Dynamic Computation Offloading Based on Deep Reinforcement Learning

Baichuan Cheng[1], Zhilong Zhang[2] and Danpu Liu[3]

Beijing University of Posts and Telecommunications, China
buptcbc@bupt.edu.cn,zhilong.zhang@outlook.com,dpliu@bupt.edu.cn

Abstract. Mobile edge computing (MEC) provides computation capability at the edge of wireless network. To reduce the execution delay, computation-intensive multimedia tasks can be offloaded from user equipments (UEs) to the MEC server. How to allocate the computational and wireless resources is one of the key issues to guarantee the quality of services, and is very challenging when tasks are generated dynamically. In this paper, we address the above problem. To minimize the sum execution delay of multiple users, we jointly optimize the offloading decision and the allocation of both computational and wireless resources. We propose a deep policy gradient (DPG) algorithm based on the deep reinforcement learning. Simulation results show that our proposed DPG method can achieve lower latency than the baselines under different numbers of users, computation capacities and wireless bandwidths.

Key words: Computation offloading, mobile edge computing, reinforcement learning, policy gradients

1 Introduction

Many promising applications in future mobile networks, such as virtual/augmented reality (VR/AR), high definition video transmission and autonomous driving, demand for more intensive computation, larger bandwidth and lower latency [1, 2]. However, current mobile devices can not well satisfy these requirements due to their limited computation and communication capacities. The tension between the evident weakness of mobile devices and the high demand of applications poses significant challenges for future networks. Although mobile cloud computing can offload the computation-intensive task to the remote public clouds to relieve the burden of UEs, the long latency for reaching the remote public cloud through a wide area network is unacceptable for delay-sensitive applications.

To reduce the delay, a novel network architecture called MEC has been recently proposed. MEC [3] provides cloud-computing capabilities in close proximity to mobile users, and offers a low-latency and high rate access service environment. With the assistance of MEC, the computation-intensive applications can be offloaded to the resource-rich cloud computing infrastructures within the edge of mobile networks. By computation offloading, the latency is expected to be significantly reduced.

MOBIMEDIA 2019, June 29-30, Weihai, People's Republic of China

DOI 10.4108/eai.29-6-2019.2282108

However, computation offloading is not always beneficial, and sometimes results in even higher latency. This is because both the computational resource and the wireless bandwidth used to transmit application data are limited in practice. If these resources are not properly allocated, the latency may be increased. Hence, how to determine the tasks to be offloaded and how to allocate different types of resource are key issues for computation offloading services in MEC systems [4].

The above mentioned problem has been widely addressed. For example, the authors in [5] propose a policy for computation offloading applications by solving a convex problem. The policy has a simple threshold structure to control the offloading data and time allocation. In [6], the energy and latency cost is minimized for MEC systems. Although these works can achieve good performance, there are common drawbacks that the dynamic arrival of tasks and the long-term latency are not considered, which are necessary for practical use.

Reinforcement learning (RL) is commonly regarded as an appropriate technology in addressing the above issue in practical system environments. A group of RL-based approaches are proposed recently. In order to minimize the cost of delay and energy consumption, the work in [7] designs a Deep Q Network (DQN) policy, which jointly optimizes the computation offloading decision and resource allocation. With the consideration of stochastic characteristics, an online resource management learning algorithm in [8] is proposed, which learns the optimal policy of dynamic workload offloading and edge server provisioning. For offloaded workload, the latency is mainly transmission delay due to network round trip time. Considering the delay of data transmission by wireless channel, the work in [9] model the computation offloading process as Markov Decision Process (MDP) and a polynomial value function approximation method for continuous state space is proposed. Considering task buffer stability constraint and a stochastic task arrival model, the work in [10] proposes a novel DRL-based framework for power-efficient resource allocation and applys a Deep Neural Network to approximate the action-value function.

