Dynamic Offloading for Multiuser Muti-CAP MEC Networks: A Deep Reinforcement Learning Approach

Chao Li, Junjuan Xia, Fagui Liu, Dong Li, Lisheng Fan, George K. Karagiannidis, *Fellow, IEEE*, and Arumugam Nallanathan, *Fellow, IEEE*

***Abstract*—In this paper, we study a multiuser mobile edge computing (MEC) network, where tasks from users can be partially offloaded to multiple computational access points (CAPs). We consider practical cases where task characteristics and computational capability at the CAPs may be time-varying, thus, creating a dynamic offloading problem. To deal with this problem, we first formulate it as a Markov decision process (MDP), and then introduce the state and action spaces. We further design a novel offloading strategy based on the deep Q network (DQN), where the users can dynamically fine-tune the offloading proportion in order to ensure the system performance measured by the latency and energy consumption. Simulation results are finally presented to verify the advantages of the proposed DQN-based offloading strategy over conventional ones.**

***Index Terms*—MEC, dynamic optimization problem, non-binary offloading, DQN.**

I. Introduction

In recent years, the research in wireless networks have gradually evolved from the pure communication to communication and computation [1]. Some practical examples include intelligent monitoring, intelligent transport, vehicular networking, etc. To support these computation-intensive services, cloud computing can be applied to compute the tasks on the cloud, at the cost of transmission and information leakage. To resolve this problem, mobile edge computing (MEC) has been proposed to assist computing the tasks by the near-by computational access points (CAPs) in the networks, which can significantly reduce the latency and energy consumption of both communication and computation [2].

A key in the design of MEC networks is the offloading strategy, which determines how many parts of the tasks will be computed by the CAPs. In this direction, the authors in [3] and [4] investigated time-invariant system environments, and adopted some numerical methods to acquire a static offloading strategy for multiuser or multi-CAP MEC networks. In practice, the system environments may be time-varying,

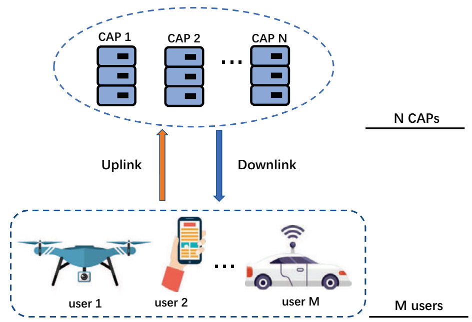


Fig. 1. A multiuser MEC network with M users and N CAPs.

which will impose a significant impact on the offloading strategy of MEC networks. For the time-varying wireless channels or varying arrival rate of computational tasks, some dynamic offloading strategies were proposed based on the game theory [5], [6]. As the above binary optimization problem [3]–[6] in MEC networks may be NP-hard, the authors in [7] further employed a deep Q network (DQN) to efficiently find a binary offloading strategy for MEC networks. However, To the best of our knowledge, there is no prior work on designing dynamic offloading strategy for multiuser multi-CAP MEC networks, by taking into account the time-varying task characteristics and computational capability at the CAPs, which motivates the work in this paper.

In this paper, we investigate a multiuser MEC network, where the tasks from users can be partially offloaded to multiple CAPs. We consider the practical environments where task characteristics and computational capability at the CAPs may be time-varying, thus, creating a dynamic offloading problem. To solve this problem, we first formulate it as a Markov decision process (MDP), and then introduce the state space and action space. We further design a novel offloading strategy based on the DQN, where the users can dynamically fine-tune the offloading proportion in order to ensure the system performance measured by the latency and energy consumption. Simulation results are finally presented to verify the advantages of the proposed DQN-based offloading strategy over conventional ones.

II. Preliminaries

A. System Model

Fig. 1 shows the system model of a multiuser MEC network, where there are M mobile users and N CAPs. The users have some computational tasks to be implemented, but they have limited computational capabilities. To facilitate the computation, these tasks can be partially offloaded to the N

C. Li, J. Xia, and L. Fan are all with the School of Computer Science, Guangzhou University, Guangzhou 510006, China. (email: lichaoacademic@e.gzhu.edu.cn, xiajunjuan@gzhu.edu.cn, lsfan@gzhu.edu.cn).

F. Liu is with School of Computer Science and Engineering, South China University of Technology, Guangzhou 510006, China (e-mail: fgliu@scut.edu.cn).

D. Li is with the Faculty of Information Technology, Macau University of Science and Technology, Taipa 999078, Macau, China (e-mail: dli@must.edu.mo).

G. K. Karagiannidis is with Aristotle University of Thessaloniki, Thessaloniki 54636, Greece (e-mail: geokarag@auth.gr).

A. Nallanathan is with the School of Electronic Engineering and Computer Science, Queen Mary University of London, London, U.K (e-mail: a.nallanathan@qmul.ac.uk).

