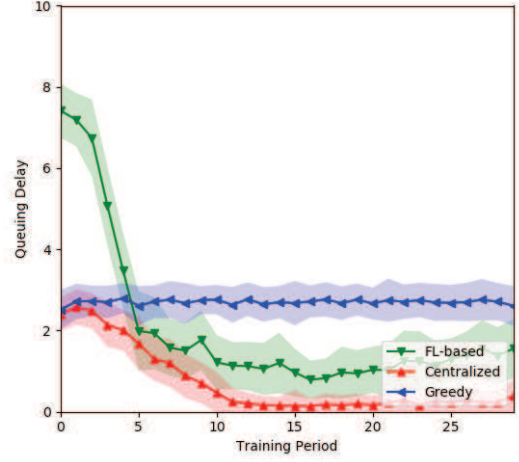


Fig. 17. Comparison of queuing delay.

Next is the comparison of the number of task drop of FL-based DRL training and centralized DRL training. The number of task drop refers to the number of tasks lost when the number of newly generated tasks in the task queue reaches the upper limit. The number of task drop of FL-based DRL training and centralized DRL training is shown in Fig. 18.

The number of task drop of FL-based DRL training is much higher than the centralized DRL training in the early stage, but the number of task drop of FL-based DRL training decreases rapidly and ultimately stable not far from the centralized DRL trained. Overall, both of them have significantly decreased in the early stage which indicates the effectiveness of the training, but the number of task drop of FL-based DRL training will be higher than the centralized DRL training. This may be due to the queuing delay and the task execution delay of the FL-based DRL training are higher so that many tasks in the task queue cannot be completed in time. Therefore, the task queue in FL-based DRL training is more easily full, which leads to the loss of tasks.

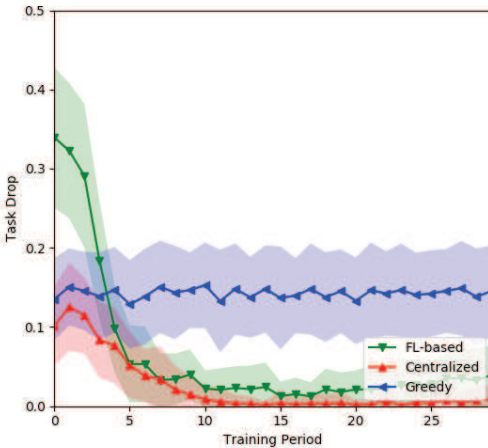


Fig. 18. Comparison of the number of task drop.

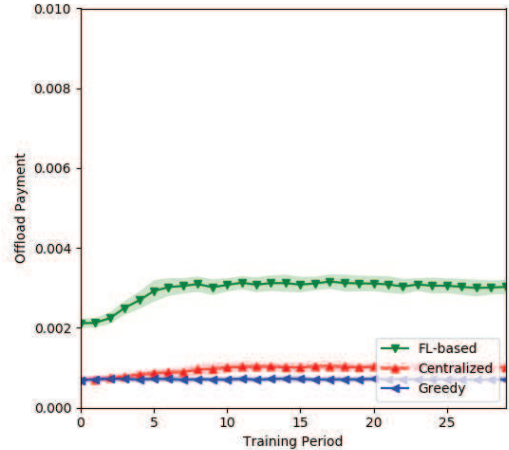


Fig. 19. Comparison of offload payment.

Besides, due to the limited computing resources of EN, the offload payment of FL-based DRL training and centralized DRL training is also recorded. Offload payment refers to the payment required to use EN, and its value is positively related to the duration of the occupation. As shown in Fig. 19, the offload payment of FL-based DRL training and centralized DRL training are relatively stable overall, but FL-based DRL training takes up more time on EN.

This part of the experiment compares the centralized DRL training, FL-based DRL training and the greedy strategy in several key parameters in detail. One of the prominent characteristics is that FL-based DRL training will transfer more tasks to the edge node for execution, resulting in the change of key parameters.

5.8 Analysis of Computation Offloading Performance

Since centralized DRL training is the benchmark for our proposed policy, we developed a further comparison with it.

Randomly soliciting three IoT devices for investigation, Fig. 20(a)-20(c) and Fig. 21(a) present the performance of FL-enabled DRL training and centralized DRL training, respectively. Accordingly, their training loss statics are correspondingly given in Fig. 20(d)-20(f) and Fig. 21(b).