Although RL has been introduced in MEC system, most of them focus on the resource allocation problem without considering the dynamic decision of computation offloading. Moreover, most existing works adopt classical RL methods based on value iteration which usually have a large time complexity for the practical system. Hence, in this paper, we aim to propose a policy iteration-based scheme which is expected to converge faster. We consider the stochastic task arrival model, and jointly optimize the offloading decision and the resource allocation to reduce the execution latency of all tasks. We adopt deep policy gradients (DPG) which uses neural network to estimate the policy gradients (PG).

In summary, the major contributions of the paper are:

– We consider the stochastic task arrival model and the tasks can be executed either at the local UE or be offloaded to the MEC server. By jointly optimizing the offloading decision and the allocation of network and computation resources, we propose a policy iteration-based algorithm to minimize the total

execution latency. Moreover, for the problem of the large space actions, we use a trick to effectively reduce the action space.
– We evaluate the performance of our proposed algorithm. The simulation results show that the performance of the proposed algorithm outperforms the baselines in delay performance under different numbers of users, computation capacities and wireless bandwidths.

The rest of the paper is organized as follows. We first present the system model in Section II. Then, we formulate the multi-user computation offloading problem in Section III. In Section IV, we provide a DRL framework where the state, action and reward are designed based on the characteristics of the computation offloading problem. We present the simulation results in Section V and conclude this paper in Section VI.

2 System Model

2.1 Network Model

We consider an MEC-enabled wireless system as shown in Fig 1. A BS connects to a high-performance server which is used to execute the offloaded tasks. A set $\mathcal{N}$ of users are located within the coverage of the BS, each of which has a computation-intensive task which can be executed either on the local equipment or offloaded to the MEC server. We assume that a task cannot be further divided into multiple parts to execute on more than one device.

The BS manages the uplink/downlink communications of mobile users. We apply a time division multiple access (TDMA) method for the channel access. In this method, the time is divided into slots which will be allocated to UEs. Only one UE can offload its task to the MEC server within a time slot. Let B denote the wireless channel bandwidth. The uploading rate for UE n is given by

$$r_n = B\log_2\left(1+\frac{g_n p_n}{BN_0}\right), \tag{1}$$

where p_n denotes the transmission power of UE n for uploading data, g_n denotes the channel gain between UE n, and the base station and N_0 is the variance of white Gaussian channel noise.

2.2 Task Model

Each UE has a computation-intensive task to execute. A task $T_n \triangleq (v_n, s_n, a_n)$ is composed of three parts: v_n denotes the size of the executed task, including input parameters and program codes; s_n denotes the amount of CPU cycles that a task consumes; a_n is the arrival time of the task generated in user n. All the three parameters can be estimated by task profiles. We assume that stochastic task model is characterized by random arrivals. s_n is positively related to the size of v_n. Due to the different computation capacities between UE and MEC server,

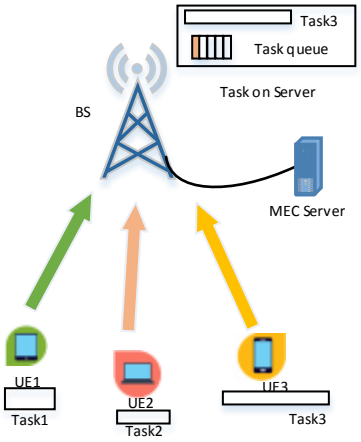

Fig. 1. System Model.

the time cost for executing the task is different. The UE first send a request to the BS and wait for the offloading decision as shown in Fig 1. The arrived but not yet executed tasks will wait on a queue for scheduling. We define $c_n \in \{0, 1\}$ as the offloading decision for UE n. If BS decides to offload the task of UE n to the MEC server, BS will set $c_n = 0$. Otherwise, $c_n = 1$.

