Resource Allocation With Edge Computing in IoT Networks via Machine Learning

Xiaolan Liu, *Student Member, IEEE*, Jiadong Yu, *Student Member, IEEE*, Jian Wang, *Member, IEEE*, and Yue Gao, *Senior Member, IEEE*

***Abstract*—In this article, we investigate resource allocation with edge computing in Internet-of-Things (IoT) networks via machine learning approaches. Edge computing is playing a promising role in IoT networks by providing computing capabilities close to users. However, the massive number of users in IoT networks requires sufficient spectrum resource to transmit their computation tasks to an edge server, while the IoT users were developed to have more powerful computation ability recently, which makes it possible for them to execute some tasks locally. Then, the design of computation task offloading policies for such IoT edge computing systems remains challenging. In this article, centralized user clustering is explored to group the IoT users into different clusters according to users' priorities. The cluster with the highest priority is assigned to offload computation tasks and executed at the edge server, while the lowest priority cluster executes computation tasks locally. For the other clusters, the design of distributed task offloading policies for the IoT users is modeled by a Markov decision process, where each IoT user is considered as an agent which makes a series of decisions on task offloading by minimizing the system cost based on the environment dynamics. To deal with the curse of high dimensionality, we use a deep *Q*-network to learn the optimal policy in which deep neural network is used to approximate the *Q*-function in *Q*-learning. Simulations show that users are grouped into clusters with optimal number of clusters. Moreover, our proposed computation offloading algorithm outperforms the other baseline schemes under the same system costs.**

***Index Terms*—Deep reinforcement learning (DRL), edge computing, resource allocation, user clustering.**

I. Introduction

A. Basic Knowledge

THE Internet of Things (IoT) has brought about a new era of smart applications, such as smart city, smart industry, as well as smart farming by using a lot of IoT devices, like sensors, actuators, and gateways to collect and process a large amount of data generated by the IoT networks [1]. Innovations in hardware and software in recent years have contributed to the expansion of IoT networks with a large number of connected devices, which increases the requirements for data processing, storage, and communications [2]. For example, in smart farming, many sensors are deployed to monitor the environmental factors and animal welfare, like temperature, humidity, level of lighting, noise, and gas levels. Generally, the generated data by sensors are collected by the gateway and then sent to the cloud server to perform further processing [3]. However, few challenges need to be addressed in this situation. First, many farms are located in remote areas where sensors might not be able to connect to the cloud services, edge computing, defined as providing computing capacities close to users, is a promising solution to complete the computation chain [3], [4]. Second, a large number of sensors requests for computation resource from the edge server, which burdens on the spectrum resource and the computation capacity. Moreover, the IoT devices have been developed to have more powerful computation ability, which makes it possible for them to perform some simple data processing. Therefore, the design of the optimal computation task offloading scheme is necessary and urgent in the IoT edge computing networks. A multiobjective optimization problem has been formulated to jointly minimize the cost, including energy consumption, execution delay, and payment cost by finding the optimal offloading probability and transmit power for each mobile device [5]. Moreover, game theory has been proposed to design a dynamic computation offloading scheme in the fog computing system with energy harvesting [6].

Compared to the cloud server, edge computing was proposed to support latency-critical services and remote IoT applications. As mentioned above, even the IoT users have more powerful computation capacities, they are still suffering serious resource scarcity problems, such as limited battery power and CPU computation resource [7]. Thus, offloading part of computation tasks to relatively resource-rich edge devices can meet the Quality-of-Service (QoS) requirements of different applications as well as augment the capabilities of IoT users for running resource-demanding applications [8]. However, the edge device, i.e., the gateway in IoT networks, cannot support a large number of IoT users to offload their computation tasks together due to the finite spectrum resource [9]. Hence, grouping the IoT users

Manuscript received September 27, 2019; revised November 24, 2019 and January 7, 2020; accepted January 19, 2020. Date of publication January 28, 2020; date of current version April 14, 2020. This work was supported in part by the Research and Innovation Bridges Cooperation Program between the U.K. and China, in part by the Ministry of Science and Technology in China under Grant 2016YFE0124200, in part by the Innovate U.K., in part by the Biotechnology and Biological Sciences Research Council under Grant BB/R005257/1, and in part by the Engineering and Physical Sciences Research Council under Grant EP/R00711X/1 in the U.K. This article was presented in part at the IEEE International Communication Conference (ICC 2019). *(Corresponding author: Yue Gao.)*

Xiaolan Liu and Jiadong Yu are with the Department of Electronic Engineering and Computer Science, Queen Mary University of London, London E1 4NS, U.K. (e-mail: xiaolan.liu@qmul.ac.uk; jiadong.yu@qmul.ac.uk).

Jian Wang is with the School of Electronic Science and Engineering, Nanjing University, Nanjing 210023, China (e-mail: wangjnju@nju.edu.cn).

Yue Gao is with the Institute for Communication Systems, University of Surrey, Guildford GU2 7XH, U.K. (email: yue.gao@ieee.org).

Digital Object Identifier 10.1109/JIOT.2020.2970110

2327-4662 © 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: HEFEI UNIVERSITY OF TECHNOLOGY. Downloaded on April 18,2021 at 08:22:51 UTC from IEEE Xplore. Restrictions apply.

into different clusters could be an effective solution, which deals with the IoT users separately according to their different demands on computation offloading in different clusters. This can be achieved by using the clustering algorithms which classify data points into different clusters, and the data points in the same cluster have similar characteristics while they are different from data points in the other clusters [10].

Therefore, designing an optimal computation task offloading scheme, i.e., the task is executed at a local IoT user or edge server, is the key challenge in the IoT edge computing networks. It has been mostly discussed in the context of mobile-edge computing (MEC), in which the mobile user makes a binary decision to either offload the computation tasks to the edge device or not [11]. The conventional optimization methods, such as Lyapunov optimization and convex optimization techniques, were adopted to solve the cost (e.g., energy consumption and task execution latency) minimization problem when exploring an optimal computation offloading scheme [12]. However, they cannot make optimal decisions based on the dynamic environment. Note that designing the computation task offloading scheme while interacting with the environment can be modeled as a Markov decision process (MDP). Reinforcement learning is an effective method to solve an MDP problem without requiring the priori knowledge of environment statistics [13]. Moreover, to break the curse of dimensionality, deep reinforcement learning (DRL) is explored in the MEC system [14].

B. Related Work

In IoT networks, the emerging of edge computing has brought many research challenges, such as dynamic computation offloading scheme design, resource (e.g., computing resource, spectrum resource) allocation, and transmit power control. Moreover, they cannot be solved independently, such as designing an optimal computation offloading scheme has to consider the limited resource of the gateway and the transmit power of the users. Computation task offloading scheme has been investigated to access to multiresources efficiently in edge computing networks, especially in the MEC system [15]–[17]. Energy consumption and task execution latency are two main considerations when designing the optimal task offloading scheme. A joint optimization of task offloading scheduling and transmit power allocation scheme has been proposed in the MEC system with single mobile user [15], while joint optimizing radio and computational resource was considered with multiuser MEC system [16]. Liang *et al.* [18] optimized the number of offloading users by exploring the computation offloading scheme for multiple users in the MEC system. The conventional optimization tools were used to solve the task offloading problem in [12], however, which cannot deal with the environment dynamics.

Moreover, a lot of IoT users contending for the finite spectrum and computation resource of the gateway makes it even more hard to design the computation offloading schemes. The problem of joint radio-and-computation resource allocation in multiuser MEC systems in the presence of input/output interference was studied in [18]. In [19], an offloading priority function which defined the priorities for users according to their channel gains and local computing energy consumption was derived, and the optimal policy was proved to have a threshold-based structure with respect to this defined function. Therefore, the users were classified into different groups with different priorities above or below a given threshold perform complete and minimum offloading, respectively. Clustering algorithms, such as K-means [20], known to cluster data set into different clusters according to the characteristics of each data point, make the way for user clustering in the IoT networks.

Machine learning approaches, such as unsupervised learning, clustering algorithms, can extract the data structures from a large amount of data; and reinforcement learning, like Q-learning algorithm, can learn the optimal strategy by interacting with the environment. The increase of computation capacity at edge devices contributes to a new research area, called edge learning, which crosses and revolutionizes two disciplines: 1) wireless communication and 2) machine learning [21], [22]. Edge learning can be accomplished by leveraging the MEC platform. Some research works have been studied to apply machine learning techniques to optimize the communication and computation task offloading schemes in the MEC network [14], [23]. In [24], an In-Edge AI has been evaluated and could achieve near-optimal performance by investigating the scenarios of edge caching and computation offloading in MEC systems. An optimal task offloading scheme was learned through a deep Q-network (DQN) by maximizing the long-term utility performance without knowing *a priori* knowledge of network dynamics [25]. In [26], an intelligent resource allocation framework was developed to solve the complex resource allocation problem by proposing a Monte Carlo tree search (MCTS) method based on a multitask RL algorithm. An optimal computation offloading scheme was proposed for an IoT device with energy harvesting techniques to choose the MEC device and determine the computation offloading rate, with efficient resources, including spectrum and power resource allocation in [27].