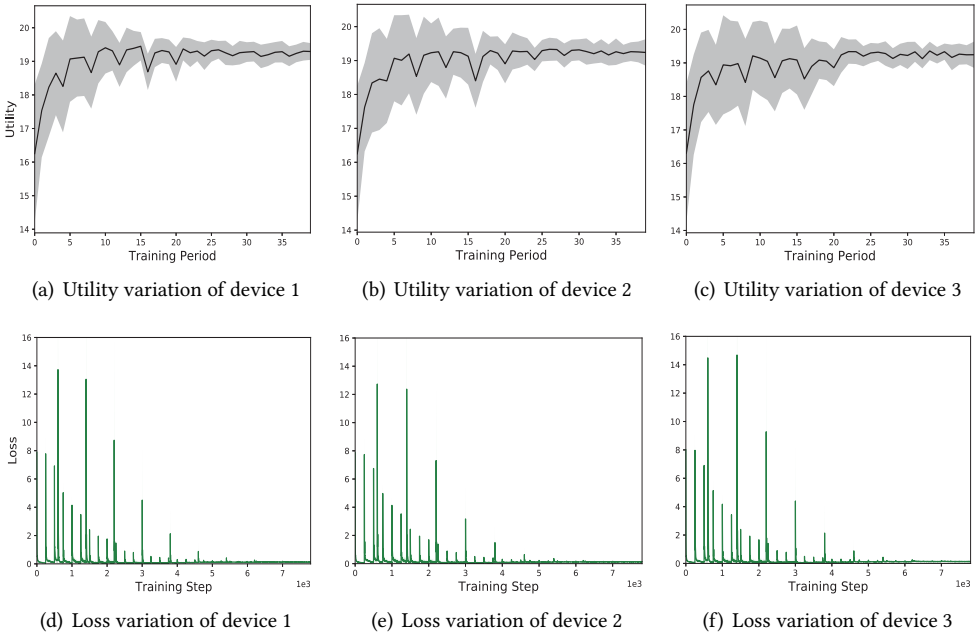


Fig. 20. Computation offloading performance with Federated Learning-based.

The performance evaluation was as follows by a comparison of experimental results. **1)** Obviously, the standard deviation of the training of the centralized training is less than the FL-based DRL training. This shows that the centralized training has better stability during training. In addition, as the training continues, the utility and standard deviations of FL-based DRL training continue to shrink, and eventually at the same level as centralized training. This experimental result verifies that FL-based DRL training achieves the same level of performance as centralized training and verifies its effectiveness. **2)** Since it is assumed that the wireless channel in the centralized

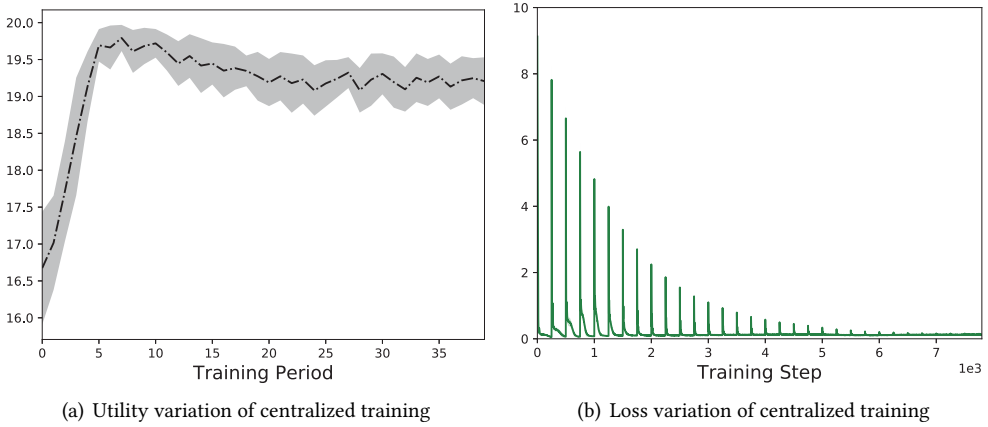


Fig. 21. Computation offloading performance with centralized DRL training training.

training can upload the training data to the EN without loss and does not cause the delay. In fact, this is impossible and this further proves the effectiveness of FL-based DRL training. Under this assumption, once trained for a period of time, FL-based DRL training can achieve the same level of performance as centralized training. Therefore, in the actual situation, the centralized training performance will be even better.

In addition, FL has two disadvantages. On the one hand, FL-based DRL training not only performs poorly during training but also requires a longer time to converge. On the other hand, FL-based DRL training is at the same level but relatively poor compared to centralized training. So how to fine-grain the scheduling of data transmission for optimization can be considered in future work.

6 CONCLUSIONS

In this paper, we investigate a computation offloading optimization problem. In more detail, each IoT device can make decisions about offloading tasks and allocating energy to maximize the expected long-term utility. For this, we propose an offloading algorithm based on FL and DRL and run in parallel across multiple IoT devices. On the one hand, DRL enables each IoT device to make decisions independently according to its own dynamic environment. On the other hand, FL further reduces the transmission consumption between IoT devices and EN and greatly enhances the privacy protection of data. In addition, we also carried out the necessary theoretical analysis and implemented a series of simulation experiments for this algorithm. The experimental results show that the proposed algorithm is applicable to various environments and verify the effectiveness of FL-based DRL. In the future, we will take a deep study in if there are model compression techniques with respect to DRL, and how to schedule the FL-based DRL training in a fine-grained fashion.