CAPs, through the wireless links. Specifically, the sets of users and CAPs in the network are denoted by $\{u_m|1 \leq m \leq M\}$ and $\{\text{CAP}_n|1 \leq n \leq N\}$, respectively. At each time slot, user u_m has a computational task X_m, which has d_m number of bits and requires c_m CPU cycles to process. The computational capability at CAP_n is denoted by f_n, measured in CPU cycles per second. In practice, the task characteristics and computational capability at the CAPs may be time-varying, due to the change of environments.

In this paper, without loss of generality, we use a typical form of uniform distribution $\mathcal{U}(\bullet)$ to characterize the variation in the time domain[1]. Specifically, $c_m \sim \mathcal{U}(c_{min}, c_{max})$, where c_{min} and c_{max} are the minimum and maximum cycles, respectively; $d_m \sim \mathcal{U}(d_{min}, d_{max})$, where d_{min} and d_{max} are the minimum and maximum numbers of bits in the task X_m, respectively; and $f_n \sim \mathcal{U}(f_{min}, f_{max})$, with f_{min} and f_{max} being the minimum and maximum computational capabilities, respectively. To compute the task X_m, user u_m can offload $\rho_{m,n}$ portion of the task to CAP_n through the wireless u_m–CAP_n link, where $0 \leq \rho_{m,n} \leq 1$. After the computation is finished, all the CAPs return the computational results to the users through some dedicated feedback links. In summary, each time slot can be divided into three stages, i.e., task offloading, task computing and result feedback. The above three stages can be described as follows:

1) **Task offloading**: In this stage, some portions of the tasks at users are offloaded to the CAPs. Let $\boldsymbol{\rho}_m = [\rho_{m,0}, \rho_{m,1}, ..., \rho_{m,N}]$ denote the $1\times(N+1)$ offloading vector for the task X_m of the user u_m, where $\rho_{m,0}$ represents the proportion of X_m to be computed locally while $\rho_{m,n}$ denotes the portion to be computed by CAP_n. Based on $\boldsymbol{\rho}_m$, user u_m flexibly divides its task X_m into $N+1$ subtasks, and sends the associated N subtasks to the N CAPs in a sequential way.

2) **Task computing**: After collecting the subtasks in the first stage, the CAPs can compute the received subtasks in parallel, which can help reduce the latency in the computation.

3) **Result feedback**: After the task computation is finished, the CAPs can feedback the associated results to the users through some dedicated feedback channels. Once this stage is finished, one time slot has been used up for the communication and computation.

B. Latency and Energy Consumption

In this paper, we investigate the latency and energy consumption in the process of communication and computation in order to measure the system cost at each time slot [2]. At the first stage, the data rate of the wireless link from user u_m to CAP_n is

$$r_{m,n} = B\log_2\Big(1 + \frac{P_m|h_{m,n}|^2}{\sigma^2}\Big), \tag{1}$$

where B is the wireless bandwidth, P_m is the transmit power at user u_m, $h_{m,n} \sim \mathcal{CN}(0,\beta)$ is the instantaneous channel parameter of the $u_m - \text{CAP}_n$ link, σ^2 is the variance of the additive white Gaussian noise (AWGN) at CAP_n. From (1), we write the transmission latency and energy consumption as

$$l_{m,n} = \frac{\rho_{m,n}d_m}{r_{m,n}}, e_{m,n} = \frac{P_m\rho_{m,n}d_m}{r_{m,n}}. \tag{2}$$

Then, the latency of task offloading is given by

$$l_m = \sum_{n=1}^{N} l_{m,n} = \sum_{n=1}^{N}\frac{\rho_{m,n}d_m}{r_{m,n}}. \tag{3}$$

The largest l_m among M ones is used as the system offloading latency at the first stage,

$$L_1 = \max\{l_1, \cdots, l_M\}. \tag{4}$$

Similarly, the energy consumption of task offloading at the first stage is given by

$$E_1 = \sum_{m=1}^{M}\sum_{n=1}^{N} e_{m,n} = \sum_{m=1}^{M}\sum_{n=1}^{N}\frac{\rho_{m,n}d_m}{r_{m,n}}P_m. \tag{5}$$