However, computation task offloading schemes in ultra-dense IoT networks with edge computing has rarely been considered. Designing computation task offloading schemes for IoT networks, especially for the applications with massive IoT users, like smart farming, is necessary. Additionally, the optimal task offloading policy has been validated to have a threshold-based structure [19], we consider using clustering algorithms to group the IoT users into different user clusters as the initial step of our proposed task offloading scheme. We define the clustering feature by user priority which is defined as a tuple, including the distance from user to the edge server and the task offloading probability. Moreover, the distributed computation task offloading scheme is designed with minimizing the system cost, task execution latency, and energy consumption. In this article, we consider the weighted sum of task execution latency and energy consumption, which was rarely considered together before as the system cost. The design of the optimal computation offloading scheme is modeled as an MDP, where the objective is to minimize the long-term system cost while considering the environment dynamics. We use a deep neural network (DNN) to approximate the Q-function in Q-learning, and then a DQN-based task offloading algorithm is proposed to learn the optimal task offloading policy.

Authorized licensed use limited to: HEFEI UNIVERSITY OF TECHNOLOGY. Downloaded on April 18,2021 at 08:22:51 UTC from IEEE Xplore. Restrictions apply.

The major contributions of this article are stated as follows.

1) We first design a computation task offloading scheme for the ultradense IoT edge computing network via machine learning approaches, which is achieved by two steps: a) user clustering and b) distributed task offloading algorithms.
2) We first cluster the IoT users into different clusters with a K-means-based clustering algorithm according to the clustering feature and user priority. The clusters with the highest and lowest user priority are assigned as edge computing and local computing, respectively.
3) For IoT users in the other clusters, the problem of designing an optimal computation offloading policy is modeled as an MDP, where the objective is to minimize the long-term system cost while interacting with the dynamic environment. Moreover, a DQN-based computation offloading algorithm is proposed to learn the optimal computation offloading policy.
4) The numerical results show that the IoT users are grouped into different user clusters and the optimal cluster number is validated. Also, the DQN-based computation offloading algorithm is demonstrated to be superior to the other baseline schemes.

The remainder of this article is organized as follows. In Section II, the system model of computation task offloading in IoT edge computing networks is built. The problem of task offloading with minimizing the system cost is formulated in Section III. In Section IV, the centralized clustering algorithm is applied to group the IoT users into different user clusters. Then, a DQN-based optimal task offloading scheme is proposed in Section V. The numerical results are given in Section VI. Finally, the conclusions are drawn in Section VII.

II. System Model

As shown in Fig. 1, an IoT edge computing network is presented with a three-layer hierarchical architecture which consists of a cloud platform, multiple gateways, and a lot of IoT users, which includes many independent IoT networks. Each IoT network provides services for a large number of IoT users (i.e., different IoT devices), which is achieved by the gateway (i.e., the edge devices) collecting data from IoT users in its coverage area and processing them with its equipped edge server. However, the edge server has finite computation capacity and spectrum resource such that the gateway can only allow a limited number of devices to be accessed at the same time. Each IoT user generates computation tasks in different sizes continuously and it has limited computation capacity and battery power as well, so offloading part of their computation tasks to the gateway may improve the computation experience in terms of energy consumption and task execution latency.

As an example, a typical independent IoT network is considered in this article, and the location distribution of IoT users is modeled by a Poisson cluster process (PCP). From Fig. 1, computation tasks can be either executed locally at the IoT user or offloaded to the gateway and executed at the edge server. However, it is hard for the IoT users to make decisions on computation offloading since it cannot know the perfect

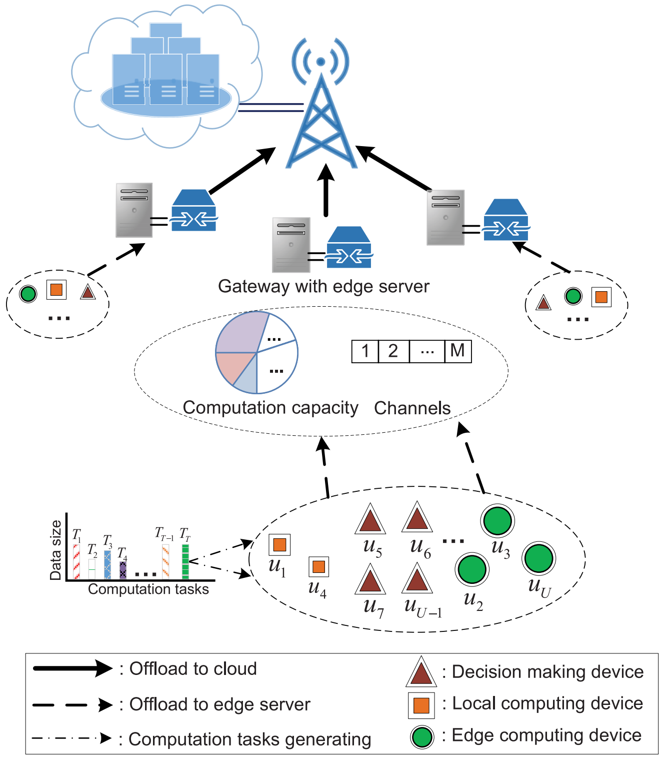

Fig. 1. System model of IoT networks with edge computing.

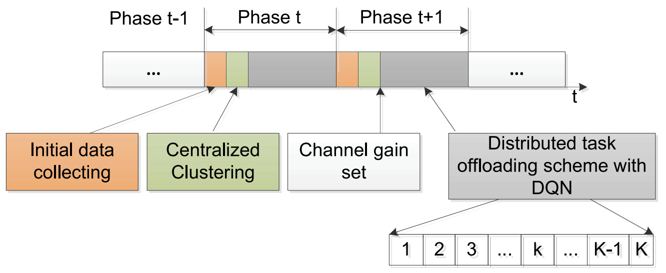

Fig. 2. Illustration of the computation offloading scheme with clustering and RL.

knowledge of the environment, like channel gains and other users' decisions. First, in this article, the centralized clustering algorithm is proposed to group the IoT users into different clusters according to user priorities, after that, the distributed computation offloading scheme with reinforcement learning is proposed for IoT users in the clusters that need more decisions. Hence, this computation offloading scheme has a time phase-based structure as presented in Fig. 2, and we discrete time horizon into time slots in each time phase t and each time slot is indexed by k.

In the considered IoT networks, the IoT users transmit their data to the gateway by accessing to a limited number of channels with the bandwidth of each channel denoted by B_w. Assuming each IoT user continuously generates independent computation tasks in different sizes. Each IoT user can store a limited number of tasks with the maximum number of tasks that cannot exceed T, hence the possible number of tasks stored at each IoT user in time slot k is denoted by $\mathcal{T}^k = \{1, \ldots, T\}$. The task generation is assumed to be an

Authorized licensed use limited to: HEFEI UNIVERSITY OF TECHNOLOGY. Downloaded on April 18,2021 at 08:22:51 UTC from IEEE Xplore. Restrictions apply.

TABLE I
PARAMETERS NOTATION

parameters	notation	parameters	notation
$\mathcal{U}$	user set	C_d^k, C_e^k	cost of task execution in local computing, edge computing
B_w	channel bandwidth	δ	penalty function of failed task execution
$\mathcal{T}^k$, t^k	task queue set, task queue state	E_d, E_s	energy consumption per CPU cycle of user, server
$\mathcal{I}^k$, I	task generation set, indicator	D_d, D_s	computation capacity of user, server
$\mathcal{M}^k$	task size set	$\mathcal{R}^k$, r^k	the remaining computation capacity set, the remaining computation capacity state
$\mathcal{O}^k$, O	users' decisions set, indicator	L_d^k,L_s^k and L_t^k	latency of local, edge task execution and task transmission
m_j^k	one possible task size	E_{cd}^k,E_{cs}^k	energy consumption of task execution in local, edge computing
R_e^k, β	system cost, weight factor	R, g^k and σ^2	transmission rate, channel gain and noise power
$\mathcal{P}^k$	transmit power set	f_d, f_s	CPU frequency of user,server
k,K	time slot, maximum time slot	$L_{th}{}^j, L_d{}^j$	task execution latency requirement and local task execution latency of j^{th} task
x_i	user priority	h, H	cluster number index, cluster number
$\mathcal{C}_h$, c_h	cluster h, cluster centroid of $\mathcal{C}_h$	SSE, SC_i	performance index of elbow method, Silhouette Coefficient of user u_i
G_h	channel gain set of $\mathcal{C}_h$	d_i, $\mathbf{Pr}_i$	distance from gateway to user, computation offloading probability of user

independent and identically distributed sequence of Bernoulli random variables with a common parameter $\tau^k \in [0, 1]$. The indicator of task generation is presented as $\mathcal{I}^k = \{0, 1\}$, where $I^k = 1$ indicates one task is generated with its task size following uniform distribution, denoted by a discrete set $\mathcal{M}^k = \{m_1, \dots, m_M\}$, otherwise, $I^k = 0$ means there is no task generated at current time slot. Here, $\tau^k = P_r\{I = 1\} = 1 - P_r\{I = 1\}$. Table I lists all the notations and descriptions of our considered system model.

