Deep Reinforcement Learning based Task Offloading for MEC with User Mobility

Jiayi Huang, Zipeng Lu, Xiajun Huang, Jun Xu*

School of Computer and Electronic Information/School of Artificial Intelligence
Nanjing Normal University, Nanjing, Jiangsu 210023, China
jyhuang@nnu.edu.cn, zplu@nnu.edu.cn, xjhuang@nnu.edu.cn, junxu@njnu.edu.cn

***Abstract*—Mobile edge computing (MEC) enables various mobile devices (MDs) to offload computation-intensive tasks to nearby edge servers (ESs), which has the potential to improve energy efficiency and reduce task completion time. However, designing efficient task offloading strategies for multiple MDs and multiple ESs is non-trivial due to the mobilities of MDs, and the hybrid decisions for how many tasks and which ES to be offloaded. To deal with these challenges, we model the task offloading problem based on the deep reinforcement learning model. We propose a mobility-aware deep deterministic policy gradient (MO-DDPG) algorithm for task offloading. The MO-DDPG algorithm is based on the DDPG algorithm, however with a novel mobility-aware experience replay buffer, which assigns higher priorities to recent experiences, and samples hybrid experiences during training. Simulation results demonstrate that our MO-DDPG algorithm outperforms the baselines in terms of cost and convergence speed.**

***Index Terms*—MEC, user mobility, task offloading, DDPG**

I. Introduction

The popularity of smart devices has led to the proliferation of intelligent mobile applications, such as face recognition and interactive games. These applications generate computation-intensive tasks, which are hard to process on resource-constrained mobile devices (MDs). By deploying edge servers (ESs) in proximity to MDs, mobile edge computing (MEC) has the potential to relieve the computation burden of the MDs with task offloading [1]–[3]. Previous studies have demonstrated that efficient task offloading strategies can reduce task completion time and energy consumption [4]–[7]. However, these strategies with static MDs are not applicable in scenarios with user mobility.

Designing efficient task offloading strategies for MEC with user mobility is challenging [8]–[10]. Due to the mobility of MDs, the channel gain between an MD to an ES changes, which impacts the transmission rate of the MD, and further impacts the task completion time and energy efficiency. Moreover, the computing resource of an ES is shared by multiple MDs and is not unlimited. Therefore, the computing resources of ESs assigned to MDs are related to the task offloading decisions, which makes the task offloading problem more difficult. In addition, an efficient task offloading strategy should consider the pay of occupying the computation resource of the ES. Besides, the task offloading decision for each client is hybrid since it should decide how many tasks and which ES to be offloaded.

Considering these challenges, we adopt the deep reinforcement learning (DRL) method to investigate the task offloading problem for multiple moving MDs and multiple ESs. Although the previous research on task offloading can not be directly used in our scenario, the DRL-based methods are demonstrated to be efficient for resource allocation in edge computing [11]–[14]. Specifically, we adopt the deep deterministic policy gradient (DDPG) method to design a task offloading strategy, which learns the dynamic environment and the mutual impacts of different MDs via interaction with the environment. Our task offloading strategy should deal with both discrete and continuous decisions. However, the DDPG-based algorithm outputs continuous values. Therefore, we employ the Gumbel Softmax method to map continuous values to discrete values. Our main contributions are listed as follows,

- We investigate the task offloading problem with multiple MDs and multiple ESs with user mobility. Moreover, multiple MDs can share the limited computing resource of an ES, and the sharing of computing capacity of the ESs relates to the task offloading decisions of MDs.
- We propose a novel mobility-aware deep deterministic policy gradient (MO-DDPG) algorithm to decide how many tasks and which ES to offload tasks. The MO-DDPG algorithm assigns priorities to experiences considering user mobility.
- We evaluate our task offloading algorithm by comparing it with some baselines. Simulation results demonstrate the advantage of our algorithm in convergence speed and cost.