Now we turn to compute the latency and energy consumption for the computation in the second stage. The local computational latency and energy consumption at user u_m are

$$l_{m,0} = \frac{\rho_{m,0}c_m}{f_0}, \quad e_{m,0} = \zeta_u\rho_{m,0}c_m f_0^2, \tag{6}$$

where f_0 is the local computational capability and ζ_u is the energy consumption coefficient of the CPU chip at the users. The computational latency and energy consumption at CAP_n are

$$l_n = \sum_{m=1}^{M}\frac{\rho_{m,n}c_m}{f_n}, \quad e_n = \sum_{m=1}^{M}\zeta_c\rho_{m,n}c_m f_n^2, \tag{7}$$

where ζ_c is the energy consumption coefficient of the CPU chip at the CAPs. From (6)-(7), we can write the latency and energy consumption at the second stage as,

$$L_2 = \max\{\max\{l_{1,0}, ..., l_{M,0}\}, \max\{l_1, ..., l_N\}\}, \tag{8}$$

$$\begin{aligned} E_2 &= \sum_{m=1}^{M} e_{m,0} + \sum_{n=1}^{N} e_n \\ &= \sum_{m=1}^{M}\zeta_u\rho_{m,0}c_m f_0^2 + \sum_{m=1}^{M}\sum_{n=1}^{N}\zeta_c\rho_{m,n}c_m f_n^2, \end{aligned} \tag{9}$$

where both the transmission latency and computational latency are considered in this paper. In some practical application scenarios, the number of bits in the feedback process is much smaller than that in the task. Hence, we can ignore the cost in the third stage. Accordingly, the total system latency and energy consumption at each time slot are summarized as

$$L_{total} = L_1 + L_2, \tag{10}$$

[1] When other kinds of distribution are used to characterize the time-varying characteristics, the proposed DQN-based optimization framework in this paper can be still applied to optimize the system offloading.

[2] As pointed out by many existing works in the literature such as [1-4], latency and energy consumption are two most significant performance metrics in the MEC networks. Specifically, latency is particularly important in the cases of video transmission, navigation, and control-orientated systems, while energy consumption attracts broad interests since the MEC nodes are energy-aware, especially when they have limited energy. Due to these reasons, we adopt the widely-used latency and energy consumption model in this work. Some other metrics such as pricing on the computation will be incorporated to measure the performance of MEC networks in future works.

$$E_{total} = E_1 + E_2. \tag{11}$$

From the equations above, we can find that the energy consumption is affected by both the latency and the associated power. However, the relationship between the latency and energy consumption is quite complicated, since the total energy consumption E_1 and E_2 are the sum of the individual one while the total latency L_1 and L_2 are the maximum of the individual one.

Besides investigating the individual latency and energy consumption, a linear combination of L_{total} and E_{total} can be used to measure the system performance,

$$\lambda L_{total} + (1-\lambda)E_{total}, \tag{12}$$

where $\lambda \in [0, 1]$ is a weight factor between the system latency and energy consumption. Note that it is reasonable to use the linear combination of latency and energy consumption as the system cost, in some MEC scenarios, due to the following reasons. Firstly, minimizing the linear combination can help reduce the latency and energy consumption. In particular, when the weight factor λ is 0 or 1, the linear combination degenerates into latency or energy consumption only. In this case, minimizing the linear combination directly leads to minimizing the latency and energy consumption. More importantly, the linear combination provides a flexible form of the system cost for the MEC networks, through adaptively adjusting the linear weight factor. Specifically, if the latency plays a more important role in the system cost, we can increase the value of λ, while we can reduce λ if the energy consumption becomes more important. Due to these reasons, the linear combination form of latency and energy consumption has been widely used to measure the system cost of MEC networks, in the existing works of the literature such as [8]–[10].

III. DQN-Based Offloading Strategy

As the offloading strategy determines how many portions of the tasks will be computed by the CAPs, it will affect the system latency and energy consumption significantly. Let $\boldsymbol{\pi} = [\boldsymbol{\rho}_1^T, \cdots, \boldsymbol{\rho}_M^T]$ be the offloading matrix. In this paper, we propose two criteria to optimize the offloading strategy. Specifically, criterion I optimizes the offloading strategy by minimizing the linear combination form of the system cost, at each time slot as,

$$\min_{\boldsymbol{\pi}} \Phi_{\mathrm{I}}(\boldsymbol{\pi}) = \lambda L_{total} + (1-\lambda)E_{total} \tag{13a}$$

$$s.t. \quad \rho_{m,0} + \sum_{n=1}^{N} \rho_{m,n} = 1, \quad \forall m \in \{1, 2, ..., M\}, \tag{13b}$$

$$0 \le \rho_{m,0} \le 1, \quad 0 \le \rho_{m,n} \le 1. \tag{13c}$$

In contrast, criterion II optimizes the offloading strategy by minimizing the energy consumption while meeting the requirement of the latency, at each time slot as,

$$\min_{\boldsymbol{\pi}} \Phi_{\mathrm{II}}(\boldsymbol{\pi}) = E_{total} \tag{14a}$$

$$s.t. \quad \rho_{m,0} + \sum_{n=1}^{N} \rho_{m,n} = 1, \quad \forall m \in \{1, 2, ..., M\}, \tag{14b}$$

$$L_{total} < L_{th}, \tag{14c}$$

$$0 \le \rho_{m,0} \le 1, \quad 0 \le \rho_{m,n} \le 1, \tag{14d}$$

where L_{th} is the latency threshold. For the decentralized implementation, criterion I should turn to minimize the locally linear combination form of cost while criterion II turns to minimize the local energy consumption with a given latency constraint, in order to obtain the local offloading strategy for each user.