III. Problem Formulation

The benefits of distributing computing are releasing the computation burden on the cloud server, shrinking the task execution latency and reducing the waste of spectrum resource. Edge computing is going to make a great achievement of these benefits when applied to the IoT networks. Moreover, the more and more powerful computation chips are deployed in IoT users, which makes it possible for local task execution. Therefore, the problem of solving the optimal computation offloading policy, that is, the task is executed locally or offloaded to the edge server, remains challenging. In this section, we first introduce two task computing modes and then calculate the system cost under each computing mode, respectively. Therefore, a typical IoT user is considered in an ultradense IoT network to learn the optimal computation offloading policy by minimizing the long-term system cost.

The design of the optimal computation task offloading scheme is to minimize the long-term system cost by choosing the best computing mode for each task generated by the IoT user, local computing mode or offloading computing mode. The possible decisions of each IoT user on computation task offloading in time slot k are $\mathcal{O}^k = \{1\} \cup \{0\}$, which indicates whether the task is offloaded or not. If the IoT user is selected to offload its computation task, its transmit power will be selected from a discrete set $\mathcal{P}^k = \{P_1, \dots, P_{\max}\}$. Note that the IoT user does not offload the computation task when $O^k = 0, O^k \in \mathcal{O}^k$, then the cost only contains local computation energy consumption and local task execution latency, and the transmit power is defined as $P_t^k = 0$ in this case. $O^k = 1, O^k \in \mathcal{O}^k$ indicates that the IoT user decides to offload computation task to the gateway, with the transmit power $P_t^k \in \mathcal{P}^k$. In both cases, the computation task is executed successfully, but there is a possibility that the task execution is failed, such as the task transmission might suffer from communication outage between the IoT user and the gateway, so we define a penalty function to illustrate this situation.

A. Local Computing Mode

We have $O^k = 0$ if the computation task is executed locally at the IoT user. We let f_d denote the fixed CPU frequency of the IoT user, which presents the number of CPU cycles required to compute one bit of input data. The energy consumption per CPU cycle is denoted by E_d. Then, $f_d E_d$ indicates the computing energy consumption per bit at the IoT user. The total energy consumption of one computation task at the IoT user, denoted by E_{cd}^k, is given by $E_{cd}^k = f_d E_d m_j^k$. Moreover, let D_d denote the computation capacity of the IoT user, which is measured by the number of CPU cycles per second. The remaining CPU resource of the IoT user is denoted by the remaining percentage of computation resource $\mathcal{R}^k = \{r_1, r_2, \dots, 1\}$. The local computing latency L_d at the IoT user is defined as $L_d^k = (f_d m_j^k)/D_d$. The energy consumption and the task execution latency are two main costs in the edge computing network. Then, we define the cost function of the local computing mode for the IoT user as

$$C_d^k = E_{cd}^k + \beta L_d^k \tag{1}$$

where β indicates the weight factor between the energy consumption and the task execution latency, which combines different types of functions with different units into a universal cost function.

B. Edge Computing Mode

Let g^k denote channel gain from the IoT user to the gateway, which independently picks a value from the channel gain state space G_h obtained from cluster $\mathcal{C}_h$ through user clustering. g^k is assumed to be constant during the offloading time epoch. P_t^k indicates the transmit power of the IoT user in time slot k, then the achievable transmission rate (bit/s) is denoted by

$$R^k = B_w \log_2\left(1 + \frac{P_t^k g^k}{\sigma^2}\right) \tag{2}$$

where σ^2 indicates the power of additive white Gaussian noise (AWGN). Then, the energy consumption of the IoT user

Authorized licensed use limited to: HEFEI UNIVERSITY OF TECHNOLOGY. Downloaded on April 18,2021 at 08:22:51 UTC from IEEE Xplore. Restrictions apply.

caused by the data transmission is indicated by P_t^k, and the transmission latency is denoted by $L_t^k = m_j^k/R^k$. Similarly, let f_s denote the computation frequency of the edge server, E_s denotes the energy consumption per CPU cycle at the edge server. D_s indicates the computation capacity of the gateway. The computation energy of the IoT user at the edge server is given by $E_{cs}^k = f_s E_s m_j^k$, and the computation latency is calculated as $L_s^k = (f_s m_j^k)/D_s$. Therefore, we can obtain the cost function of the edge computing mode, and it is presented as

$$C_e^k = E_{cs}^k + P_t^k * L_t^k + \beta\left(L_s^k + L_t^k\right). \tag{3}$$

As discussed above, the task execution is assumed to be successful in both cases. But there is a possibility that the task execution is failed, like the task transmission might suffer transmission outage. Then, we define a penalty function δ as the cost when the task execution fails. Therefore, the cost function is calculated in three cases as

$$C^k = \begin{cases} C_d^k, & \text{if local} \\ C_e^k, & \text{if edge, not failed} \\ \delta, & \text{if edge, failed.} \end{cases} \tag{4}$$

IV. Centralized Clustering Algorithms

A. User Priority Initialization

As mentioned above, an ultradense IoT network is considered with a large number of IoT users denoted by $\mathcal{U} = \{u_1, \ldots, u_i, \ldots, u_U\}$, and many IoT users are requesting for computing services from the resource-restricted edge server, but it is noted that those users have different priorities of task execution. In this section, a centralized user clustering method is investigated at the gateway to group the IoT users according to user priorities which are then assumed similar across users in the same cluster. Here, the user priority of each IoT user is defined by a tuple, including its distance d_i to the gateway and its computation offloading probability $\mathbf{Pr}_i$. We define the computation offloading probability of each IoT user as in Definition 1.

At the IoT user u_i, each task is generated with its requirement on task execution latency, if the local computing cannot meet its requirement, the task has to be offloaded to the gateway. The clustering algorithm has to reserve an initial process that is used for the gateway to initialize the user priority, which is shown in Fig. 2. It contains that the users calculate and send their computation offloading probabilities to the gateway, and the gateway records the distances from it to all the users. After that, the gateway performs the clustering algorithm to group the IoT users into a bunch of clusters based on their priorities. Then, the optimal distributed computation offloading scheme for users within the clusters (except for the clusters with the highest and lowest user priority) is designed and solved by the deep Q-network. The detailed process of the designed computation offloading scheme is shown in Fig. 2.

Definition 1: Considering a typical IoT user u_i as an example, each task is generated with its task execution latency requirement as ${L_{th}}^j$ and the task size as m_j. The task execution latency at the IoT user is calculated as ${L_d}^j = m_j/D_d$.

Algorithm 1 Priority-Driven Clustering Based on K-Means

```
1:  Input:
2:  number of observed tasks $J$, clusters $H$ and clustering
    feature vector, user priority $x_i$
3:  Initialization:
4:  Randomly initialize $H$ cluster centroids $c_1, \ldots, c_H$
5:  Repeat:
6:  for $i = 1 : U$ do
7:      for $h = 1 : H$ do
8:          Calculate cluster index $h_i$ for $x_i$ as:
            $h_i = \arg\min_h \|x_i - c_h\|_2^2$
9:      end for
10: end for
11: for $h = 1 : H$ do
12:     Update the cluster centroids by
        $c_h := \frac{\sum_{i=1}^{U} 1\{h_i=h\}x_i}{\sum_{i=1}^{U} 1\{h_i=h\}}$
13: end for
14: Until: No change of the cluster centroids
```

So the computation offloading probability is defined as

$$\mathbf{Pr} = Pr(L_d > L_{th}) = Pr\left(\sum_{j=1}^{\infty} {L_d}^j > \sum_{j=1}^{\infty} {L_{th}}^j\right). \tag{5}$$

Here, the generated tasks are independent, so the computation offloading probability can be approximately presented by the frequency of the user's tasks offloading. This is calculated in the initial process in the clustering algorithm from Fig. 2, and later it is improved and updated in each time phrase.