The rest of this paper is organized as follows. Section II

This work was supported in part by College Students' innovation and entrepreneurship training program, in part by National Natural Science Foundation of China under Grant 61901306, in part by the Research Start-up Fund of Nanjing Normal University, in part by the Dual-innovation Doctor Program of Jiangsu Province under Grant JSSCBS20220423, and in part by the Nanjing Science and Technology Innovation Project for Oversea researchers. (*corresponding author: Jun Xu)

presents the network model, mobility model and formulates the task offloading problem. Section III proposes the MO-DDPG algorithm. Section IV evaluates the performance of the task offloading algorithm. Section V concludes this paper.

II. System Model

We consider an MEC system with M MDs denoted as $\mathcal{M} = \{1, 2, \ldots, M\}$ and N ESs denoted as $\mathcal{N} = \{1, 2, \ldots, N\}$. An example of the system model is given in Fig. 1. Each ES is connected to a base station (BS) via wired connection. The MDs move randomly within a range. They may move from the coverage of one BS to that of another.

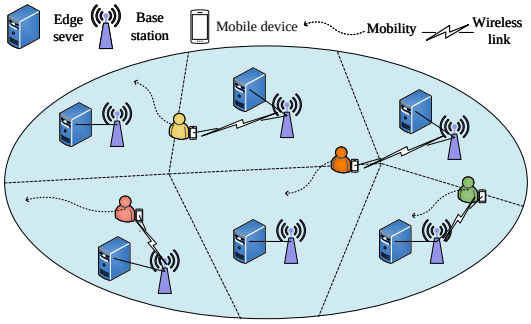

Fig. 1. An MEC system with multiple MDs and multiple ESs.

We consider a discrete system where time is divided into equal-length time slots, indexed by $\mathcal{T} = \{1, 2, \ldots, T\}$. At each time slot $t \in \mathcal{T}$, MD i ($i \in \mathcal{M}$) generates a task with size L_i^t in the range $[L_{i,min}, L_{i,max}]$, where $L_{i,min}$ and $L_{i,max}$ represent its minimum and maximum size, respectively.

For our task offloading problem, the decision maker should decide which ES the task of an MD should be offloaded to and what is the size of the task to be offloaded. Let $a_i^t = (\alpha_i^t, \lambda_i^t)$ denote the task offloading decision of MD i at time slot t, where α_i^t indicates which ES the task should be offloaded to, and λ_i^t is the ratio of the task to be offloaded.

Let F_j denote the computation capacity of ES j, and $\beta_j^i(t) \in [0, 1]$ denote the proportion of the computing resource of ES j allocated to MD i at time slot t. A reasonable way is to assign the computing resource of an ES equally to the MDs offloading tasks to it, which is $\beta_j^i(t) = \frac{F_j}{\sum_{i=1}^{M} \mathbb{1}(\alpha_i^t = j)}$, where $\mathbb{1}(x)$ is an indicator which is 1 when x is true, otherwise it is 0. It implies that the computing resource of an ES assigned to an MD relates to the task offloading decisions of the other MDs.

A. Mobility Model

We consider M MDs moving randomly within a specified area. Each MD moves one unit of distance in a random direction during each time slot. Let (x_i^t, y_i^t) denote the location of MD i at time slot t, where x_i^t and y_i^t represent the horizontal and vertical coordinates, respectively. Let x_j and y_j denote the coordinates of the j-th ES. Unlike the MDs, the positions of ESs are fixed. The distance between MD i and ES j at time slot t is $d_i^j(t) = \sqrt{(x_i^t - x_j)^2 + (y_i^t - y_j)^2}$. It implies the changing of distance from an MD to a certain ES. It further implies that the channel gain varies from time slot to time slot since it is highly related to the distance.

B. Completion Delay

Let p_i denote the transmission power of MD i and $G_i^j(t)$ denote the channel gain from MD i to ES j at time slot t. The signal-to-noise ratio (SNR) from MD i to ES j at time slot t, denoted as $SNR_i^j(t)$, is given by

$$SNR_i^j(t) = \frac{p_i G_i^j(t)}{B N_0}, \tag{1}$$

where B is the channel bandwidth, N_0 is the variance of the Gaussian white noise. The channel gain $G_i^j(t)$ is calculated by $-30 - 35 \log_{10}(d_i^j(t))$, which is highly related to the distance from MD i to ES j at time slot t.