Although the above two criteria in (13) and (14) can optimize the offloading strategy, it is however difficult to employ the conventional optimization method to solve an optimal $\boldsymbol{\pi}$ for each time slot. This is because that the optimization involves some complicated operations including the max operation and the associated derivative with respect to $\boldsymbol{\pi}$ is very complicated to solve. More importantly, the conventional optimization method performs the optimization at each individual time slot, and it cannot perform the optimization for the current time slot by exploiting the optimization result of the previous time slot, which however can act as an important reference for the current time slot. In time-varying environments, where the task characteristics and computational capability are varying, a learning based scheme should be developed to adaptively optimize the offloading strategy according to the dynamic environments.

In this paper, we adopt the DRL based algorithm to optimize the offloading strategy for the considered system. As one of the powerful decision-making algorithms in artificial intelligence field, the DRL performs the dynamic programming to achieve an excellent performance and effectiveness in tackling the optimization under dynamic environments. In the following, we will introduce the Markov decision process (MDP) and the implementation of deep Q-network (DQN), which are two important parts in the DRL based optimization framework.

A. Markov Decision Process (MDP)

The MDP is used to characterize the time-varying environments, which involve the state space and action space. As the time slot $t = 1, 2, ..., \infty$, we use $\mathcal{S} = \{\mathbf{s}_t | \mathbf{s}_t = [\mathbf{D}_t, \mathbf{C}_t, \mathbf{F}_t, \boldsymbol{\pi}_t]\}$ to denote the state space, where $\mathbf{D}_t = [d_1(t), \cdots, d_M(t)]$ and $\mathbf{C}_t = [c_1(t), \cdots, c_M(t)]$ are two $1 \times M$ task characteristic vectors at time slot t; $\mathbf{F}_t = [f_1, \cdots, f_N]$ is the $1 \times N$ computational capability vector at time slot t; and $\boldsymbol{\pi}_t$ is the $M \times N$ offloading matrix at time slot t. In addition, we use $\mathcal{A} = \{a_{m,n} \in \{1, -1, 0\} | 1 \le m \le M, 1 \le n \le N\}$ to denote the action space. For a given action $a_{m,n}$, we have

$$\begin{cases} \rho_{m,n} = \rho_{m,n} + \delta, & \rho_{m,0} = \rho_{m,0} - \delta \quad \text{If} \quad a_{m,n} = 1, \\ \rho_{m,n} = \rho_{m,n} - \delta, & \rho_{m,0} = \rho_{m,0} + \delta \quad \text{If} \quad a_{m,n} = -1, \\ \rho_{m,n} = \rho_{m,n}, & \rho_{m,0} = \rho_{m,0} \quad\quad\;\; \text{If} \quad a_{m,n} = 0, \end{cases} \tag{15}$$

where $\delta \in [0, 1)$ is an iterative gradient to fine-tune the offloading matrix. At the current time slot t, the environment state is denoted as $\mathbf{s}_t \in \mathcal{S}$ and then according to $\mathbf{s}_t$, the users execute an action noted by $a_t \in \mathcal{A}$. Considering the fairness among users, we can impose a constraint on the offloading ratio for the execution a_t, given by

$$|\rho_{m_1,0} - \rho_{m_2,0}| < \kappa, \forall m_1, m_2 \in \{1, \cdots, M\}, \tag{16}$$

where $\kappa \in [0, 1]$ is the fairness factor used to adjust the fairness among users. A smaller κ indicates a more strict constraint on the fairness. If (16) cannot hold, the agent will not execute the selection action a_t, and turn to choose another action to update the offloading matrix $\boldsymbol{\pi}_t$. When an element in the matrix $\boldsymbol{\pi}_t$ is updated, the offloading matrix transits from $\boldsymbol{\pi}_t$ to $\boldsymbol{\pi}_{t+1}$. Moreover, $\mathbf{D}_t$, $\mathbf{C}_t$ and $\mathbf{F}_t$ accordingly transit to $\mathbf{D}_{t+1}$, $\mathbf{C}_{t+1}$ and $\mathbf{F}_{t+1}$, respectively. Therefore, the environment state transits from $\mathbf{s}_t$ to $\mathbf{s}_{t+1}$ with a conditional probability $\mathcal{P}$, and meanwhile the users acquire the instant reward. We now discuss the reward function for the two criteria. For criterion I, we can design the reward function $\Psi_{\mathrm{I},t}$ as

$$\Psi_{\mathrm{I},t} = \lambda L_{total,t} + (1 - \lambda) E_{total,t}. \tag{17}$$