B. Priority-Driven Clustering Based on K-Means

As discussed above, we define the user priority of computation offloading for the IoT user u_i as x_i, which is indicated by the feature vector, including the distance d_i from user u_i to the gateway and computation offloading probability $\mathbf{Pr}_i$, that is, $x_i = \{d_i, \mathbf{Pr}_i\}$. Therefore, a priority-driven user clustering algorithm based on the standard K-means algorithm is proposed to partition IoT users into H clusters according to the priorities of IoT users by minimizing the sum of squared error (SSE)

$$\min_{\mathcal{C}_h, c_h} \sum_{h=1}^{H} \sum_{x_i \in \mathcal{C}_h} \|x_i - c_h\|_2^2 \tag{6}$$

where H is the initial number of clusters, $\mathcal{C}_h$ means the cluster h, and x_i is the user priority feature of user u_i. Also, the centroid of cluster c_h is calculated as

$$c_h = \frac{1}{|\mathcal{C}_h|} \sum_{x_i \in \mathcal{C}_h} x_i \tag{7}$$

where $|\cdot|$ is the cardinality and the detailed description is provided in Algorithm 1.

It is noted that the optimal cluster number H_{op} is hard to get. One good method to validate the number of clusters is called the elbow method. Its basic idea is to run K-means clustering on the data set for range values of cluster number H.

Authorized licensed use limited to: HEFEI UNIVERSITY OF TECHNOLOGY. Downloaded on April 18,2021 at 08:22:51 UTC from IEEE Xplore. Restrictions apply.

If the selected cluster number H' is smaller than the real cluster number H, the SSEs are reduced dramatically when the cluster number increases by 1. Inversely, the SSE does not have obvious changes. Then, the optimal cluster number is located at the turning point, known as the elbow. The performance index of the elbow method, i.e., SSE is defined as

$$\text{SSE} = \sum_{h=1}^{H'} \sum \text{dist}(x_i, c_h)^2. \tag{8}$$

However, the elbow method is sometimes ambiguous, the average silhouette method can be an alternative to validate the optimal cluster number jointly. This method defines a silhouette coefficient to measure the similarity of a user to its own cluster compared to its neighboring clusters, its silhouette coefficient is calculated by

$$SC_i = \frac{b_i - a_i}{\max\{e_i, b_i\}}, \text{ if } |\mathcal{C}_h| > 1 \tag{9}$$

where e_i indicates the mean distance between the user u_i and all the other users in the same cluster $\mathcal{C}_h$. b_i is the mean distance between the user u_i and all the users in its closest neighboring cluster $\mathcal{C}_l$ which is given by

$$\mathcal{C}_l = \arg\min_{\mathcal{C}_h} \frac{1}{q} \sum_{x_l \in \mathcal{C}_h} |x_l - x_i|^2 \tag{10}$$

where q indicates the number of users in the cluster $\mathcal{C}_h$.

Then, the average silhouette coefficient is obtained by calculating the mean of SC_i over all the users. The average silhouette coefficient is in the range $[-1, 1]$, the higher value means the cluster number is more appropriate for the clustering algorithm.

Noted that our proposed K-means algorithm is the first method to cluster IoT users by defining the user priority of users' computation offloading; other explorations could be done in the future with different features, such as radio propagation, usage, user mobility pattern, etc. Moreover, the user clustering algorithm is performed periodically based on the coverage of users' location changes and the monitored task types changes.

C. Complexity Analysis

Reminding that finding an optimal solution to the clustering problem for observations in N dimensions is an NP-hard problem. If the cluster number H and the dimension N are fixed, the computational complexity can be exactly calculated as $O(U^{NH+1})$, where U is the number of users that need to be clustered and N is the dimension of the cluster feature vector. It depends on the two main steps: 1) calculate the cluster index and 2) update the cluster centroids. There are many heuristic algorithms that can be used to find the optimal solution, like Lloyd's algorithm whose complexity is $\mathcal{O}(IUNH)$. Here, I is the iteration number until the algorithm is converged, which has $I = 2^{\Omega(\sqrt{(U)})}$ iterations in the worst case.

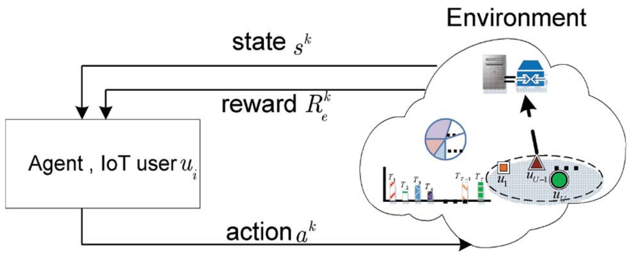

Fig. 3. Framework of reinforcement learning for IoT networks.

V. Computation Offloading Scheme via Deep Reinforcement Learning

After centralized user clustering, the IoT users are grouped into different clusters by the gateway according to their unique features, user priorities. In each cluster, each IoT user is assumed to have the same user priorities. Here, the cluster with the highest priority is designated as edge computing while the cluster with the lowest priority is assigned as local computing. For the other clusters, the optimal distributed computation offloading scheme solved by DRL is proposed in this section. First, the basic formulations of reinforcement learning are shown and then a computation offloading scheme is designed with modeling the computation offloading process as an MDP. Therefore, a deep Q-network-based computation offloading algorithm is proposed to learn the optimal computation offloading policy, which can deal with the MDP problem with large-state space.

A. Reinforcement Learning

As shown in Fig. 3, reinforcement learning is the process that the agents continuously learn optimal strategies by interacting with the environment. In this scenario, each IoT user is considered as an agent and everything else in the IoT network is made up of the environment. Here, due to performing user clustering, the scale of the computation offloading problem in the IoT network is reduced. The computation offloading scheme is assumed to have time slots structure as shown in Fig. 2. From Fig. 3, each agent, the IoT user u_i observes a state s^k from state space $\mathcal{S}$ at time epoch k, and then, an action a^k is taken from the action space $\mathcal{A}$, that is, decides which computing mode is taken by selecting different transmit powers based on the policy π. Therefore, the environment changes, new state s^{k+1} is observed and a reward R_e^k is obtained. In our scenario, the reward is designated as a negative reward, the system cost C^k, the weighted sum of energy consumption, and task execution latency. A reinforcement learning problem usually contains few key elements, including {Action, State, Reward, Environment}, and the environment is typically formulated as an MDP.

1) Action: At time step k, the action a^k for each IoT user in cluster $\mathcal{C}_h$ is the discrete transmit power, in which $P_t^k = 0$ indicates choosing local computing while others are actual transmit power in edge computing.[1] In our scenario, the action set is denoted by $\mathcal{A} = \{0, \mathcal{P}\}$.

[1]Assume that the transmit power is larger than 0 in edge computing.

Authorized licensed use limited to: HEFEI UNIVERSITY OF TECHNOLOGY. Downloaded on April 18,2021 at 08:22:51 UTC from IEEE Xplore. Restrictions apply.

2) State: The states of the agent represent the exploration information from the environment. Here, we use the channel gain g^k, task queue t^k stored at the IoT user, and its remaining computation capacity ratio r^k to present the exploration information. Hence, the state of the user in cluster $\mathcal{C}_h$ at the time step k is given as

$$s^k = \left\{g^k, t^k, r^k\right\} \tag{11}$$

where $s^k \in \mathcal{S}$, $g^k \in G_h$. G_h is the channel gain set in cluster $\mathcal{C}_h$ except the clusters that have the highest and lowest user priority, which is obtained after performing the centralized user clustering algorithm.

3) Reward: The function of the reward signal is to encourage the learning algorithm to reach the goal of the optimization problem. In this article, the negative reward is adopted to minimize the system cost while making the right decisions on computation offloading and power allocation. Here, the system cost is measured by the weighted sum of energy consumption and task execution latency as derived in (4) in the IoT network. To make it clear, we use reward shaping to make the reward more informative and accelerate the training. Therefore, the reward for each IoT user in the cluster $\mathcal{C}_h$ at the time step k is given by

$$R_e^k = \begin{cases} C_d^k, & \text{if local} \\ C_e^k, & \text{if edge, } 1-p \\ \delta, & \text{if edge, } p \end{cases} \tag{12}$$

where p is the failure rate of task transmission (considering the possibility of failed task transmission) in the edge computing mode, for simplicity, which is set as a fixed value in this article. δ is the penalty function when the task execution is failed.