According to the Shannon's formula, the transmission rate from MD i to ES j at time slot t is

$$r_i^j(t) = B \log_2 \left(1 + SNR_i^j(t)\right). \tag{2}$$

In this study, the task of an MD can be partially offloaded to an ES. The size of the task offloaded to the ES is $\lambda_i^t L_i^t$, while the size of the locally computed task is $(1 - \lambda_i^t) L_i^t$. Let $D_i(t)$ denote the completion delay of MD i at time t, which depends on the local computation delay and offloading delay.

Since the size of $(1 - \lambda_i^t) L_i^t$ task is processed locally on MD i, the local computation delay for MD i at time slot t is

$$D_{i,\text{loc}}(t) = \frac{(1 - \lambda_i^t) L_i^t}{f_i}, \tag{3}$$

where f_i is the computation frequency of MD i.

The remaining part of the task of MD i is offloaded to an ES. Let say the remaining task is offloaded to ES j, then its transmission delay is

$$D_{i,\text{tr}}^j(t) = \frac{\lambda_i^t L_i^t}{r_i^j(t)}, \tag{4}$$

and the computation delay on ES j is

$$D_{i,\text{mec}}^j(t) = \frac{\lambda_i^t L_i^t}{\beta_j^i(t) F_j}. \tag{5}$$

Therefore, the offloading delay for the task offloaded to ES j is

$$D_i^j(t) = D_{i,\text{tr}}^j(t) + D_{i,\text{mec}}^j(t). \tag{6}$$

The different parts of the task are processed simultaneously on the MD and an ES. Therefore, the delay is the maximum of the local computation delay and the offloading delay, which is

$$D_i(t) = \max\{D_{i,\text{loc}}(t), D_i^j(t)\}. \tag{7}$$

C. Offloading Fee

Considering that the deployment and operation of ESs incur costs. It implies that MDs should pay for the computation resources they occupy on the ES. The fee varies based on the services provided by the service operators. We denote $P_i(t)$ as the fee associated with the task offloading decision for MD i at time slot t, which is

$$P_i(t) = e_j \lambda_i^t L_i^t \tag{8}$$

where e_j denotes the fee per bit paid to ES j.

D. Cost

The total cost for MD i, denoted as $C_i(t)$, is composed of both the delay and the offloading fee at time slot t, which is

$$C_i(t) = \omega D_i(t) + (1-\omega) P_i(t), \tag{9}$$

where $\omega \in [0,1]$ is the weight.

The cost of the MEC system at time slot t is

$$C^t(\mathbf{s}^t, \mathbf{a}^t) = \sum_{i=1}^{M} C_i(t), \tag{10}$$

where $\mathbf{s}^t$ and $\mathbf{a}^t$ are the system state and action at time slot t whose details are given in Section III.

Our goal is to minimize the long-term discounted cost over T time slots, which is

$$\text{Minimize} \sum_{t=1}^{T} \gamma^{t-1} C^t(\mathbf{s}^t, \mathbf{a}^t), \tag{11}$$

where γ is the discount factor.

III. Deep Reinforcement Learning Based Task Offloading Algorithm

In this section, we first introduce the main elements of the DRL-based model. We then address our MO-DDPG algorithm.

The main elements of the DRL model are composed of state, action, and reward.

State. The state at each time slot records the task sizes and the positions of all the MDs. Let $s_i^t = (L_i^t, x_i^t, y_i^t)$ denote the state of MD i at time slot t, where L_i^t is the task size of MD i, x_i^t and y_i^t are the horizontal and vertical coordinates of MD i. The state at time slot t denoted by $\mathbf{s}^t$ composes of states of all the MDs, which is $\mathbf{s}^t = \{s_1^t, s_2^t, \ldots, s_M^t\}$.

Action. The task offloading strategy decides which ES an MD should choose and the ratio of task to be offloaded to the ES. Therefore, the action for each MD has two dimensions. The action for MD i at time slot t is denoted as $a_i^t = (\alpha_i^t, \lambda_i^t)$, where $\alpha_i^t \in \{0, 1, \ldots, N\}$ and $\lambda_i^t \in [0,1]$. If $\alpha_i^t = 0$, it means that MD i will process its task locally at time slot t. If $\alpha_i^t = j$ ($j \in \{1, 2, \ldots, N\}$), it means that MD i will offload a ratio of λ_i^t task to the ES j, and process the remaining task locally.