For criteria II, we can design the reward function $\Psi_{\mathrm{II},t}$ as

$$\Psi_{\mathrm{II},t} = \begin{cases} -\mu_1 & \text{if} \quad L_{total,t} \geq L_{th}, \\ \mu_2 & \text{if} \quad L_{total,t} < L_{th} \quad \text{and} \quad E_{total,t} - E_{total,t-1} < 0, \\ -\mu_2 & \text{if} \quad L_{total,t} < L_{th} \quad \text{and} \quad E_{total,t} - E_{total,t-1} \geq 0, \end{cases} \tag{18}$$

where μ_1 and μ_2 are two positive values with $\mu_1 > \mu_2$. Specifically, if the latency at the current time slot exceeds L_{th}, then the instant reward is $-\mu_1$. Otherwise, we need to observe the change in the energy consumption. The instant reward is μ_2 if the energy consumption decreases, while equal to $-\mu_2$ otherwise. According to the two instant reward functions, we can formulate the long-term expected average rewards for the network optimization under the strategy $\boldsymbol{\pi}_t$ as

$$V_i(\mathbf{s}_t = [\mathbf{D}_t, \mathbf{C}_t, \mathbf{F}_t, \boldsymbol{\pi}_t]) = \lim_{T \to \infty} \mathbb{E}[\sum_{t=1}^{T} \xi^t \Psi_{i,t}], \tag{19}$$

where $i \in \{\mathrm{I}, \mathrm{II}\}$ and $\xi \in (0, 1]$ is a discount factor to control the effect of historical data. We can try to find the optimal offloading strategy $\boldsymbol{\pi}^*$ by minimizing $V(\mathbf{s}_t)$ as

$$\boldsymbol{\pi}^* = \arg\min_{\boldsymbol{\pi}} V_i(\mathbf{s}_t), \quad \forall \mathbf{s}_t \in \mathcal{S}. \tag{20}$$

However, it is difficult for the users to know the conditional probability $\mathcal{P}$ for the state transition. Hence, we employ the following DQN-based approach to solve the offloading strategy for the considered MEC networks.

B. DQN-Based Solution

To show the effect of action on the strategy, we rewrite the state value function shown in (19) in a recursive form by using the state-action value function as

$$Q_i(\mathbf{s}_t, a_t) = \Psi_{i,t} + \xi \min_{a_{t+1}} Q_i(\mathbf{s}_{t+1}, a_{t+1}), \tag{21}$$

which is known as the Q function. In the conventional Q-learning algorithm, it is assumed that the number of states is limited, so that we can use a lookup table to record the state-action value pair. However, in this paper, due to the large number of environment states, we have to employ a deep neural network (DNN) to approximate the Q function. As shown in Fig. 2, at the current time slot t, we collect the current environment state $\mathbf{s}_t$ as the input data of the evaluation network, and the evaluation network outputs the value $Q_i(\mathbf{s}_t, a)$, for $a \in \mathcal{A}$. Then, we apply the ϵ-greedy policy to select an action a_t. Next, the users execute the action a_t, and then the state transits from $\mathbf{s}_t$ to another state $\mathbf{s}_{t+1}$ with the instant cost $\Psi_{i,t}$. Based on the cost $\Psi_{i,t}$, we update the parameters of the evaluation network. After many trials, the evaluation is trained to output an optimal value $Q_i(\mathbf{s}_t, a_t)$. Similar to the other deep learning networks, we use the mean square error based loss function to evaluate the training,

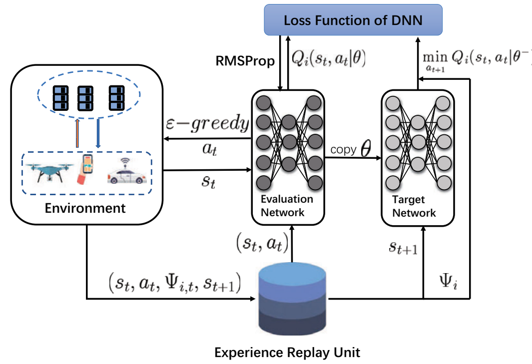


Fig. 2. The framework of the DQN-based offloading strategy.

$$Loss_t = \mathbb{E}[(Y_t - Q_i(\mathbf{s}_t, a_t|\theta))^2], \tag{22}$$

where θ is the parameter of the evaluation network, and Y_t is the target value that represents the optimization object of the evaluation network. Nevertheless, if we use the same DNN to obtain the target value, the optimization object will be changed with the parameter θ at each iteration. Therefore, we apply the target network which has the same structure with the evaluation network, except that the parameter update of the target network θ^- is t_{copy} time slots later than that of the evaluation network. For the two criteria in (13) and (14), we can calculate the target value Y_t as

$$Y_t = \Psi_{i,t} + \xi \min_{a_{t+1}} Q_i(\mathbf{s}_{t+1}, a_{t+1}|\theta^-). \tag{23}$$