Here, we denote the objective function as a negative reward, i.e., the system cost shown in (4). In addition, the state transition and system cost are stochastic, which can be modeled as an MDP, where the state transition probabilities and the cost depend only on the environment and the obtained policy. The transition probability $\mathbb{P} = (s^{k+1}, R_e^k | s^k, a^k)$ is defined as the transition from state s^k to s^{k+1} with the obtained reward R_e^k when the action a^k is taken according to the policy. Generally, the MDP problem is solved by finding an optimal policy that will maximize some cumulative function of the random rewards. In this case, the long-term expected cumulative negative reward, i.e., the system cost, over a finite horizon is given by

$$V(s, \pi) = \mathbb{E}_\pi\left[\sum_{k=1}^{K} \gamma^k R_e^k\right] \tag{13}$$

where s is the state and π is the policy, $\gamma \in [0, 1]$ is the discount factor and $\mathbb{E}$ indicates the statistical conditional expectation with transition probability $\mathbb{P}$.

Basically, the conventional solutions, like policy iteration and value iteration [28] belonging to dynamic programming can be used to solve the MDP optimization problem with the known state transition function. But it is hard for the agent to know the prior information of the state transition function, which is determined by the environment. Therefore, a model-free reinforcement learning approach is proposed to investigate this decision-making problem since the agent cannot make predictions about what the next state and cost will be before it takes each action.

In our scenario, the IoT user is trying to design an optimal computation offloading scheme according to some statistical information, such as the possible channel conditions, the possible remaining computation capacity percentage of the user, and the possible task queue, observing from the environment. Particularly, the user is exploring the optimal policy π^* that minimizes the long-term cost $V(s, \pi)$. This means for any given network state s, the optimal policy π^* can be obtained by

$$\pi^* = \arg\min_{\pi} \; V(s, \pi) \quad \forall s \in \mathcal{S}. \tag{14}$$

The problem of designing an optimal computation task offloading for the IoT user is a classic single-agent finite-horizon MDP problem. Then, we use the classic model-free reinforcement learning approach, Q-learning algorithm, to explore the optimal computation offloading policy by minimizing the long-term expected accumulated discounted cost, $V(s, \pi)$. We denote Q-value, $Q(s, a)$, as the expected accumulated discounted cost when taking an action $a^k \in \mathcal{A}$ following a policy π for a given state–action pair (s, a).

Thus, we define the action–value function $Q(s, a)$ as

$$Q(s, a) = \mathbb{E}_\pi\left[R_e^{k+1} + \gamma Q_\pi\left(s^{k+1},\ a^{k+1}\right) | s^k = s, a^k = a\right]. \tag{15}$$

In our proposed algorithm, $Q(s, a)$ indicates the value calculated from the cost function (4) for any given state s and action a stored in the Q-table which is built up to save all the possible accumulative discounted cost. The Q-value is updated during the time epoch if the new Q-value is smaller than the current Q-value. The $Q(s, a)$ is updated incrementally based on the current cost function R_e^k and the discounted Q-value $Q(s^{k+1}, a)$ $\forall a \in \mathcal{A}$ in the next time epoch. This is achieved by the one-step Q-update equation

$$\begin{aligned} Q\left(s^k, a^k\right) \leftarrow{} & (1-\alpha) \cdot Q\left(s^k, a^k\right) \\ & + \alpha\left(R_e^k + \gamma \cdot \min_a Q\left(s^{k+1}, a\right)\right) \end{aligned} \tag{16}$$

where R_e^k is the cost obtained in the current state, and α is the learning rate $(0 < \alpha \leq 1)$. Q-learning is a model-free and off-policy solution to solve an MDP, in each time slot, we calculate the Q-value in the next step with all the possible actions that it can take, then choose the minimum Q-value and record the corresponding action.

Moreover, to explore the unknown states instead of trusting the learned values of $Q(s, a)$ completely, the ϵ-greedy approach is used in the Q-learning algorithm, where the agent picks a random action with small probability ϵ, or with $1-\epsilon$ it chooses an action that minimizes the $Q(s^{k+1}, a)$ as shown in (16) in each time epoch.

B. Optimality and Approximation

The agent in the reinforcement learning algorithm aims to solve sequential decision-making problems by learning an optimal policy. In practice, the requirement for Q-learning to

Authorized licensed use limited to: HEFEI UNIVERSITY OF TECHNOLOGY. Downloaded on April 18,2021 at 08:22:51 UTC from IEEE Xplore. Restrictions apply.

obtain the correct convergence performance is that all the state action pairs $Q(s, a)$ continue to be updated. Moreover, if we explore the policy infinitely, the Q-value $Q(s, a)$ has been validated to converge with possibility 1 to $Q^*(s, a)$, which is given by

$$\lim_{n\to\infty} \mathbb{P}_r\left(\left|Q^*(s,a) - Q(s,a)_n\right| \geq \varsigma\right) = 0 \tag{17}$$

where n is the index of the obtained sample and $Q^*(s, a)$ is the optimal Q-value while $Q(s, a)_n$ is one of the obtained samples. Therefore, Q-learning can identify an optimal action-selection policy based on infinite exploration time and a partly random policy for a finite MDP model.

C. Deep Reinforcement Learning

As mentioned above, the classical Q-learning algorithm can find the optimal policy when the state–action is small, and it depends on a Q-table to store the Q-values and has to look up for every state in the table. If the state–action space becomes huge, it takes a long time for the Q-table to converge. The reason is that the states will be visited infrequently and the corresponding Q-values are updated rarely for a large number of states. In our problem, multiple state dimensions and continuous environment variations generate a large-state space. To solve this problem, a neural network can be used to approximate the Q-function, which can predict the Q values based on the input (states), that is, once the weight vector ($\boldsymbol{\theta}$) is obtained through training, we can get the output Q values $Q(s^k, a^k)$ quickly. Specifically, DNN has achieved more success in solving this problem, known as the deep Q-network. The DNN can be applied to the larger scale problems, then it will be appropriate to address sophisticated mappings from states to the desired Q-values output.

The DQN follows the rule of neural network to update its weight vector $\boldsymbol{\theta}$ at each iteration by minimizing the loss function, which is defined as

$$\text{Loss}(\boldsymbol{\theta}) = E\left[\left(Q'\left(s^k, a^k\right) - Q_{\text{target}}\left(s^k, a^k; \theta\right)\right)^2\right] \tag{18}$$

where $Q_{\text{target}}(s^k, a^k; \theta)$ is the target output Q-value, and the current Q-value $Q'(s^k, a^k)$ is the prediction which is given as

$$Q'\left(s^k, a^k\right) = R_e^k + \min_{a\in\mathcal{A}} Q\left(s^k, a, \theta\right) \tag{19}$$

where R_e^k is the corresponding negative reward, i.e., the system cost.

Here, the agent uses a replay memory with finite size to store state transitions $(s^k, a^k, s^{k+1}, a^{k+1}, R_e^k)$. We use the experience replay technique to sample a mini-batch from the memory pool, which is then used to train the DQN in the direction of minimizing (18). The detailed process of training the DQN-based computation task offloading is presented in Algorithm 2. After training, given an initial state, the IoT user selects an action that has the minimum estimated $Q(s^k, a, \theta^*)$. Hence, the test algorithm is proposed as a DQN-based computation task offloading scheme shown in Algorithm 3.

Algorithm 2 DQN Training for Computation Task Offloading

```
Initialization
    Initialize the size of replay memory and the mini-batch,
    Initialize Q-network with random weight vector $\boldsymbol{\theta}$,
    Initialize parameters: discount factor $\gamma$, learning rate $\alpha$,
    exploration rate $\epsilon$, and the channel gain set $G_h$ in the
    clusters $\mathcal{C}_h$
Procedure
 1: while $i' \leq iteration$ do
 2:     Observe initial state $s^k = (g^k, t^k, r^k)$
 3:     Select an action $a^k$ according to $\epsilon$-greedy policy.
 4:     After taking action $a^k$, calculate the cost $R_e^k$ by (12),
        and new state $s^{k+1} = (g^{k+1}, t^{k+1}, r^{k+1})$ is observed.
 5:     Store the state transition $(s^k, a^k, R_e^k, s^{k+1})$in the replay
        memory
 6:     Randomly sample a mini-batch of transitions from the
        experience pool
 7:     Update weight vector $\boldsymbol{\theta}$ by minimizing (18)
 8:     Every C steps update the Q-network
 9:     Update time epoch:$k = k + 1$
10: end while
```

Algorithm 3 DQN-Based Computation Task Offloading Scheme

```
Initialization
    Given an initial state $s^1 = (g^1, t^1, r^1)$
1: while $t^k > 0$ do
2:     Select an action $a^k = \min_{a\in\mathcal{A}} Q(s^k, a, \theta^*)$.
3:     State $s^{k+1} = (g^{k+1}, t^{k+1}, r^{k+1})$ is observed.
4: end while
```