The action for all the MDs at time slot t is denoted by $\mathbf{a}^t = \{a_1^t, \ldots, a_M^t\}$.

Reward. Given a specific state $\mathbf{s}^t$ and an action $\mathbf{a}^t$, we define the reward as the negative of the corresponding cost, which is

$$R(\mathbf{s}^t, \mathbf{a}^t) = -C^t(\mathbf{s}^t, \mathbf{a}^t). \tag{12}$$

Our MO-DDPG algorithm bases on the DDPG algorithm, which composes of four networks: actor network μ with parameters θ^μ, target actor network μ' with parameters $\theta^{\mu'}$, critic network π with parameters θ^π, and target critic network π' with parameters $\theta^{\pi'}$ [15].

Fig. 2 shows the main processes of the MO-DDPG algorithm and the relation between the four networks. The main processes of the MO-DDPG algorithm are: updating the critic network, updating the actor network, updating the target network parameters, and the mobility-aware replay buffer which is a novelty of our algorithm.

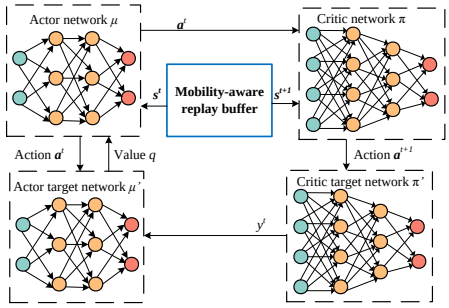

Fig. 2. The main process and the relation of the four networks of MO-DDPG

In the DDPG algorithm, the sets $(\mathbf{s}^t, \mathbf{a}^t, r^t, \mathbf{s}^{t+1})$ are stored in the experience replay buffer. During the training process, the DDPG algorithm randomly selects a set of κ experiences in the buffer to form a mini-batch and inputs them into the actor network and the critic network, respectively. We now introduce the processes with the current sampled experience $(\mathbf{s}^t, \mathbf{a}^t, r^t, \mathbf{s}^{t+1})$.

1). Update of the Critic Network. The critic network is updated by minimizing a loss function that relies on the other networks. The MO-DDPG algorithm utilizes the actor target network to compute the action for state $\mathbf{s}^{t+1}$, which is denoted as $\mathbf{a}^{t+1} = \mu'(\mathbf{s}^{t+1}|\theta^{\mu'})$. It then uses the target critic network to calculate the target value for the state-action pair $(\mathbf{s}^{t+1}, \mathbf{a}^{t+1})$, which is

$$y^t = r^t + \gamma Q(\mathbf{s}^{t+1}, \mu'(\mathbf{s}^{t+1}|\theta^{\mu'}))|\theta^{\pi'}). \tag{13}$$

The algorithm uses the critic network to obtain the estimated value for state-action pair $(\mathbf{s}^t, \mathbf{a}^t)$, which is $q = Q(\mathbf{s}^t, \mathbf{a}^t|\theta^\pi)$. The critic network updates its parameters by minimizing the loss function $L = (y^t - q)^2$.

2). Update of the Actor Network. First, the actor network obtains a new task offloading action for state $\mathbf{s}^t$, which is $\mathbf{a} = \mu(\mathbf{s}^t|\theta^\mu)$. Using critic network, the MO-DDPG algorithm can obtain the value $Q(\mathbf{s}^t, \mathbf{a})$. Then, the actor network parameters can be updated by maximizing the value, which can be obtained based on the policy gradient $\nabla_{\theta^\mu} Q(s, a|\theta^\pi)\Big|_{\mathbf{s}=\mathbf{s}^t, \mathbf{a}=\mu(\mathbf{s}^t|\theta^\mu)}$. The detailed introduction of the policy gradients can be found in [15].

3). Update of the Target Network Parameters. We adopt the same rules in [15] to update the parameters of the target

actor network and the target critic network, which are $\theta^{\mu'} = \tau\theta^{\mu} + (1-\tau)\theta^{\mu'}$ and $\theta^{\pi'} = \tau\theta^{\pi} + (1-\tau)\theta^{\pi'}$, where τ is a small constant.