In addition, the input data is independent in the supervised learning, while the observation data of the network is sequential. Motivated by this, we set an experience relay unit (ERU) in the framework of DQN. For the two criteria, we can collect the transition sample $(\mathbf{s}_t, a_t, \Psi_{i,t}, \mathbf{s}_{t+1})$ generated by the interaction between the environment and agent into the memory of ERU. During the training process, we randomly catch a mini-batch of transitions of the ERU memory at each iteration to break the dependence of data set. The proposed DQN-based offloading strategy is summarized in **Algorithm 1**.

C. Complexity Analysis

In this subsection, we provide some computational complexity analysis of the DQN used in this paper. As the complexity

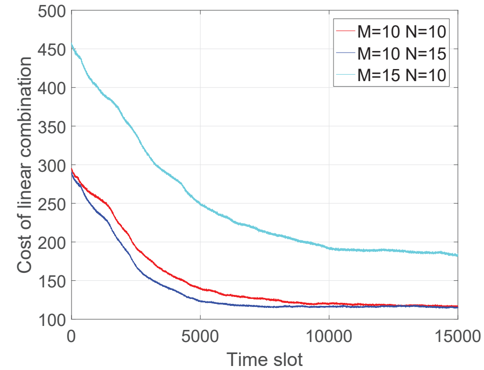
Algorithm 1: DQN-based offloading strategy

Clear up the memory of ERU
Randomly initialize the parameter of evaluation network θ and the parameter of target network θ^-, let $\theta = \theta^-$
Loop for each episode:
  Initialize $\mathbf{s}_0 \in \mathbf{S}$
  Loop for each time slot t:
    Choose a_t from $\mathbf{s}_t$ using policy derived from ϵ-greedy
    Carry out action a_t, and observe the system cost $\Psi_{i,t}$ and $\mathbf{s}_{t+1}$
    Store the transition sample $(\mathbf{s}_t, a_t, \Psi_{i,t}, \mathbf{s}_{t+1})$ in ERU
    Catch a minibatch of transitions from ERU
    $Y_t = \Psi_{i,t} + \xi \min_{a_{t+1}} Q_i(\mathbf{s}_{t+1}, a_{t+1}|\theta^-)$
    Execute RMSPropOptimizer to $(Y_t - Q_i(\mathbf{s}_t, a_t|\theta))^2$ respect to θ
    Every t_{copy} time slots reset $\theta^- = \theta$
    Let $\mathbf{s}_t = \mathbf{s}_{t+1}$
  end for
end for

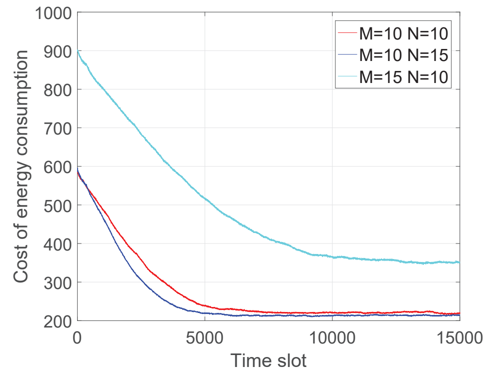
of ϵ-greedy policy based Q-learning algorithm is $O(T)$ [11] and the DQN framework is a combination of Q-learning and two DNNs with the identical structure, the computational complexity of the DQN comes from the matrix operation of DNNs. Since the DNNs of this paper employ the full-connection networks, the computational complexity of each training step is $O(\sum_{j=1}^{J} K_{j-1}K_j)$, where K_j represents the neural size of the j-th layer $(1 \le j \le J)$ among J layers. As the target value network only operates the forward propagation at each training step, the total computational complexity of the DQN algorithm in this paper is $O(3T\sum_{j=1}^{J} K_{j-1}K_j)$.