D. Complexity Analysis

In this section, the complexity of the proposed computation task offloading algorithm is analyzed, which is mainly determined by the training algorithm shown in Algorithm 2. For Q-learning algorithm, its time complexity depends on the episodes I and the steps K in each episode, denoted as $O(IK)$. In the deep Q-network, DNN is used to approximate the Q-value function in the Q-learning algorithm. Hence, its time complexity mainly comes from training the DNN, which can be calculated as $O(H_i \cdot H_1 + H_1 \cdot H_2 + \cdots + H_n \cdot H_o)$. Moreover, how many times required for training this DNN relies on the total episodes I and the replay memory size D, denoted by $\mathcal{W} = \lfloor(I/D)\rfloor$. Here, the input layer of the DNN is determined by the batch size $\mathcal{B}$ and state space $\mathcal{S}$, i.e., $H_i = \mathcal{BS}$, while the output layer depends on the action space $\mathcal{A}$. Therefore, the time complexity of the proposed algorithm is $O(W \times (\mathcal{BS} \cdot H_1 + H_1 \cdot H_2 + H_2 \cdot H_3 + H_3 \cdot \mathcal{A}))$

VI. Numerical Results

In this section, the designed computation task offloading scheme is analyzed via the simulations. The scheme is achieved by user clustering and DRL. Here, the user clustering is based on K-means clustering algorithms according to users' unique characteristics and user priorities. After performing

Authorized licensed use limited to: HEFEI UNIVERSITY OF TECHNOLOGY. Downloaded on April 18,2021 at 08:22:51 UTC from IEEE Xplore. Restrictions apply.

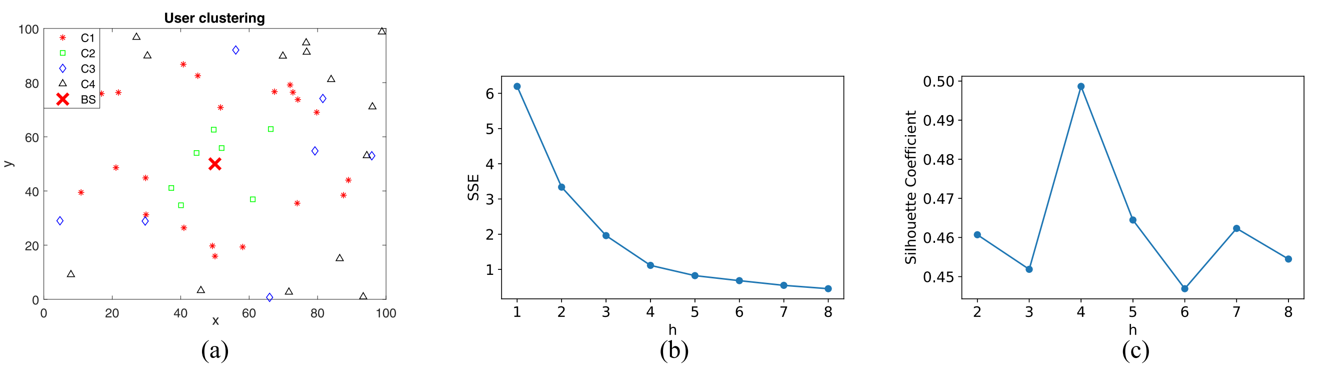

Fig. 4. *K*-means-based user clustering and the optimal cluster number validation. (a) *K*-means based user clustering. (b) Optimal cluster number by SSE. (c) Optimal cluster number by SC.

user clustering, the optimal distributed computation offloading policy is learned with DQN for the clusters without the highest and lowest user priority.

A. *Parameters Setup*

In our simulations, the distribution of the IoT users' locations is modeled by the PCP with $\lambda = 50$. The small-scale fading of the channels between the users and gateway is set as the Rayleigh fading with the parameter $B = 3$, and the path-loss parameter is $a = 2.5$. The input cluster number of the *K*-means user clustering algorithm is 4, the result of user clustering and the validation of the optimal cluster number are shown in Fig. 4.

For the DQN-based computation offloading algorithm, the DNN with three fully connected feedforward hidden layers are used to approximate the *Q*-function, the neurons of each hidden layer are 256, 256, and 512, respectively. The activation function of the first two hidden layers is rectified linear units (ReLUs), and the final hidden layer employs the tanh function as the activation function. The replay memory is set to have the capacity of 1000 recent transitions due to the dramatically changing environment, and the mini-batch size is 300. The action exploration is following the ϵ-greedy policy with $\epsilon = 0.9$, and its linearly decreasing with the iteration number. The channel gain states between the IoT user and the gateway are set as the channel gain set obtained in the clustering algorithm in the considered cluster $\mathcal{C}_h$. The other simulation parameters of the proposed DQN-based computation offloading algorithm are shown in Table II.

B. *DQN-Based Computation Offloading Scheme*

In this section, the performance of the optimal distributed task offloading policy learned by the DQN-based task offloading algorithm is quantified. For comparison, we also provide another three baselines: 1) local computing; 2) edge computing; and 3) greedy scheme, which are defined as follows.

1) *Local Computing:* The computation task is locally executed at the IoT user in each time slot whatever size of the task is generated.
2) *Edge Computing:* All the computation tasks generated at the IoT user are offloaded to the edge device and then processed at the edge server.

TABLE II
SIMULATION PARAMETERS

Parameters	Values
γ, α	0.9, 0.5
B_w, σ	$10^6 Hz, -174 + 10log_{10}B_w$
Channel gain states	G_h
Task queue	$\mathcal{T} = (0, 1, ..., 19)$
Task size	$\mathcal{M} = (10, ..., 30)Kbits$
remaining computation resource	$\mathcal{R} = (0, ..., 1)$
$f_s = f_d$	$600\ cycles/bit$
E_s	$10^{-8}\ J\ per\ CPU\ cycle$
E_d	$10^{-7}\ J\ per\ CPU\ cycle$
Ds, D_d	$4GHz, 500MHz$
β	100

3) *Greedy Scheme:* When the task queue is not empty and the remaining computation resource of the user is still enough, the IoT user decides to execute the task locally or offload it to the gateway while minimizing the system cost, that is, min $\{C_d^k, C_e^k\}$.

After performing user clustering, the IoT users are grouped into different clusters with different user priorities. A typical IoT user in the cluster $\mathcal{C}_h$ (except the cluster with the highest and lowest user priority) is considered as an example to learn the optimal computation offloading policy based on Algorithm 2.

First, the convergence performance of the DQN-based computation offloading algorithm is validated. We set the parameter of task generation as $\tau^k = 0.5$. From Fig. 5, we can observe that the loss is converged to a stable value by simulating the loss function shown in (18). Based on the convergence performance of the proposed DQN-based computation offloading algorithm, the following simulation results are analyzed from the trained DQN running for 2×10^4.

After the training in Algorithm 2 is finished, the computation offloading scheme is tested through Algorithm 3. We initialize the task queue as 15 and the terminal state is set as the task queue equalling to zero for the first time. The performance of the cumulative system cost C over the experienced time slots is analyzed in Fig. 6. The comparisons of the cumulative system cost among local computing, edge computing, greedy scheme, and DQN-based scheme over the time slots are shown in Fig. 6. The proposed DQN-based computation offloading scheme has a lower cumulative system cost

Authorized licensed use limited to: HEFEI UNIVERSITY OF TECHNOLOGY. Downloaded on April 18,2021 at 08:22:51 UTC from IEEE Xplore. Restrictions apply.

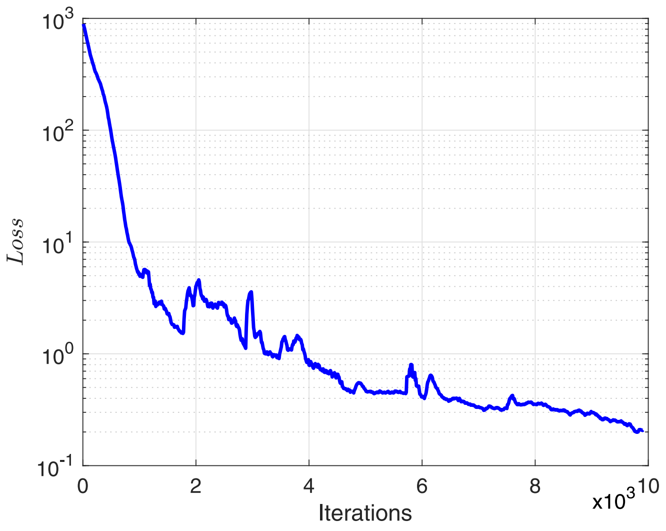

Fig. 5. Convergence performance of the proposed DQN-based computation offloading algorithm measured by loss function. The weight factor $\beta = 100$.

