Double DQN Reinforcement Learning-Based Computational Offloading and Resource Allocation for MEC

Chen Zhang[1], Chunrong Peng[2], Min Lin[1], Zhaoyang Du[3], and Celimuge Wu[3](✉)

[1] College of Computer Science and Technology, Inner Mongolia Normal University, Saihan District, Hohhot, Inner Mongolia Autonomous Region, People's Republic of China
[2] Library, Inner Mongolia University of Finance and Economics, Xincheng District, Hohhot, Inner Mongolia Autonomous Region, People's Republic of China
[3] Graduate School of Informatics and Engineering, The University of Electro-Communications, 1-5-1 Chofugaoka, Chofu, Tokyo 182-8585, Japan
celimuge@uec.ac.jp

Abstract. In recent years, numerous Deep Reinforcement Learning (DRL) neural network models have been proposed to optimize computational offloading and resource allocation in Mobile Edge Computing (MEC). However, the diversity of computational tasks and the complexity of 5G networks pose significant challenges for current DRL algorithms apply to MEC scenarios. This research focuses on a single MEC server-multi-user scenario and develops a realistic small-scale MEC offloading system. In order to alleviate the problem of overestimation of action value in current Deep Q-learning Network (DQN), we propose a normalized model of Complex network based on Double DQN (DDQN) algorithm to determine the optimal computational offloading and resource allocation strategy. Simulation results demonstrate that DDQN outperforms conventional approaches such as fixed parameter policies and DQN regarding convergence speed, energy consumption and latency. This research showcases the potential of DDQN for achieving efficient optimization in MEC environments.

Keywords: mobile edge computing · computational offloading · resource allocation · single MEC server-multi-user · DDQN algorithm

1 Introduction

Currently, most of IoT applications have increasingly high demands for latency and transmission reliability. Achieving Ultra-Reliable and Low-Latency Communication (URLLC) is one of the major challenges for 5G networks, Fig. 1 illustrates the communication requirements for IoT-related applications in terms of latency and reliability [1], with latency varying between 1 ms (ultra-low) and 100 ms (low) and reliability varying between 1–10^{-2} (high) and 1–10^{-9} (ultra-high). With the exponential growth of IoT devices, Mobile Edge Computing (MEC) undoubtedly provides an efficient solution for

© ICST Institute for Computer Sciences, Social Informatics and Telecommunications Engineering 2024
Published by Springer Nature Switzerland AG 2024. All Rights Reserved

C. Wu et al. (Eds.): MONAMI 2023, LNICST 559, pp. 240–253, 2024.
https://doi.org/10.1007/978-3-031-55471-1_18

computationally intensive programs [2]. MEC sinks computing services to the edge of the network by deploying cloud servers at the Base Station (BS), thus bringing computing resources closer to the users to provide more efficient services. Due to the limited computational resources of end-user devices, they can offload latency-sensitive computing tasks to MEC servers via wireless channels to acquire additional computational resources.

Therefore, with the advent of 5G and 6G networks, MEC has attracted considerable interest. Nowadays, DRL has gained significant attention for successful perception-based decision making in complex scenarios [3]. This has prompted researchers to explore the integration of RL with conventional MEC algorithm models, resulting in notable advancements in this domain [4].

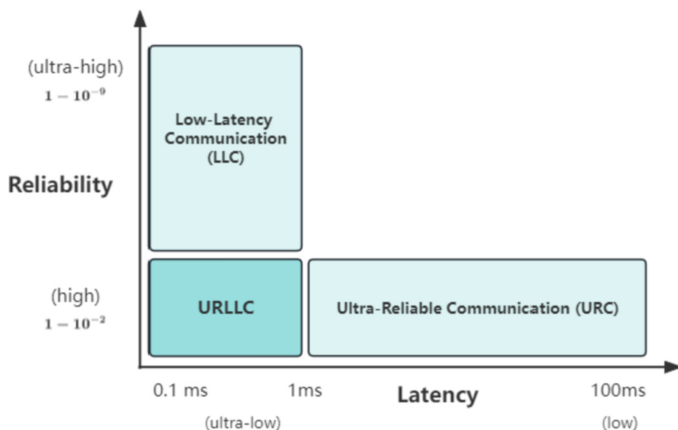

Fig. 1. Delay and transmission reliability communication requirements.