Local Computing Model. Different UEs may have various computation capacities, we denote the computation capacity of UE n by f_n. If a task is executed locally, the execution delay T_n^l is given by

$$T_n^l = \frac{s_n}{f_n}. \tag{2}$$

Offloading Computing Model. If a task is executed in the MEC server, the whole offloading procedure will consist of three steps. Firstly, a UE uploads its task program codes and parameters to the BS via wireless channel. Then, the BS forwards the data to the MEC server. Secondly, the MEC server allocates some of the computation resource to execute the task. Finally, the BS sends the computation result back to the mobile user. Based on (1), we can obtain the data transmission delay T_n^t by

$$T_n^t = \frac{v_n}{r_n}. \tag{3}$$

We assume that only one task can be executed by the MEC server at a time, and other tasks will wait until the BS makes the offloading decision. Hence, the task may wait for scheduling on the local equipment and be offloaded in the next time slot. We set the time that the UE uploads the transmission data via wireless channel as t_n, which means the network scheduling time. The execution delay T_n^e is given by:

$$T_n^e = \frac{s_n}{f^c}, \tag{4}$$

where f^c is the MEC server computation resource (i.e., CPU cycles per second). More than one task on the MEC server may wait to be executed, so there is a queue on the MEC server that stores the offloading tasks. Once the running task ends, MEC server gets the next task from queue. We set the time that task begins to execute on the MEC server as τ_{n}, which means the computing scheduling time.

Finally, the computation result has to be sent back to the corresponding UE. According to [11, 12], the data size of computation outcome is very small compared to the uploading data, so we ignore the backhaul delay.

Since the MEC server starts to perform the task after the task data has been uploaded, the following inequality should be satisfied that

$$\tau_n \geq t_n + T_n^t. \tag{5}$$

The total delay for UE n to offload its task to the BS is given by:

$$T_n^o = \tau_n + T_n^e - a_n. \tag{6}$$

3 PROBLEM FORMULATION

Our aim is to minimize the total latency. Based on (2), (5) and (6), an optimization problem is formulated as follows

$$\begin{aligned} \min_{c_n, t_n, \tau_n} \quad & \sum_{n=1}^{N} c_n T_n^l + (1 - c_n) T_n^o \\ s.t. \quad & c_n \in \{0, 1\}, \\ & \tau_n \geq t_n + T_n^t. \end{aligned} \tag{7}$$

We can solve the problem (7) by finding the optimal value of c_n , t_n and τ_n. However, the problem is not convex due to the integer optimization variables. The solution space of the problem increases rapidly with the increase of the UEs. Hence, we adopt a DRL method to solve the non-convex problem.

4 DRL-based algorithm design

In this section, we present our DRL-based algorithm design. Firstly, we formulate the state, the action, and the reward of the RL model. Then we introduce our DPG method.

4.1 RL Formulation

State The state contains the BS network, MEC server computation resources occupation and the jobs waiting to be scheduled. Fig.2 shows the allocation of computation and network resource, starting from current time slot and looking ahead T steps into the future. When UE has a task needs to be executed, it will send the task information. The BS will store the job in the backlog. Then if the job slot is empty, the job in the backlog will schedule the place and wait the BS deciding to offload or not. The job in the slot will represent the network and computation resources consumption, which are represented by the time slots that need to be allocated from the current time slot for network and computation resource. For example, the red job in the slot 1 need two time slots computation resource and a time slot network resource from the current time slot. If BS offloads the job to the MEC server, MEC resource will consume a time slot network resource and two time slots computation resource. Because the task needs to be uploaded to the BS, it will first allocate the network resource and then allocate computation resource.

We can have as many job slots in the state as there are jobs waiting for service. However, it is not desirable for neural network input that needs to have a fixed state representation. Therefore, we use a column to represent the network resource and a column to represent the computation resource for the BS, and we limit the number of tasks that BS can offload to be K. The length of the backlog is limited. If the backlog is full, a UE's request will be ignored by the BS.