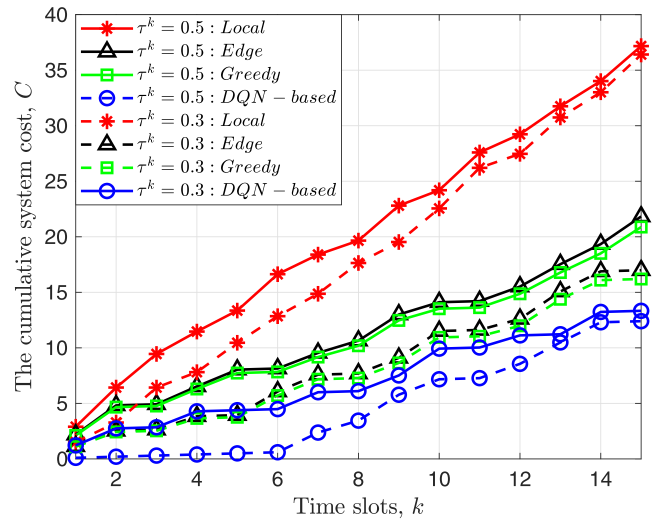

Fig. 6. Performance comparison of cumulative system cost C with the proposed DQN-based computation offloading scheme and the other three baseline schemes versus the time slots. $\beta = 100$.

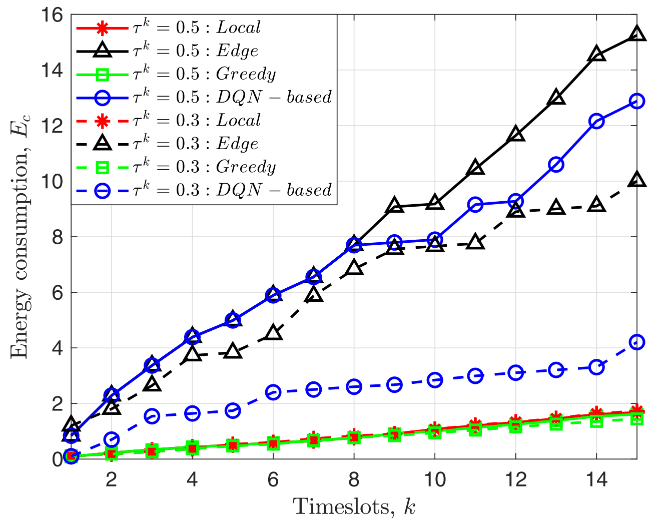

Fig. 7. Performance comparison of cumulative energy consumption P_c with the proposed DQN-based computation offloading scheme and the other three baselines versus the time slots. $\beta = 100$.

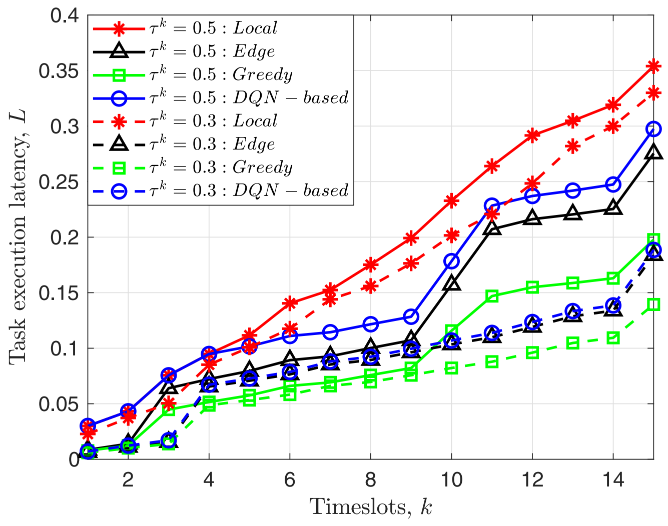

Fig. 8. Performance comparison of cumulative task execution latency L with the proposed DQN-based computation offloading scheme and the other three baselines versus the time slots. $\beta = 100$.

than the other three schemes. The reason is that our proposed DQN-based scheme can learn the optimal policy to choose the task computing mode by minimizing the long-term system cost while the greedy scheme only considers the current minimum system cost. Here, the higher task generation probability, $\tau^k = 0.5$, means more computation tasks are generated and executed, hence the system cost is higher.

Fig. 7 shows the comparisons of energy consumption E_c among edge computing, local computing, greedy scheme, and our proposed DQN-based scheme over the time slots. We can see that the edge computing has the highest energy consumption, this is because the IoT user consumes transmit power to offload the computation tasks to the gateway. But for the local computing mode, it consumes the least energy since all the energy consumption only comes from the task execution. In our proposed scheme, the decisions on computation task offloading are made by observing environment information. Compared to local computing, it has higher energy consumption due to its optimal policy including edge computing. Here, the greedy scheme has lower energy consumption since it considers the current minimum cost.

The performance of the task execution latency L among the three baseline schemes and our proposed DQN-based scheme is shown in Fig. 8. Local computing has the highest execution latency since the computation ability of the IoT user is much weaker than the edge server, i.e., the gateway. Specifically, the task with a large-task size costs a large amount of time to be executed locally. Correspondingly, the computation task can be executed more quickly when it is offloaded to the gateway and executed at the edge server. Similarly, the optimal policy learned by our proposed DQN-based scheme includes both actions: local computing and edge computing, hence it has neutral performance of task execution latency.

Fig. 9 presents the performance comparison of average cost under different edge server computation capacity D_s. We can

Authorized licensed use limited to: HEFEI UNIVERSITY OF TECHNOLOGY. Downloaded on April 18,2021 at 08:22:51 UTC from IEEE Xplore. Restrictions apply.

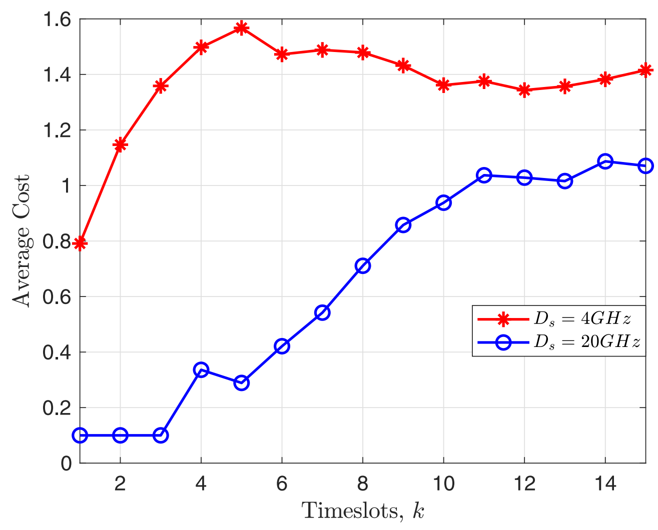

Fig. 9. Performance comparison of average cost under different edge server computation capacity versus time slots k. $\beta = 50$.

observe that the average cost is smaller when the edge server has a larger computation capacity $D_s = 20$ GHz compared to $D_s = 4$ GHz. This is because more powerful computation capacity can provide faster task execution, that is, smaller task execution latency.

VII. Conclusion

In this article, we designed the optimal computation task offloading schemes for ultradense IoT edge computing networks by considering the statistics of environment information, including the time-varying channel conditions, the dynamic task queue and the remaining computation capacity of the IoT users via machine learning approaches. The computation offloading scheme was developed by two steps: 1) centralized user clustering and 2) distributed computation offloading scheme. The centralized user clustering was achieved by using the K-means clustering algorithm with user priorities as the clustering feature. With user clustering, the clusters with the highest and lowest user priority were assigned as edge computing and local computing, respectively. For a typical user in any other clusters, an optimal distributed computation offloading policy was learned by using DQN with minimizing the long-term system cost. The K-means user clustering algorithm has been demonstrated by the simulations and the optimal cluster number was validated. Moreover, the DQN-based computation offloading algorithm has been simulated, it has lower system cost compared to the other three baseline schemes and a neutral performance was obtained with separately considering energy consumption and task execution latency.

In this article, we deal with the massive computation offloading in IoT networks from the aspect of user clustering, and then a distributed computation offloading scheme is designed for users required special computation offloading decisions. We consider a specific user makes decisions on its computation task execution by interacting with the dynamic environment. In our next work, we will consider multiple users to make their computation offloading decisions together, which can be formulated as a stochastic game that makes it more suitable for practical scenarios.