IV. Simulation Results

In this part, we demonstrate some simulation results to verify the effectiveness of the proposed DQN-based dynamic offloading strategy for the considered MEC networks. To simulate the time-varying environments, we set $c_m \sim \mathcal{U}(2\times 10^9, 3\times 10^9)$, $d_m \sim \mathcal{U}(2\times 10^8, 3\times 10^8)$, and $f_n \sim \mathcal{U}(5\times 10^9, 7\times 10^9)$. The local computational capability f_0 is set to $\mathcal{U}(1.5\times 10^9, 2\times 10^9)$. Moreover, the wireless bandwidth B is equal to 40MHz, whereas the average channel gain is set to 4, and the transmit SNR is set to 10dB. In further, the DQN network is implemented by using the well-known Tensorflow library on the Python platform, and there are three hidden layers in the network, with 16, 32 and 64 nodes in the layers in order. The Rectified Linear Unit (ReLU) is used as the activation function, and the 'RMSPropOptimizer' is employed as the optimizer to minimize the loss function in (22). Furthermore, the iterative gradient δ is 0.01, the factor ϵ in $\varepsilon-$ greedy policy is 0.8, and the sizes of the ERU and minibatch are set to 1000 samples and 200 samples, respectively. We reset $\theta^- = \theta$ every 200 time slots. In each simulation, we initialize $\rho_{m,0} = 1$ and $\rho_{m,n} = 0$ for $1 \le m \le M$, and repeat the experiment 100 times to calculate the average cost. If not specified, we set λ to 0.5 for criterion I and L_{th} = 1s for criterion II, and set $\kappa = 1$ for both criteria.



(a) Criterion I

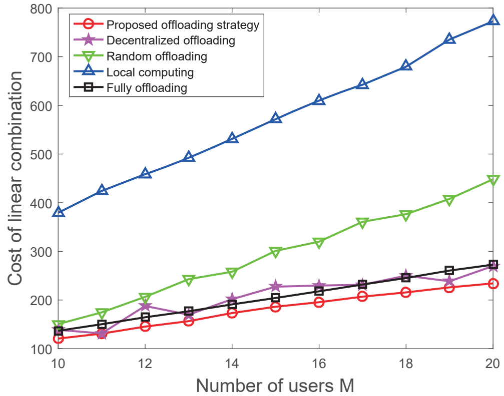


(b) Criterion II

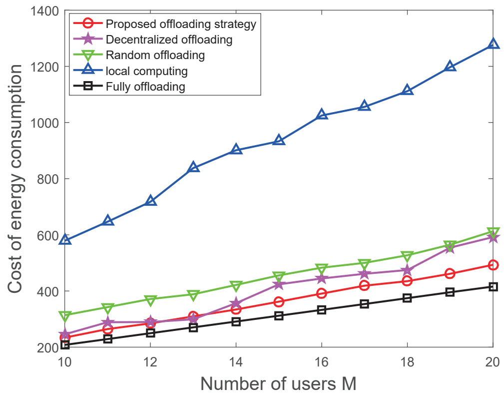
Fig. 3. Convergence of the proposed DQN approach.

Fig. 3 shows the training process of the proposed DQN approach for both criteria, where there are 15000 time slots and Fig. 3 (a) uses the linear combination to evaluate criterion I while Fig. 3 (b) employs energy consumption to measure criterion II. In particular, three groups of parameter setting are plotted with $(M,N) = (10,10)$, $(M,N) = (10,15)$, and $(M,N) = (15,10)$, respectively. As observed from this figure, we can find that for different numbers of users and CAPs, the curves of the proposed approach drop sharply with the increasing number of time slots, and the system cost becomes convergent after enough number of time slots. In particular, when $(M,N) = (10,10)$ and $(M,N) = (10,15)$, the system cost becomes convergent after about 5000 time slots, while for $(M,N) = (15,10)$, the system cost requires about 10000 time slots to be convergent. These results verify that after enough number of trials, the proposed DQN approach can find a suitable offloading strategy for the considered MEC network.

Fig. 4 illustrates the system cost of the proposed DQN-based offloading strategy versus the number of users for both criteria, where Fig. 4 (a) uses the linear combination to evaluate criterion I while Fig. 4 (b) employs energy consumption to measure criterion II. For comparison, we also plot the system cost of random offloading, local computing, fully offloading, and decentralized offloading in Fig. 4, where the decentralized offloading adopts the distributed DQN method without a centralized entity and each user can adaptively adjust its own offloading strategy via observing its local task characteristic and offloading vector by itself [12]. We can observe from this figure that for criterion I, the proposed DQN-



(a) Criterion I



(b) Criterion II

Fig. 4. System cost versus the number of users with $N = 15$.

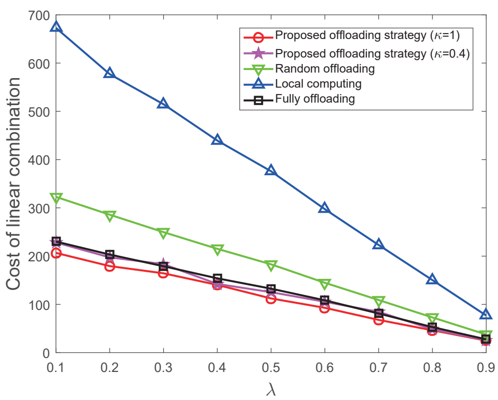


Fig. 5. Effect of the weight factor λ on the system cost of criterion I.

based offloading strategy outperforms the other four strategies for various values of M, indicating that the proposed strategy can effectively exploit the communication and computation resources. In contrast, the proposed strategy of criterion II outperforms the random, local computing and decentralized strategies, and it is assumed worse than the full offloading which however has a large transmission latency. The reason why the proposed strategy outperforms the decentralized one lies in that the latter cannot learn the global features by training separately without interacting with each other. In further, the system cost of the five strategies increases with a larger M, as more users cause an increasing number of tasks to the MEC network.