Action The action refers to the decisions that whether offload a job to the MEC server or local equipment. The BS can schedule the offloaded tasks from job slots. The BS scheduler may choose any subset of the job slots. Therefore, the action space is as large as 2^K . So we use a trick to make the action space small. In a time slot, the BS scheduler can execute more than one action continuously. The action space is $\{0, 1, 2, ..., 2K\}$. $a = 0$ means there is no job scheduling and it is a void action. $a \in \{1, 2, ..., K\}$ means the BS chooses the job at the a-th job slot to offload to the MEC server. $a \in \{K + 1, K + 2, ..., 2K\}$ means the job at the $a - K$-th job slot to offload the local UE. The time slot is non-frozen if BS scheduler chooses either void action or invalid action (e.g. the BS scheduler choose the job at job slot 3 to offload to the MEC server, but the MEC resource cannot afford to execute the job). For each valid action (e.g. the first time slot can satisfy the job resource's requirement till completion), the agent will observe a state transition due to the job moving to the appreciate position on the MEC resource. If the BS scheduler chooses void action or invalid action, the time slot will proceed and move one step down. The BS scheduler can choose again job due to the new state. Hence, we can keep the action space linear by choosing multiple actions in a time slot.

Reward The agent on reinforcement learning is to maximize the expected cumulative discounted reward $\sum_{t=0}^{n} \gamma^t r_t$, γ is the discounted factor. Our goal is to minimize the tasks latency. Specifically, we set the reward $r_t = \sum_{j \in J} -1$, where J is

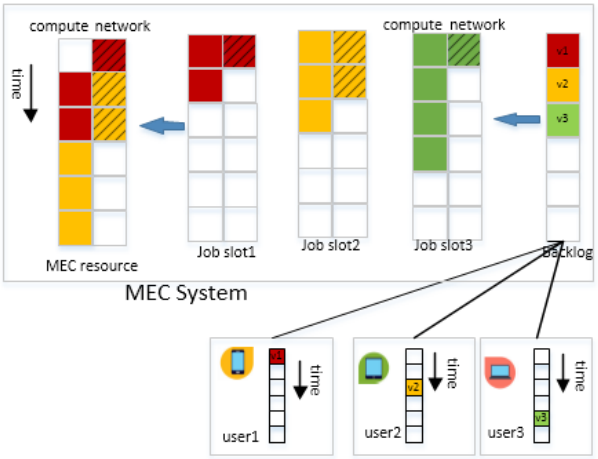

Fig. 2. System state representation. BS stores the user's task information and choose task from job slot.

the job on the system. The agent doesn't receive reward when time slot is frozen. When the job completes on the local UE or MEC server, it no longer belongs to J. We set the discounted factor $\gamma = 1$, the maximizing cumulative discounted sum is equivalent to minimizing the sum of jobs latency. Hence, maximizing the cumulative sum is equivalent to minimizing the jobs latency.

4.2 Deep Policy Gradient Algorithm

The traditional RL method simulates the state change through tables. When the number of states is too large, it is hard to traverse each state and results in failures of the algorithm. By contrast, the PG schemes can overcome the drawback. We first briefly introduce the PG technology that we adopt in the paper. The agent chooses an action based on a policy, defined as $\pi, \pi(s, a) \rightarrow [0, 1]$, which means the probability that the action is chosen for the state. Due to the large number of {state, action} pairs, it is impossible to store each policy in tabular. Hence, we use the neural network to represent the policy $\pi_\theta(s, a)$. The objective is to maximize the cumulative reward, and we define $Q^{\pi_\theta}(s, a)$ as the cumulative reward that the action a is chosen in the state s. The gradients of maximizing the cumulative reward is

$$\nabla_\theta E_{\pi_\theta}[\sum_{t=0}^{\infty} \gamma^t r_t] = E_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s, a) Q^{\pi_\theta}(s, a)]. \tag{8}$$

The key idea in equation (8) is to estimate the value $Q^{\pi_\theta}(s, a)$. In the Monte Carlo Method, it is estimated by agent sampling multiple trajectories and computing the cumulative reward v_t at each timestep t. The parameters θ is updated by

$$\theta \leftarrow \theta + \alpha \sum_{t} \nabla_{\theta} \log \pi_{\theta}(s_t, a_t) v_t. \tag{9}$$

In order to derive a strategy that maps an input state to an output action, we adopt a policy gradient method. We represent the strategy using neural network. The input state is shown in Fig 2, and its output is a probability distribution of all actions. We train the network in an episodic setting. The episodic will not terminate until all task are completed.