Liu et al. [5] implemented edge node selection and offloading sequence through heuristic algorithm and reconstruction linearization technology in a single-user multi-MEC server scenario to achieve a balance of delay performance and reliability performance in MEC. Liu et al. [6] studied the single MEC and muti-users in high reliability and low latency scenarios, then they introduce probability and extreme value theory to analyze the user's task queue and use Lyapunov theory to solve the problem of minimizing calculation and transmission energy. Huang et al. [7] proposed a DQN based task offloading and resource allocation algorithm for MEC, motivated by the concomitant need for proper resource allocation for computational offloading via MEC. Liang et al. [8] proposed a combination of DQN and Deep Deterministic Policy Gradient (DDPG) to optimize the total cost of UE transmission delay and energy consumption. Liang et al. [9] investigated the problem of computational offloading of interference perception in single MEC server and multi-user scenarios. Wu et al. [10] formulated the offloading decision and resource allocation problem. Li et al. [11] proposed method that users' tasks can be offloaded to multiple computing access points (CAPs). Gan et al. proposed an offloading strategy to jointly minimize latency [12], energy consumption of ES, and task loss rate while preserving privacy (PP). The above works have designed strategies

from different perspectives and methods, but they lack consideration in terms of convergence speed and learning stability. Furthermore, in certain stochastic environments, the widely recognized reinforcement learning algorithm Q-learning exhibits significant performance deficiencies. The inadequate performance arises from the overestimation of action values, which is a consequence of Q-learning's utilization of the maximum action value as an estimate for the maximum expected action value, thereby introducing a positive bias [13].

We combine the DDQN algorithm with the principle of partial offloading to alleviate the problem of overestimation of action values. As shown in Fig. 2, in the simulated single MEC server-multi-user edge computing scenario, a group of N mobile User Equipment (UE) offloads their computational tasks to MEC server over a wireless link, each end-user device offloads computing tasks and acquires computing resources at minimal total task cost. The contributions of this study are as follows:

- In the proposed MEC system, each UE can independently make sound decisions to minimize total consumption costs.
- We apply DDQN to solve the MDP model considering both latency and energy consumption, and give a generic offloading strategy for different scenarios.
- We contemplate establishing a physically existing MEC network environment that captures dynamic network structure and reliability of communication. Unlike most studies that rely on simulation software, our approach utilizes real-world data to estimate the performance of offloading schemes. Additionally, the impact of each offload decision on the overall MEC system, including the CPU utilization of other UEs, is taken into account.
- According to the experimental findings, the use of the DDQN algorithm yields the lowest total cost of ownership compared to other baselines.

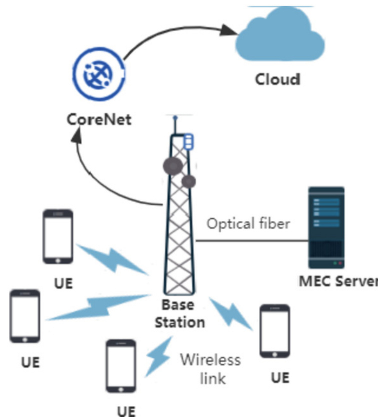

Fig. 2. The model of MEC server-multi-user edge computing scenario

The paper is structured as follows. In Sect. 2, we define the network environment and proposed computational offloading and resource allocation optimization. Section 3

presents our algorithm to address the problem mentioned above. Subsequently, in Sect. 4 conducts simulations of the proposed algorithm and discusses the obtained results. Section 5 provides the conclusions of this study.

2 System Overview

In this section, a system model for the optimization of computational offloading and resource allocation strategy is given. Initially, we present the network infrastructure of the real-life scenario of the MEC being simulated. On top of this it is analyzed and the optimization problems studied are presented in detail.