Fig. 5 demonstrates the system cost of criterion I with several offloading strategies versus the weight factor λ, where $M = 10$, $N = 10$ and λ varies from 0.1 to 0.9. We use two values of fairness factor with $\kappa = 0.4$ and 1, to see the impact of fairness on the system cost. As observed from Fig. 5, we can find that for various values of λ, the proposed strategy is superior to the other three strategies, which further verifies the effectiveness of the proposed strategy. Moreover, the system cost of the proposed strategy increases with a smaller κ, due to the sacrifice to protect the fairness among users. In further, the system cost of the five strategies decreases with a larger λ, as the energy consumption plays a more important role than latency in the linearly weighted cost, under the given parameter setting.

V. Conclusions

In this paper, we have investigated a multiuser muti-CAP MEC network, where task characteristics and computational capability at the CAPs are time-varying. To propose a dynamic offloading strategy, we have first formulated the dynamic offloading as MDP, and then introduced the state space and action space. We further designed a novel offloading strategy based on the DQN, where the users could dynamically fine-tune the offloading proportion in order to ensure the system performance measured by the latency and energy consumption. Simulation results were finally presented to verify the advantages of the proposed DQN-based offloading strategy over the conventional ones.

References

[1] J. Tang, H. Tang, X. Zhang, K. Cumanan, G. Chen, K. Wong, and J. Chambers, "Energy minimization in D2D-assisted cache-enabled internet of things: A deep reinforcement learning approach," *IEEE Trans. Ind. Informat.*, vol. 16, no. 8, pp. 5412–5423, 2020.

[2] Z. Liang, Y. Liu, T. Lok, and K. Huang, "Multiuser computation offloading and downloading for edge computing with virtualization," *IEEE Trans. Wireless Commun.*, vol. 18, no. 9, pp. 4298–4311, 2019.

[3] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Trans. Netw.*, vol. 24, no. 5, pp. 2795–2808, 2016.

[4] J. Yan, S. Bi, Y. J. Zhang, and M. Tao, "Optimal task offloading and resource allocation in mobile-edge computing with inter-user task dependency," *IEEE Trans. Wireless Commun.*, vol. 19, no. 1, pp. 235–250, 2020.

[5] H. Cao and J. Cai, "Distributed multiuser computation offloading for cloudlet-based mobile cloud computing: A game-theoretic machine learning approach," *IEEE Trans. Vehic. Tech.*, vol. 67, no. 1, pp. 752–764, 2018.

[6] L. Xiao, Y. Li, X. Huang, and X. Du, "Cloud-based malware detection game for mobile devices with offloading," *IEEE Trans. Mobile Comput.*, vol. 16, no. 10, pp. 2742–2750, 2017.

[7] S. Bi and Y. J. A. 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, 2017.

[8] Y. Pan, M. Chen, Z. Yang, N. Huang, and M. Shikh-Bahaei, "Energy-efficient NOMA-based mobile edge computing offloading," *IEEE Commun. Lett.*, vol. 23, no. 2, pp. 310–313, 2019.

[9] Z. Yang, C. Pan, J. Hou, and M. Shikh-Bahaei, "Efficient resource allocation for mobile-edge computing networks with NOMA: Completion time and energy minimization," *IEEE Trans. Commun.*, vol. 67, no. 11, pp. 7771–7784, 2019.

[10] A. Ndikumana, N. H. Tran, T. M. Ho, Z. Han, W. Saad, D. Niyato, and C. S. Hong, "Joint communication, computation, caching, and control in Big data multi-access edge computing," *IEEE Trans. Mob. Comput.*, vol. 19, no. 6, pp. 1359–1374, 2020.

[11] J. Chi, A.-Z. Zeyuan, B. Sebastien, and J. M. I., "Is Q-learning provably efficient?" in *Proceedings of NIPS*, 2018, pp. 4868 – 4878.

[12] B. Peng, G. Seco-Granados, E. Steinmetz, M. Frohle, and H. Wymeersch, "Decentralized scheduling for cooperative localization with deep reinforcement learning," *IEEE Trans. Vehic. Tech.*, vol. 68, no. 5, pp. 4295–4305, 2019.