In the process of training network, we use different job sets, where the arrival time and number of jobs are different. It is not sufficient to train the policy by only exploring once. So we use N networks in parallel to explore the possible action space for each job set and store the corresponding results of state, action and reward.

Note that, policy gradients may fail to converge due to the high variance. To reduce variance, the cumulative reward usually subtracts a baseline, which is obtained by averaging the cumulative discounted reward on the same time slot of all episodes with the same job set. Due to the exploration of N networks, the policy enables current network to converge fast to the best one.

5 SIMULATION

5.1 Simulation setup

We simulate a wireless system composed of a BS and multiple users. The BS is connected to a MEC server and locates at the center of the cell. The UEs are randomly scattered within 500 meters away from the BS. The main simulation setting is listed in Table. 1. We compare our proposed algorithm with four baselines, namely, *mobile execution*, *MEC server execution*, *random execution*, and *DPG with the average allocation of network resource by offloading multiple users* (DPGM) [7]. They work as follows:

- **Mobile Execution:** All the tasks are executed on the local equipment.
- **MEC Server Execution:** All the tasks are offloaded to the MEC server.
- **Random Execution:** When the job slots are not empty, the job is executed in the MEC server or the local equipment randomly.
- **DPGM:** The system bandwidth is allocated equally to the offloaded users, then the computation resource is allocated simultaneously to these users.

Jobs are generated randomly. Job computation and network resource demands is chosen as follows: 20% of the jobs have duration uniformly chosen from 1 to 3 time slots; the remaining are chosen uniformly from 4 to 8 time slots.

5.2 Performance Evaluation

Fig.3 shows the latency all jobs are completed achieved by the learning policy. Due to the N networks in parallel to explore the possible action space, DPG

Table 1. Simulation Parameters

Parameters	Values
Number of job slots	5
Length of backlog	60
System bandwidth	20MHz
Shadow fading	10dB
User transmission power	23dB
Local CPU frequency	1GHZ
Number of UEs	30
MEC CPU frequency	3GHZ
Gaussian Noise power	-88dB

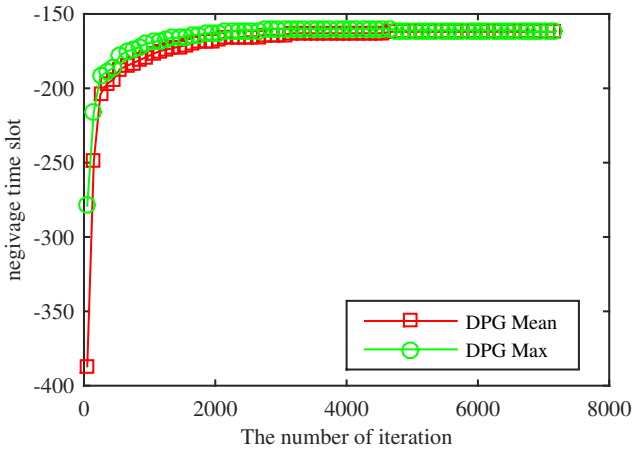

Fig. 3. Convergence of the total reward in the training procedure.

Max means the maximum reward across all of the Monte Carlo running at each iteration and DPG Mean means the average reward. The large gap between the maximum and average reward implies the policy needs to be improved further. When the DPG Max isn't equal to DPG Mean at each iteration, it indicates there are some action paths that are much better than the average action paths, so it is necessary to adjust the policy to increase reward. Conversely, if the gap narrows, it means the policy has chosen better action than ever before. As expected, the DPG Mean and DPG Max improve with the iteration number increase. When the iteration number is 200, the gap between the maximum and average reward is smaller. Finally, the model has totally converged when the number of iterations is near 5000.