2.1 Network Module

To better emulate the heterogeneity and flexibility of MEC networks, we simulated a real MEC system using Raspberry Pi 4B and personal laptops. Raspberry Pi serves as an intelligent mobile device, incorporating computation, communication, and storage modules. As shown in Fig. 3, the laptop simulates the MEC server that provides computing resources for the entire MEC system. Raspberry Pi-1 simulates the base station connected to the MEC server and acts as an Access Point (AP). All Raspberry Pi devices (Pi-1, Pi-2, Pi-3, Pi-4) are part of an Ad hoc network environment. Ad hoc networks are characterized by self-organization, temporariness, and decentralization. In a self-organizing network, devices communicate through temporary connections to form an ad hoc network, independent of traditional infrastructure such as routers or base stations [14]. The MEC server linked to the AP via a network cable serves the n UEs in the ad hoc network. The system's main task is real-time object detection using the YOLO algorithm [15] on fixed-size images captured by the surveillance system. Due to the resource limitations of the Raspberry Pi devices, partial or binary offloading to the MEC server or other UEs [16] is required to handle computationally intensive tasks.

The Fig. 3 illustrates the network environment in which the experiments take place. We regard the simulated scenario as a fundamental network entity and portray it through the collective attributes of all UEs. Subsequently, the underlying network entity in the network scenario provide observations for the environment. M is the maximum workload of the MEC server, while C represents the maximum computational resources provided by the MEC server. Within the network entity, let $U_i(i \in \{1, 2, ..., N\})$ denote the set of UE in the network. Each UE can be defined as $UE_i = \{m_i, f_i, t_i, r_i, s_i\}$, for $\forall i \in U_i$ where mi represents the computation load of the user's task. If mi = 0, it implies that the terminal does not exist. f_i denotes the required computational resources for the user's task, while t_i represents the maximum waiting time of the UE. The variable r_i indicates the current state of the computation, when $r_i = 1$ indicating that the task is being computed. In the case where $m_i = 0$ and $r_i = 1$, the resources occupied by the terminal are still reserved and only the consumption of the other UEs is computed. The variable $s_i \in \{1, 2, 3\}$ represents the set of system offloading decisions, representing different offloading strategies. $s_i = 1$ denotes local computation, $s_i = 2$, represents task offloading, and $s_i = 3$ indicates task migration to another MEC server.

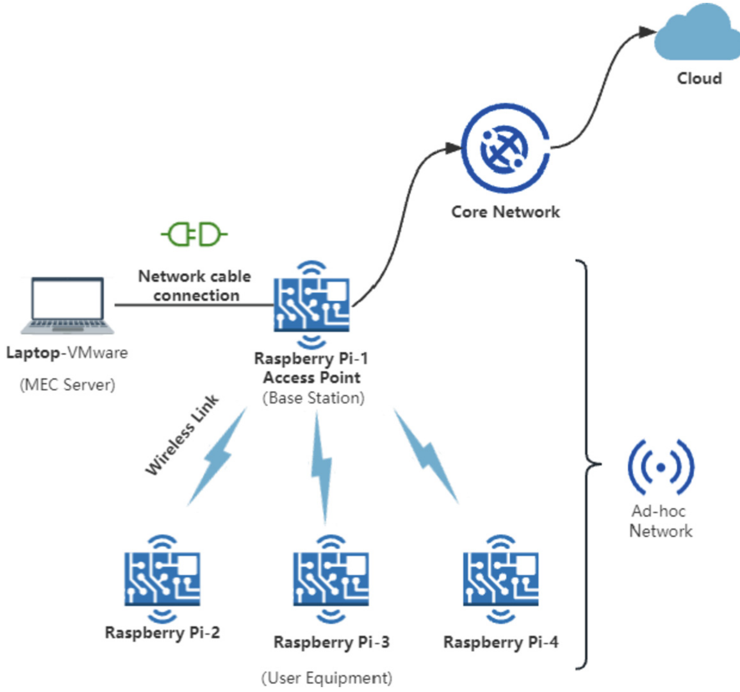

Fig. 3. MEC system network model

2.2 Computational Module

1. Local computing: We assume that the CPU in UE_i operates at a frequency f_i, representing the local computational power in terms of CPU cycles. Therefore, the local computation time can be calculated as follows:

$$t_i^{local} = \frac{f_i}{c_i^{local}}, \ \forall i \in U_i \tag{1}$$

Based on the findings presented in reference [17], the local energy consumption can be mathematically represented as:

$$E_i^{local} = kf_i(c_i^{lcoal})^2, \forall i \in U_i \tag{2}$$

Incorporating the influence of chip architecture [18], the effective switching capacitance κ is introduced in the equation. The total energy consumption is given by the expression (α, β is Loss factors):

$$C_i^{local} = \alpha t_i^{local} + \beta E_i^{local} \tag{3}$$

2. Edge computing: Delays and energy losses are introduced when tasks are offloaded to the MEC server. Assuming a constant transmission power for the MEC, denoted as pi and utilizing the standard path loss propagation index θ, di is the distance between

UE_i and the base station, the $UE_j{\prime}s$ signal-to-noise ratio (SNR) can be formulated as follows:

$$SNR_i = \frac{p_i h_i d_i^{-\theta}}{\sigma^2}, \forall i \in U_i \tag{4}$$

The SNR of UE_i is determined by the power of additive Gaussian white noise (σ^2) and the channel gain (h_i). Based on this, the upload transmission rate of UE_i can be described as:

$$R_i = W_0 \log_2(1 + SNR_i) \tag{5}$$

Thus, transmission delay is:

$$t_i^{tran} = \frac{m_i}{R_i}, \forall i \in U_i \tag{6}$$

Transmission energy loss denotes as:

$$E_i^{tran} = p_i t_i^{tran}, \forall i \in U_i \tag{7}$$

The time requirement of edge computing is:

$$t_i^{edge} = \frac{f_i}{c_i}, \forall i \in U_i \tag{8}$$

The total consumption including time loss and energy consumption gives:

$$C_i^{edge} = a(t_i^{tran} + t_i^{edge}) + \beta E_i^{tran}, \forall i \in U_i \tag{9}$$

3. Migrating the computation: When the MEC server experiences excessive workload or when neighboring servers have available resources, the UE can migrate its computational tasks to the adjacent servers. This migration incurs additional transmission overhead for the UE, which involves transferring data between MEC servers. The magnitude of the transmission overhead is determined based on the allocated computational resources provided by the target server in response to the migration request from the UE, and therefore the transmission delay is:

$$t_i^{mig} = \frac{m_i}{R_i^{mig}} + Z_i^{mig}, \forall i \in U_i \tag{10}$$

Assuming that the migration cost only depends on the task size which denoted as $Z_i^{mig} = \delta m$, the transmission energy loss is:

$$E_i^{mig} = p_i t_i^{mig}, \forall i \in U_i \tag{11}$$

The computational delay and energy consumption of the UE is computed based on the resource allocation strategy of the migration server, with the computational formula remaining unchanged. Consequently, the total cost of the migration computation is expressed as:

$$C_i^{mig} = a(t_i^{mig} + t_i^{edge}) + \beta E_i^{mig}, \forall i \in U_i \tag{12}$$

Due to the limited computational capacity of the MEC server, the allocated computational resources to the UE may not be sufficient to complete the task within the maximum duration. Therefore, we assign a very low reward to such allocation strategies, allowing them to be excluded from the model iteration. Combining 1, 2, 3 we can get the expression as:

$$C_i^{ue} = \begin{cases} C_i^{local} s_i = 0 \\ C_i^{local} s_i = 0, \quad \forall i \in U_i \\ C_i^{mig} s_i = 0 \end{cases} \tag{13}$$