4). Mobility-aware Replay Buffer. Considering the mobilities of MDs, the recent batch of experiences is more meaningful for training than the previous ones. We optimize the storage and sampling strategy of experiences by assigning a priority for every ζ experiences when adding experiences to the experience replay buffer. The most recent batch of ζ experiences are assigned a higher priority. When sampling from the replay buffer, the MO-DDPG algorithm selects $\frac{\kappa}{2}$ experiences randomly and the other $\frac{\kappa}{2}$ experiences with the highest priority.

The pseudo-code of the MO-DDPG algorithm is given in Algorithm 1, in which line 12 and line 13 show the novel experience pool and experience sampling method.

Algorithm 1 MO-DDPG algorithm

1: **Initialization**:
2: **for** episode $= 1, 2, \ldots, E_{max}$ **do**
3: **Initialize**:
4: Randomly initialize critic network $\pi(\mathbf{s}^t, \mathbf{a}^t|\theta^{\pi})$ and actor $\mu(s|\theta^{\mu})$ with parameters θ^{π} and θ^{μ}.
5: Initialize target network π' and μ' with weights $\theta^{\pi'} \leftarrow \theta^{\pi}$, $\theta^{\mu'} \leftarrow \theta^{\mu}$.
6: Initialize replay buffer $\mathcal{R}$.
7: Initialize a random process $\mathcal{N}$ for action exploration.
8: Initialize the starting state $\mathbf{s}^0$.
9: **for** $t = 1, \ldots, T-1$ **do**
10: Select action $\mathbf{a}^t = \mu(\mathbf{s}^t|\theta^{\mu}) + \mathcal{N}^t$ according to the current actor network and exploration noise.
11: Execute action $\mathbf{a}^t$ and observe reward r^t and observe new state $\mathbf{s}^{t+1}$.
12: Store the experience $\left(\mathbf{s}^t, \mathbf{a}^t, r^t, \mathbf{s}^{t+1}\right)$ in $\mathcal{R}$ with priorities determined for every ζ experiences.
13: Sample κ experiences from $\mathcal{R}$: random sample $\frac{\kappa}{2}$ experiences and the other $\frac{\kappa}{2}$ experiences with the highest priorities.
14: Set $y_k^t = r_k^t + \gamma Q(\mathbf{s}_k^{t+1}, \mu(\mathbf{a}_k^{t+1})|\theta^{\pi})$.
15: Update critic network by minimizing the loss:

$$L = \frac{1}{\kappa}\sum_{k=1}^{\kappa}\left(y_k^t - Q\left(\mathbf{s}_k^t, \mathbf{a}_k^t \mid \theta^{\pi}\right)\right)^2$$

16: Update the actor network based on the sampled policy gradient $\frac{1}{\kappa}\sum_{k=1}^{\kappa}\nabla_{\theta^{\mu}} Q(\mathbf{s}, \mathbf{a}|\theta^{\mu})\Big|_{(\mathbf{s}=\mathbf{s}_k^t, \mathbf{a}=\mu(\mathbf{s}_k^t|\theta^{\mu}))}$
17: Update the parameters of the target networks:

$$\theta^{\pi'} \leftarrow \tau\theta^{\pi} + (1-\tau)\theta^{\pi'}$$

$$\theta^{\mu'} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'}$$

18: **end for**
19: **end for**

IV. Performance Evaluation

In this section, we evaluate our task offloading algorithm by simulations. The ESs are distributed in an area of $200m \times 200m$. The computation frequency of each MD is 1.5 GHz and the computation frequency of each ES is 50 GHz. The transmission power of each MD is 100 mW. The variance of noise is -174 dBm/Hz, i.e., $N_0 = -174$ dBm/Hz. The bandwidth is set as $B = 10$ MHz. The size of tasks generated by each MD ranges from 0.16 to 0.48 Gbit. For each episode, we run the simulation for 200 time slots. The parameters of the neural network are as follows $\tau = 0.01$, $|\mathcal{R}| = 40000$, $E_{max} = 5000$, $\kappa = 64$.