Fig.4 shows the completing all job latency with respect to the different number of UEs. The execution latency increase with the increase of UE number. DPG method achieves the best result that the latency is smallest when the UE number is different. When the UE number is 50, DPG outperforms DPGM and

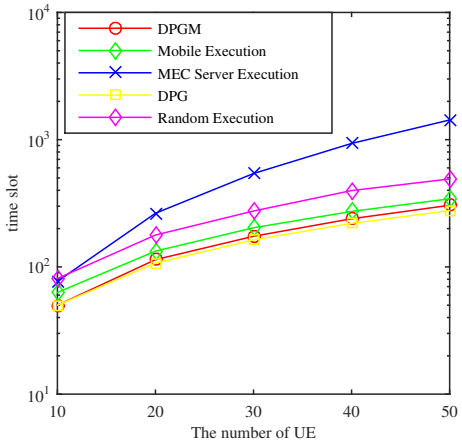

Fig. 4. Latency with different number of UEs.

offers more than 10% performance gain. Because network resource is divided into many parts by DPGM method, it will increase the transmission latency and the job offloaded to the MEC server will consume longer time due to the allocation of computation resource. MEC Server Execution gets the worst result. Because there is only one server to execute the offloading tasks, job needs to wait for execution at the server queue. Random Execution can utilize the local computation resource. The Mobile Execution outperforms Random , and the time slot consumed is reduced by 30% when UE number is 50. It is because the task can be offloaded to the local equipment directly. The DPG and DPGM outperform Mobile Execution. Because it utilizes the MEC computation resource and the task that is offloaded to the MEC server will not consume too many time slot to wait for execution.The training algorithm can adappt to the changes in the environment.

Fig. 5 shows the latency cost with the increase of MEC CPU frequency to local equipment CPU frequency ratio. When MEC CPU frequency is equal to the local equipment CPU frequency, DPG, DPGM, Mobile Execution are equal. Because the MEC CPU frequency is same as local CPU frequency. Hence, the best strategy that DPG explores is to execute the task on the local equipment. Task is offloaded to the MEC server will consume more time slot due to transmission latency. With the increase of MEC CPU frequency, DPG outperforms DPGM and Mobile Execution. Due to the smaller execution latency on the MEC, DPG and DPGM explore better strategy than Mobile Execution. When MEC CPU frequency to local equipment CPU frequency ratio is 5, DPG offers more than 20% performance gain compared than DPGM. The allocation of computation and network resource for offloading tasks causes longer latency in DPGM. DPG will schedule the tasks that wait to execute on the server to local equipment if the latency by offloading is larger. MEC Server Execution is the worst result. It will lead to all tasks offloaded to the MEC server to execute. Firstly, all tasks

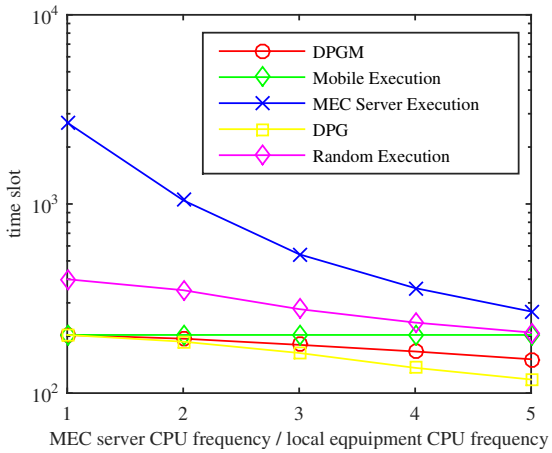

Fig. 5. The latency for different MEC CPU frquency.

need to allocate network resource to transmit data, which causes many tasks will wait longer time slot to transmit data. Secondly, the offloaded tasks will wait for execution on the MEC server. Random Execution randomly schedule some tasks to MEC server, the other tasks is executed on the local equipment. Due to the local computation utilization, it outperforms MEC Server Execution.