2.3 Problem Formulation

On the basis of the above theory, C_{all} is expressed as the sum of the total cost of local computing, edge computing, and migration computing on each UE, computation offloading and resource allocation. Our purpose is to minimize the computing overhead cost of each UE, then it can be expressed for:

$$\mathrm{C_{all}} = \sum\nolimits_{n=i}^{N} C_i^{ue},\ \forall\, i \in \mathrm{U_i} \tag{14}$$

$$\mathrm{C}_{all} = \sum\nolimits_{n=i}^{N} \mathrm{C}_i^{ue} = \sum\nolimits_{n=i}^{N} (\mathrm{C}_{\mathrm{i}}^{local} + C_i^{edge} + C_i^{mig}), \forall i \in U_i \tag{15}$$

Under the premise of minimizing $\mathrm{C_{all}}$, the problem can be described as:

$$\begin{aligned} & Min\mathrm{C_{all}} \\ & s.t.\ C1 : \sum\nolimits_{n=1}^{N} C_i \le C,\ \forall i \in U_i; \\ & C2 : a_i \in 0, 1, 2,\ \forall i \in U_i; \\ & C3 : t_i^{tran} + t_i^{edge} < t_i,\ \forall i \in U_i \end{aligned} \tag{16}$$

3 Algorithm Application

In this section we discuss the application of the algorithm within the model in detail.

3.1 Key Elements in RL

Reinforcement learning methods involve three key elements: state, action, and reward.

- State: We utilize an observation tensor as the state, which includes information about MEC Server and UE in the environment.
- Action: Since each UE is responsible for three computation tasks, there will be 3N possible offloading decisions for N UEs. Using the variable ν to express the offloading policy, which simplifies the output of the decision network.
- Reward: A reward is provided at each training step. Considering the objective of minimizing the transmission time and energy consumption for UEs, the defined reward should be negatively correlated with the objective. It can be expressed as follows:

$$\mathrm{R}(s, a) = \sum\nolimits_{n=i}^{N} \gamma (C_i^{local} - C_i^{edge}), \forall i \in U_i \tag{17}$$

3.2 DDQN Algorithm Application

In the code implementation of DDQN-based computational offloading decision, we decouple the environment module from the neural network. On one hand, it enables us to test different RL algorithms and explore various hyperparameter settings without the need to reimplement the entire environment for each test. This approach facilitates performance optimization, code maintenance and feature expansion, significantly enhancing development efficiency. On the other hand, it enhances the generality of the DDQN algorithm for computing offloading decisions in different network environments [19]. This allows the environment module to be specifically responsible for generating environment feedback and computing rewards, which are then provided to the neural network module for training, as illustrated in Fig. 4.

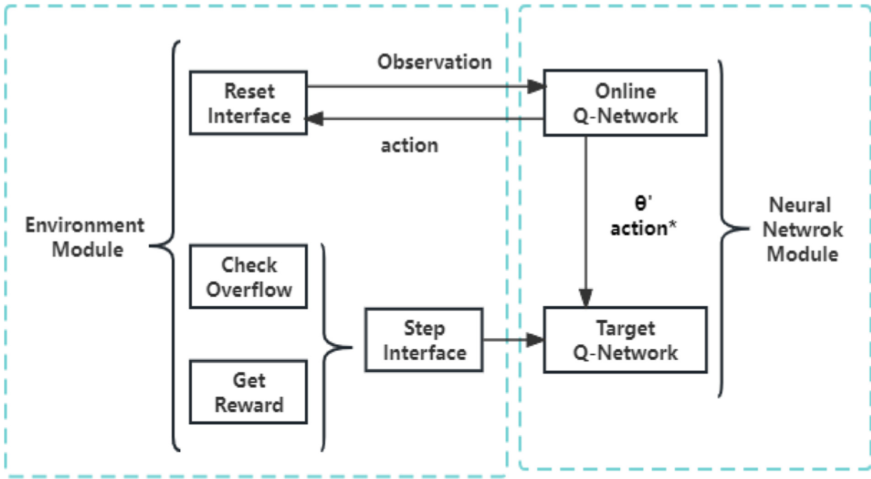

Fig. 4. The module of environment and neural network.