We compare our task offloading algorithm with three baseline algorithms, which are DDPG-based task offloading, NN, and random algorithms. The DDPG-based task offloading algorithm. It adopts the typical DDPG algorithm to obtain the task offloading decisions. The NN algorithm selects for each MD the nearest ES as the offloading ES, and offloads all tasks of the MD to the selected ES. The random offloading algorithm. For each MD, it randomly decides to process its tasks locally or offload the tasks to a randomly selected ES.

Fig. 3 shows the costs of our MO-DDPG algorithm and the other three baselines for one episode with 200 time slots. Our MO-DDPG, DDPG-based, and the Random task offloading algorithm converges to almost the same cost, which is much lower than the NN algorithm. Moreover, our MO-DDPG algorithm converges the fastest among all the four algorithms.

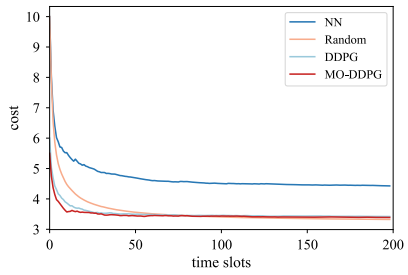

Fig. 3. Costs for one episode with 200 time slots

In Fig. 4, we fix the number of ESs to be 7 and vary the number of MDs from 3 to 10. We run the simulations for 500 episodes and evaluate the average costs for all four algorithms. The results show that the costs of the four algorithms increase with the number of MDs. Moreover, our MO-DDPG algorithm achieves the lowest cost compared with the other three algorithms.

In Fig. 5, we fix the number of MDs to be 10 and vary the number of ESs from 3 to 7. The results show that our MO-DDPG algorithm achieves the lowest cost.

Fig. 6 shows the training processes of our MO-DDPG and DDPG-based task offloading algorithms for 5000 episodes when $M = 10$ and $N = 7$. We evaluate the costs of all the episodes of our MO-DDPG and the DDPG-based task offloading algorithms. The results show that both algorithms converge to low cost when increasing the number of episodes. Moreover, our MO-DDPG algorithm achieves a lower cost than that of the DDPG-based task offloading algorithm with faster convergence speed.

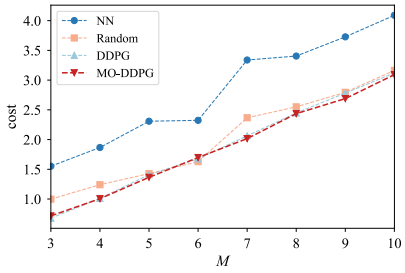

Fig. 4. Costs for different numbers of MDs

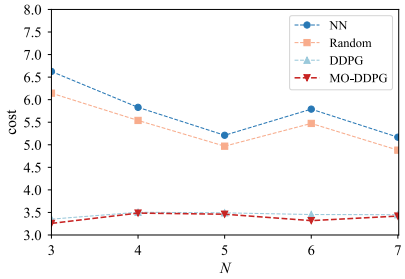

Fig. 5. Costs for different numbers of ESs

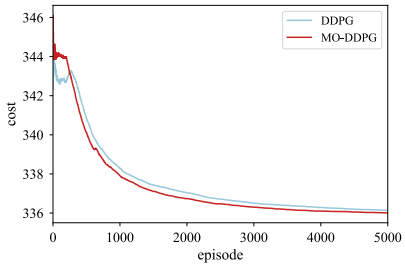

Fig. 6. Training process

To evaluate the balance between delay and charge in the goal function, we evaluate the costs under different weights ω. The larger value of ω implies that the system cares more about the delay. Fig. 7 shows that the costs converge under all settings of ω. Besides, increasing ω generally results in a larger cost which implies that delay plays a critical role.