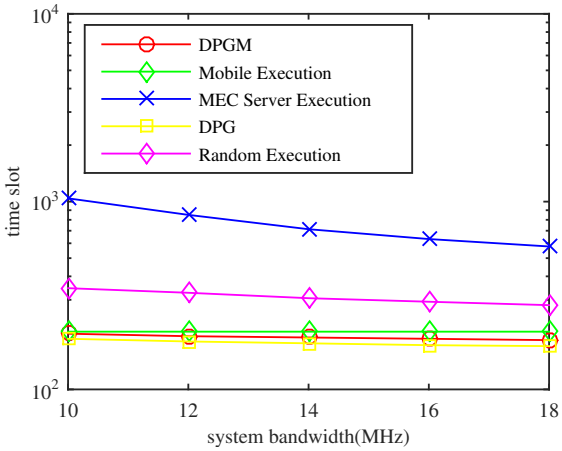

Fig. 6. The latency for different system bandwidth

Fig.6 shows the total latency of completing all job with respect to the different system bandwidth. Mobile Execution is constant under different system bandwidth parameters, due to the task is same. The different system bandwidth

parameters will lead to different transmission latency. However, our algorithm outperforms baselines. MEC Server Execution is still worst. Random Execution outperforms MEC Server Execution due to the utilization of local computation resource. DPGM is worse than DPG because the allocation of computation and network resource by offloading tasks cause longer time slot to transmit data and execute task.

6 CONCLUSION

In this paper, we address the probem of computation offloading and the allocation of network and computation resource in a multi-user system. We formulate the state, action, and reward models, and propose the DPG algorithm. Our simulation results show that our proposed method outperforms baselines under different system parameters. For the future work, we will extend our research by jointly consider the energy and latency consumption.

Acknowledgment. This work is supported by the National Natural Science Foundation of China under Grant 61801051, the Beijing Natural Science Foundation under Grant No. L172032, the Open Project of Information Security Laboratory of National Defense Scientific Research and Test under Grant No. 2016XXAQ09, and the 111 project under Grant No.B17007.

References

1. https://www.ngmn.org/5g-white-paper/5g-white-paper.html
2. Z. Zhang, D. Liu, X. Wang, "Joint carrier matching and power allocation for wireless video with general distortion measure," *IEEE Trans. Mobile Comput.*, vol. 17, no. 3, pp. 577-589, March 2018.
3. S. Wang et al., "A survey on mobile edge networks: Convergence of computing, caching and communications," IEEE Access, vol. 5, pp. 6757-6779, 2017.
4. P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," IEEE Communications Surveys Tutorials, vol. 19, no. 3, pp. 1628-1656, 2017
5. C. You, K. Huang, H. Chae, and B.-H. Kim, "Energy-efficient resource allocation for mobile-edge computation offloading," IEEE Trans. Wireless Commun., vol. 16, no. 33, pp. 1397-1411, Mar. 2016.
6. 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, Oct. 2016.
7. J. Li, H. Gao, T. Lv, and Y. Lu, "Deep Reinforcement Learning based Computation Offloading and Resource Allocation for MEC" in Proc. IEEE Wireless Commun. Netw. Conf. (WCNC), Barcelona, Spain, Apr. 2018 pp. 1-9.
8. J. Xu, L. Chen and S. Ren, "Online learning for offloading and autoscaling in energy harvesting mobile edge computing," IEEE Trans.Cogn. Commun. Netw., vol. 3, no. 3, pp.361-373, Sep. 2017.

9. Z. Wei, B. Zhao, J. Su and X. Lu, "Dynamic Edge Computation Offloading for Internet of Things with Energy Harvesting: A Learning Method," in IEEE Internet of Things Journal. doi: 10.1109/JIOT.2018.2882783
10. Z. Xu, Y. Wang, J. Tang, J. Wang, and M. C. Gursoy, "A deep reinforcement learning based framework for power-efficient resource allocation in cloud rans," in 2017 IEEE International Conference on Communications (ICC), May 2017, pp. 1-6.
11. X. Chen, L. Jiao, W. Li and X. Fu, "Efficient Multi-User Computation Offloading for Mobile-Edge Cloud Computing," in IEEE/ACM Transactions on Networking, vol. 24, no. 5, pp. 2795-2808, October 2016. doi: 10.1109/TNET.2015.2487344
12. K. Zhang et al., "Energy-Efficient Offloading for Mobile Edge Computing in 5G Heterogeneous Networks," in IEEE Access, vol. 4, pp. 5896-5907, 2016. doi: 10.1109/ACCESS.2016.2597169