In the following sections, we will present the detailed implementation, functionality, and interaction process of these two modules. The environment module provides two crucial interfaces of reset and step to the neural network. The reset interface generates the network environment based on the observation tensor provided by the environment. The step interface calculates rewards based on the neural network's input actions and provides the next observation and reward for training the neural network. The environment obtains rewards for each step using the algorithm of Table 1 and 2. In the neural network module, we employ the DDQN method to train the online Q-network and target Q-network. DDQN's network update mechanism is based on the principles of Q-learning, but it mitigates the problem of overestimation bias by using two neural networks. Here are the key Eqs. (18) for DDQN network update:

$$Q_{target(s,a)} = r + \gamma Q_{current}\left(s', \underset{a}{argmax}\, Q_{current}\left(s', a\right)\right) \tag{18}$$

where Q_{target} is the target Q-value, which represents the expected reward for performing action 'a' in state 's'. 'r' is the reward obtained after performing action 'a'. 'γ' is a discount factor to balance the importance of the current and future rewards. $Q_{current}$ represents the current Q-value for selecting the best action 'a" for the current Q-value.

In the current Q-value calculation, the current state 's' and action 'a' are used to calculate the current Q-value. The mathematical representation of this process is:

$$Q_{current}(s, a) = Q_{current}(s, a) + \alpha[Q_{t\arg et}(s, a) - Q_{current}(s, a)] \tag{19}$$

$Q_{current}$ refers to the current Q-value, which serves as an estimate of the expected reward when taking action 'a' in state 's'. Q_{target} represents the Q-value that has been previously calculated and acts as the target value for the update process. The parameter 'α', often referred to as the learning rate, α's principal purpose is twofold: it regulates the extent of adaptation to the Q-value while playing a pivotal role in determining the convergence and stability of the reinforcement learning algorithm. This fine-tuning mechanism ensures the Q-values converge towards an optimal policy without the risk of overshooting.

Table 1. The neural network module

Algorithm 1 DDQN Based Computational Offloading strategy algorithm

Initialize *Online network* Q_θ, *replay buffer* $\mathcal{D}$,
Target network $Q_{\theta'}$, $\gamma \ll 1$,
*Observation_*
for each episode **do**
  for each environment step **do**
    Observe state s_t *and select* $a_t \sim \pi(s_t, a_t)$
    Execute a_t *and observe next state* s_{t+1}
    and reward $r_t = R(s_t, a_t)$
    Store (s_t, a_t, r_t, s_{t+1}) *in replay buffer* $\mathcal{D}$
  end for
  for each update step **do**
    Sample $e_t = (s_t, a_t, r_t, s_{t+1}) \sim \mathcal{D}$
    Compute target Q-value:
    $Q^*(s_t, a_t)$
    $\approx r_t + \gamma Q_\theta(s_{t+1}, argmax_{a'} Q_{\theta'}(s_{t+1}, a'))$
    Perform gradient descent step with loss:
    $||Q^*(s_t, a_t) - Q_\theta(s_t, a_t)||^2$
    Update target network parameters:
    $\theta' \leftarrow \tau * \theta + (1 - \tau) * \theta'$
  end for
  return *Observation_*
end for

These two equations form the update mechanism of the DDQN network. By alternating between the two neural networks to estimate and optimize the Q-value, DDQN can mitigate the over-estimation problem in Q-learning and thus improve the stability and performance of training. This updating process will be iterated several times during training to gradually improve the estimation of Q-values.

Table 1 and 2 show the pseudo-code of the algorithm for the two modules.