REFERENCES

- [1] Nasir Abbas, Yan Zhang, Amir Taherkordi, and Tor Skeie. 2017. Mobile edge computing: A survey. *IEEE Internet of Things Journal* 5, 1 (2017), 450–465.
- [2] Kofi Sarpong Adu-Manu, Nadir Adam, Cristiano Tapparello, Hoda Ayatollahi, and Wendi Heinzelman. 2018. Energy-harvesting wireless sensor networks (EH-WSNs): A review. *ACM Transactions on Sensor Networks (TOSN)* 14, 2 (2018), 10.
- [3] Ganesh Ananthanarayanan, Paramvir Bahl, Peter Bodík, Krishna Chintalapudi, Matthai Philipose, Lenin Ravindranath, and Sudipta Sinha. 2017. Real-time video analytics: The killer app for edge computing. *computer* 50, 10

- (2017), 58–67.
- [4] Bigmoyan. 2017. Keras: The Python Deep Learning library. <https://keras.io>
 - [5] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konecny, Stefano Mazzocchi, H Brendan McMahan, Timon Van Overveldt, David Petrou, Daniel Ramage, and Jason Roselander. 2019. Towards federated learning at scale: System design. *arXiv preprint arXiv:1902.01046* (2019).
 - [6] Min Chen and Yixue Hao. 2018. Task Offloading for Mobile Edge Computing in Software Defined Ultra-Dense Network. *IEEE Journal on Selected Areas in Communications* 36, 3 (2018), 587–597. <https://doi.org/10.1109/JSAC.2018.2815360>
 - [7] Xu Chen, Lei Jiao, Wenzhong Li, and Xiaoming Fu. 2016. Efficient Multi-User Computation Offloading for Mobile-Edge Cloud Computing. *IEEE/ACM Trans. Netw.* 24, 5 (2016), 2795–2808. <https://doi.org/10.1109/TNET.2015.2487344>
 - [8] Xianfu Chen, Honggang Zhang, Celimuge Wu, Shiwen Mao, Yusheng Ji, and Mehdi Bennis. 2018. Optimized Computation Offloading Performance in Virtual Edge Computing Systems via Deep Reinforcement Learning. *IEEE Internet of Things Journal* (2018), 1–1. <https://doi.org/10.1109/JIOT.2018.2876279>
 - [9] ETSI. 2014. Mobile-Edge Computing—Introductory Technical White Paper. https://portal.etsi.org/Portals/0/TBpages/MEC/Docs/Mobile-edge_Computing_-_Introductory_Technical_White_Paper_V1%2018-0
 - [10] Michael R Garey and David S Johnson. 1978. “Strong”NP-Completeness Results: Motivation, Examples, and Implications. *Journal of the ACM (JACM)* 25, 3 (1978), 499–508.
 - [11] Marco ET Gerards, Johann L Hurink, and Jan Kuper. 2014. On the interplay between global DVFS and scheduling tasks with precedence constraints. *IEEE Trans. Comput.* 64, 6 (2014), 1742–1754.
 - [12] Google. 2017. Deploy machine learning models on mobile and IoT devices. <https://www.tensorflow.org/lite>
 - [13] Google. 2017. Federated Learning: Collaborative Machine Learning without Centralized Training Data. <https://research.googleblog.com/2017/04/federated-learning-collaborative.html>
 - [14] Dennis Grewe, Marco Wagner, Mayutan Arumathurai, Ioannis Psaras, and Dirk Kutscher. 2017. Information-centric mobile edge computing for connected vehicle environments: Challenges and research directions. In *Proceedings of the Workshop on Mobile Edge Communications*. ACM, 7–12.
 - [15] Xiaoya Hu, Bingwen Wang, and Han Ji. 2013. A wireless sensor network-based structural health monitoring system for highway bridges. *Computer-Aided Civil and Infrastructure Engineering* 28, 3 (2013), 193–209.
 - [16] SM Riazul Islam, Daehan Kwak, MD Humaun Kabir, Mahmud Hossain, and Kyung-Sup Kwak. 2015. The internet of things for health care: a comprehensive survey. *IEEE Access* 3 (2015), 678–708.
 - [17] Tommi Jaakkola, Michael I Jordan, and Satinder P Singh. 1994. Convergence of stochastic iterative dynamic programming algorithms. In *Advances in neural information processing systems*. 703–710.
 - [18] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. 2014. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*. ACM, 675–678.
 - [19] Yin Jie, Ji Yong Pei, Li Jun, Guo Yun, and Xu Wei. 2013. Smart home system based on iot technologies. In *2013 International Conference on Computational and Information Sciences*. IEEE, 1789–1791.
 - [20] David S Johnson and Michael R Garey. 1979. *Computers and intractability: A guide to the theory of NP-completeness*. Vol. 1. WH Freeman San Francisco.
 - [21] Md Kamruzzaman, Nurul I Sarkar, Jairo Gutierrez, and Sayan Kumar Ray. 2017. A study of IoT-based post-disaster management. In *2017 International Conference on Information Networking (ICOIN)*. IEEE, 406–410.
 - [22] Nacer Khalil, Omprakash Gnawali, Driss Benhaddou, and Jaspal Subhlok. 2018. SonicDoor: A Person Identification System Based on Modeling of Shape, Behavior, and Walking Patterns. *ACM Transactions on Sensor Networks (TOSN)* 14, 3-4 (2018), 27.
 - [23] Jakub Konečný, H Brendan McMahan, Daniel Ramage, and Peter Richtárik. 2016. Federated optimization: Distributed machine learning for on-device intelligence. *arXiv preprint arXiv:1610.02527* (2016).
 - [24] Jakub Konečný, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. 2016. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492* (2016).
 - [25] Yuyi Mao, Changsheng You, Jun Zhang, Kaibin Huang, and Khaled B. Letaief. 2017. A Survey on Mobile Edge Computing: The Communication Perspective. *IEEE Communications Surveys & Tutorials* 19, 4 (2017), 2322–2358.
 - [26] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. 2017. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, AISTATS 2017, Fort Lauderdale, FL, USA (Proceedings of Machine Learning Research)*, Vol. 54. PMLR, 1273–1282. <http://proceedings.mlr.press/v54/mcmahan17a.html>
 - [27] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharmashan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529–533. <https://doi.org/10.1038/nature14236>