References

[1] O. Hahm, E. Baccelli, H. Petersen, and N. Tsiftes, "Operating systems for low-end devices in the Internet of Things: A survey," *IEEE Internet Things J.*, vol. 3, no. 5, pp. 720–734, Oct. 2016.
[2] J. Dizdarević, F. Carpio, A. Jukan, and X. Masip-Bruin, "A survey of communication protocols for Internet of Things and related challenges of fog and cloud computing integration," *ACM Comput. Surveys*, vol. 51, no. 6, p. 116, Feb. 2019.
[3] A. Yousefpour *et al.*, "All one needs to know about fog computing and related edge computing paradigms: A complete survey," Sep. 2018. [Online]. Available: arXiv:1808.05283.
[4] L. Bittencourt *et al.*, "The Internet of Things, fog and cloud continuum: Integration and challenges," *Internet Things*, vols. 3–4, pp. 134–155, Oct. 2018.
[5] L. Liu, Z. Chang, and X. Guo, "Socially aware dynamic computation offloading scheme for fog computing system with energy harvesting devices," *IEEE Internet Things J.*, vol. 5, no. 3, pp. 1869–1879, Jun. 2018.
[6] L. Liu, Z. Chang, X. Guo, S. Mao, and T. Ristaniemi, "Multiobjective optimization for computation offloading in fog computing," *IEEE Internet Things J.*, vol. 5, no. 1, pp. 283–294, Feb. 2018.
[7] C.-H. Hong and B. Varghese, "Resource management in fog/edge computing: A survey," Sep. 2018. [Online]. Available: arXiv:1810.00305.
[8] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," Mar. 2017. [Online]. Available: arXiv:1702.05309.
[9] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "Mobile edge computing: Survey and research outlook," Jan. 2017. [Online]. Available: arXiv:1701.01090.
[10] S. Bugdary and S. Maymon, "Online clustering by penalized weighted GMM," Feb. 2019. [Online]. Available: arXiv:1902.02544.
[11] S. Bi and Y. J. Zhang, "Computation rate maximization for wireless powered mobile-edge computing with binary computation offloading," *IEEE Trans. Wireless Commun.*, vol. 17, no. 6, pp. 4177–4190, Jun. 2018.
[12] X. He, H. Xing, Y. Chen, and A. Nallanathan, "Energy-efficient mobile-edge computation offloading for applications with shared data," Sep. 2018. [Online]. Available: arXiv:1809.00966.
[13] J. Xu and S. Ren, "Online learning for offloading and autoscaling in renewable-powered mobile edge computing," in *Proc. Global Commun. Conf. (GLOBECOM)*, Washington, DC, USA, Feb. 2016, pp. 1–6.
[14] X. Chen, H. Zhang, C. Wu, S. Mao, Y. Ji, and M. Bennis, "Performance optimization in mobile-edge computing via deep reinforcement learning," Mar. 2018. [Online]. Available: arXiv:1804.00514.
[15] Y. Mao, J. Zhang, and K. B. Letaief, "Joint task offloading scheduling and transmit power allocation for mobile-edge computing systems," in *Proc. Wireless Commun. Netw. Conf. (WCNC)*, San Francisco, CA, USA, Jan. 2017, pp. 1–6.
[16] S. Sardellitti, G. Scutari, and S. Barbarossa, "Joint optimization of radio and computational resources for multicell mobile-edge computing," *IEEE Trans. Signal Inf. Process. Over Netw.*, vol. 1, no. 2, pp. 89–103, Jun. 2015.
[17] A. Al-Shuwaili and O. Simeone, "Energy-efficient resource allocation for mobile edge computing-based augmented reality applications," *IEEE Wireless Commun. Lett.*, vol. 6, no. 3, pp. 398–401, Jun. 2017.
[18] Z. Liang, Y. Liu, T.-M. Lok, and K. Huang, "Multiuser computation offloading and downloading for edge computing with virtualization," Nov. 2018. [Online]. Available: arXiv:1811.07517.
[19] C. You and K. Huang, "Multiuser resource allocation for mobile-edge computation offloading," in *Proc. Global Commun. Conf. (GLOBECOM)*, Washington, DC, USA, Dec. 2016, pp. 1–6.
[20] D. Borthakur, H. Dubey, N. Constant, L. Mahler, and K. Mankodiya, "Smart fog: Fog computing framework for unsupervised clustering analytics in wearable Internet of Things," in *Proc. Global Conf. Signal Inf. Process. (GlobalSIP)*, Montreal, QC, Canada, Nov. 2017, pp. 472–476.
[21] G. Zhu, D. Liu, Y. Du, C. You, J. Zhang, and K. Huang, "Towards an intelligent edge: Wireless communication meets machine learning," Sep. 2018. [Online]. Available: arXiv:1809.00343.
[22] Y. Du and K. Huang, "Fast analog transmission for high-mobility wireless data acquisition in edge learning," Jul. 2018. [Online]. Available: arXiv:1807.11250.

Authorized licensed use limited to: HEFEI UNIVERSITY OF TECHNOLOGY. Downloaded on April 18,2021 at 08:22:51 UTC from IEEE Xplore. Restrictions apply.

[23] C. Zhang, Z. Liu, B. Gu, K. Yamori, and Y. Tanaka, "A deep reinforcement learning based approach for cost-and energy-aware multi-flow mobile data offloading," *IEICE Trans. Commun.*, vol. E101.B, no. 7, pp. 1625–1634, Jan. 2018.

[24] X. Wang, Y. Han, C. Wang, Q. Zhao, X. Chen, and M. Chen, "In-edge AI: Intelligentizing mobile edge computing, caching and communication by federated learning," Sep. 2018. [Online]. Available: arXiv:1809.07857.

[25] X. Chen, H. Zhang, C. Wu, S. Mao, Y. Ji, and M. Bennis, "Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4005–4018, Jun. 2019.

[26] J. Chen, S. Chen, Q. Wang, B. Cao, G. Feng, and J. Hu, "iRAF: A deep reinforcement learning approach for collaborative mobile edge computing IoT networks," *IEEE Internet Things J.*, vol. 6, no. 4, pp. 7011–7024, Apr. 2019.

[27] M. Min, L. Xiao, Y. Chen, P. Cheng, D. Wu, and W. Zhuang, "Learning-based computation offloading for IoT devices with energy harvesting," *IEEE Trans. Veh. Technol.*, vol. 68, no. 2, pp. 1930–1941, Feb. 2019.

[28] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. New York, NY, USA: Wiley, 2014.

Xiaolan Liu (Student Member, IEEE) received the B.S. and M.S. degrees in communication engineering from Jilin University, Changchun, China. She is currently pursuing the Ph.D. degree with the Department of Electronic Engineering and Computer Science, Queen Mary University of London, London, U.K.

Her research interests include energy harvesting and machine learning for IoT networks with low-power communication technology, and millimeter wave communications.

Jiadong Yu (Student Member, IEEE) received the B.S. degree in communication engineering from Dalian Maritime University, Dalian, China, and the M.S. degree (with Distinction) in wireless communication from the University of Southampton, Southampton, U.K. She is currently pursuing the Ph.D. degree with the Department of Electronic Engineering and Computer Science, Queen Mary University of London, London, U.K.

Her research interests include compressive sensing and millimeter-wave communications.

Jian Wang (Member, IEEE) received the Ph.D. degree from Nanjing University, Nanjing, China, in 2006.

He is currently an Associate Professor with the School of Electronic Science and Engineering and the Institute of Space-Terrestrial Intelligent Networks, Nanjing University. His research interests include mobile *ad hoc* networks, spatial information network, satellite communications and networks, video coding, and transmission.

Yue Gao (Senior Member, IEEE) received the Ph.D. degree from the Queen Mary University of London (QMUL), London, U.K., in 2007.

He was a Lecturer, a Senior Lecturer, and a Reader of antennas and signal processing with QMUL. He is currently a Professor and a Chair of wireless communications with the Institute for Communication Systems, University of Surrey, Guildford, U.K. He has published over 180 peer-reviewed journal and conference papers, two patents, one book, and five book chapters. He focuses on developing fundamental research into practice in the interdisciplinary area of smart antennas, signal processing, spectrum sharing, millimeter wave, and Internet-of-Things technologies in mobile and satellite systems.

Prof. Gao was a co-recipient of the EU Horizon Prize Award on Collaborative Spectrum Sharing in 2016. He served as the Signal Processing for Communications Symposium Co-Chair for IEEE ICCC 2016, the Publicity Co-Chair for IEEE GLOBECOM 2016, the Cognitive Radio Symposium Co-Chair for IEEE GLOBECOM 2017, and the General Chair for IEEE WoWMoM and iWEM 2017. He is the Chair of the IEEE Technical Committee on Cognitive Networks and the IEEE Distinguished Lecturer of the Vehicular Technology Society. He is an Editor of the IEEE INTERNET OF THINGS JOURNAL, the IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY, and the IEEE TRANSACTIONS ON COGNITIVE NETWORKS. He is an Engineering and Physical Sciences Research Council Fellow from 2018 to 2023.

Authorized licensed use limited to: HEFEI UNIVERSITY OF TECHNOLOGY. Downloaded on April 18,2021 at 08:22:51 UTC from IEEE Xplore. Restrictions apply.