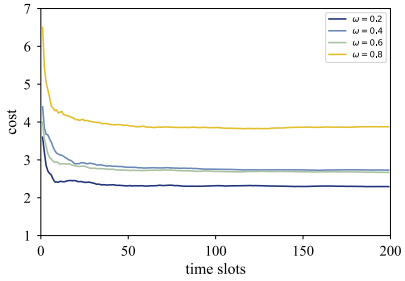

Fig. 7. Cost for different weights between delay and payment

V. Conclusion and Future Work

We studied the task offloading problem of MEC system with multiple moving MDs and multiple ESs. We formulated the task offloading problem using the deep reinforcement learning model. Based on this, we proposed the MO-DDPG algorithm with a novel mobility-aware experience replay buffer. The MO-DDPG algorithm makes hybrid decisions for MDs to decide which ES and how many tasks to be offloaded. Simulation results demonstrated that our MO-DDPG algorithm achieves lower costs with faster convergence speed compared to the baselines. Future work can consider more practical user traces.

References

[1] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Communications Surveys and Tutorials*, vol. 19, no. 3, pp. 1628–1656, 2017.

[2] K. B. Letaief, Y. Shi, J. Lu, and J. Lu, "Edge artificial intelligence for 6g: Vision, enabling technologies, and applications," *IEEE Journal on Selected Areas in Communications*, vol. 40, no. 1, pp. 5–36, 2022.

[3] S.-W. Ko, K. Han, and K. Huang, "Wireless networks for mobile edge computing: Spatial modeling and latency analysis," *IEEE Transactions on Wireless Communications*, vol. 17, no. 8, pp. 5225–5240, 2018.

[4] T. X. Tran and D. Pompili, "Joint task offloading and resource allocation for multi-server mobile-edge computing networks," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 1, pp. 856–868, 2019.

[5] K. Guo, R. Gao, W. Xia, and T. Q. S. Quek, "Online learning based computation offloading in mec systems with communication and computation dynamics," *IEEE Transactions on Communications*, vol. 69, no. 2, pp. 1147–1162, 2021.

[6] N. Zhao, Z. Ye, Y. Pei, Y.-C. Liang, and D. Niyato, "Multi-agent deep reinforcement learning for task offloading in uav-assisted mobile edge computing," *IEEE Transactions on Wireless Communications*, vol. 21, no. 9, pp. 6949–6960, 2022.

[7] Z. Zhai, X. Dai, B. Duo, X. Wang, and X. Yuan, "Energy-efficient uav-mounted ris assisted mobile edge computing," *IEEE Wireless Communications Letters*, vol. 11, no. 12, pp. 2507–2511, 2022.

[8] U. Saleem, Y. Liu, S. Jangsher, Y. Li, and T. Jiang, "Mobility-aware joint task scheduling and resource allocation for cooperative mobile edge computing," *IEEE Transactions on Wireless Communications*, vol. 20, no. 1, pp. 360–374, 2021.

[9] E. F. Maleki, L. Mashayekhy, and S. M. Nabavinejad, "Mobility-aware computation offloading in edge computing using machine learning," *IEEE Transactions on Mobile Computing*, vol. 22, no. 1, pp. 328–340, 2023.

[10] R. Singh, R. Sukapuram, and S. Chakraborty, "A survey of mobility-aware multi-access edge computing: Challenges, use cases and future directions," *Ad Hoc Networks*, vol. 140, p. 103044, 2023.

[11] L. Huang, S. Bi, and Y.-J. A. Zhang, "Deep reinforcement learning for online computation offloading in wireless powered mobile-edge computing networks," *IEEE Transactions on Mobile Computing*, vol. 19, no. 11, pp. 2581–2593, 2019.

[12] Z. Gan, R. Lin, and H. Zou, "A multi-agent deep reinforcement learning approach for computation offloading in 5g mobile edge computing," in *2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE, 2022, pp. 645–648.

[13] J. Tan, R. Khalili, H. Karl, and A. Hecker, "Multi-agent distributed reinforcement learning for making decentralized offloading decisions," in *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*. IEEE, 2022, pp. 2098–2107.

[14] K. Yan, H. Shan, T. Sun, H. Hu, Y. Wu, L. Yu, Z. Zhang, and T. Q. Quek, "Reinforcement learning-based mobile edge computing and transmission scheduling for video surveillance," *IEEE Transactions on Emerging Topics in Computing*, vol. 10, no. 2, pp. 1142–1156, 2021.

[15] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. M. O. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *CoRR*, vol. abs/1509.02971, 2015.