- [28] REGULATION (EU) 2016/679 OF THE EUROPEAN PARLIAMENT and OF THE COUNCIL. 2016. Mobile-Edge Computing–Introductory Technical White Paper. <https://eur-lex.europa.eu/legal-content/EN/TXT/?qid=1564833514064&uri=CELEX:32016R0679>
- [29] Alexander Pyattaev, Kerstin Johnsson, Sergey Andreev, and Yevgeni Koucheryavy. 2015. Communication challenges in high-density deployments of wearable wireless devices. *IEEE Wireless Communications* 22, 1 (2015), 12–18.
- [30] Yongming Rao, Jiwen Lu, and Jie Zhou. 2017. Attention-aware deep reinforcement learning for video face recognition. In *Proceedings of the IEEE International Conference on Computer Vision*. 3931–3940.
- [31] S Sujin Issac Samuel. 2016. A review of connectivity challenges in IoT-smart home. In *2016 3rd MEC International conference on big data and smart city (ICBDSC)*. IEEE, 1–4.
- [32] Pararth Shah, Marek Fiser, Aleksandra Faust, J Chase Kew, and Dilek Hakkani-Tur. 2018. Follownet: Robot navigation by following natural language directions with deep reinforcement learning. *arXiv preprint arXiv:1805.06150* (2018).
- [33] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. 2016. Edge Computing: Vision and Challenges. *IEEE Internet of Things Journal* 3, 5 (2016), 637–646. <https://doi.org/10.1109/JIOT.2016.2579198>
- [34] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. 2016. Mastering the game of Go with deep neural networks and tree search. *nature* 529, 7587 (2016), 484.
- [35] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.
- [36] Hado van Hasselt, Arthur Guez, and David Silver. 2016. Deep Reinforcement Learning with Double Q-Learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, 2016, Phoenix, Arizona, USA*. AAAI Press, 2094–2100. <http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12389>
- [37] Tian Wang, Guangxue Zhang, Anfeng Liu, Md Zakirul Alam Bhuiyan, and Qun Jin. 2018. A secure IoT service architecture with an efficient balance dynamics based on cloud and edge computing. *IEEE Internet of Things Journal* (2018).
- [38] Yuchen Yang, Longfei Wu, Guisheng Yin, Lijie Li, and Hongbin Zhao. 2017. A survey on security and privacy issues in Internet-of-Things. *IEEE Internet of Things Journal* 4, 5 (2017), 1250–1258.
- [39] Engin Zeydan, Ejder Bastug, Mehdi Bennis, Manhal Abdel Kader, Ilyas Alper Karatepe, Ahmet Salih Er, and Merouane Debbah. 2016. Big data caching for networking: moving from cloud to edge. *IEEE Communications Magazine* 54, 9 (sep 2016), 36–42. <https://doi.org/10.1109/MCOM.2016.7565185>