Table 2. The environment module

Algorithm 2 The environment module algorithm
require: *action*
return *reward, done, Observation*
Initialize *Observation, UE, loop*
get *action*
if overflow **then**
reward = 0
done = True
Observation ← *Observation_*
return *reward, done, Observation_*
else
get consume (action)
get consume (local)
get reward
done = False
update *Observation*
end if

4 Simulation Result

In this section, we deploy a realistic MEC system for experimentation purposes. The network model consisted of a single Base Station, three additional UE, and a single MEC server. We use Raspberry Pi 4b as the UE, with a power consumption of 6.4 W during normal operation and 2.7 W in idle mode [20]. The MEC server is equipped with a GTX 3050 GPU. All UE are placed in a distributed communication mobile self-organizing network (Ad-hoc) environment with a radius of approximately 1 m. We introduce a real-time surveillance recognition system as a computational task that can generate 20 unprocessed 256×256 images per second in the UE as a batch for a pending task. Tests are carried out to verify that all UE and MEC servers met the requirements for running the YOLOv5 algorithm for object detection.

Subsequently, we conduct an analysis of the practical performance of the DDQN-based computation offloading and resource allocation strategy in edge computing and compare it with the simple DQN offloading strategy and the fixed-parameter strategy. The user devices are uniformly distributed in an area with a radius of approximately 1 m. The architecture of deep neural network consists of an input layer, a hidden layer with 256 neurons and an output layer with 3 neurons. Besides, the activation function is ReLU, and during the weight updating process, the experience replay method and the Adam optimizer are employed.

As shown in Fig. 5, the graph demonstrates the optimal approach of the DDQN-based computation offloading and resource allocation strategy in a MEC system under long-term dynamic and complex network environments. The expected reward of the system converges to approximately -0.24 after around 460 training episodes, which outperforms the other two baseline models. As depicted, the fixed-parameter strategy experiences significant disturbances during the training process from episodes 165 to 464, exhibiting poor robustness in practical network environments. Moreover, it lacks the memory function for the environment, making it unsuitable for complex and dynamic network environments.

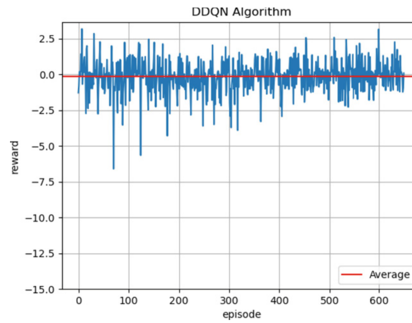

Fig. 5. The performance of DDQN method

As shown in Fig. 6 and Fig. 7 the DQN strategy starts to converge at around 570 episodes, reaching a stable and satisfactory reward value under the optimal policy. This indicates that the UE achieves relatively good performance in the environment and is less susceptible to environmental disturbances compared to the fixed-parameter strategy. However, there are still notable differences compared to the DDQN method in terms of convergence speed and action value estimation.

The Fig. 8 illustrates the dynamic changes in the average reward value during the training process. The DDQN method, without preset parameters, could select appropriate offloading actions based on the MEC system's state in the shortest possible time after a sufficient number of work events. Furthermore, it alleviates the problem of overestimating action values in the DQN method, thus improving training stability to some extent. The overall performance remains relatively stable.

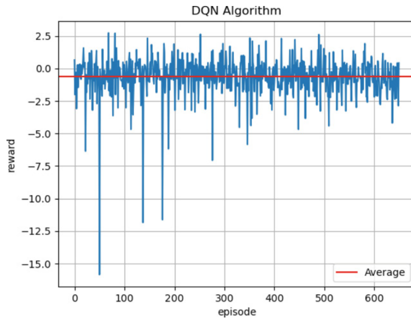

Fig. 6. The performance of DQN method

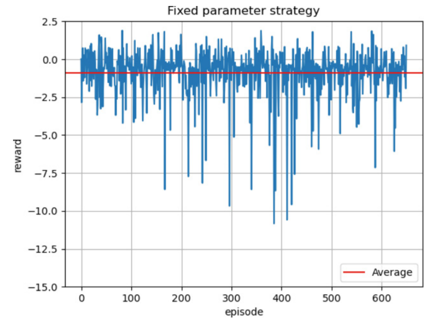

Fig. 7. The performance of Fixed-parameter method

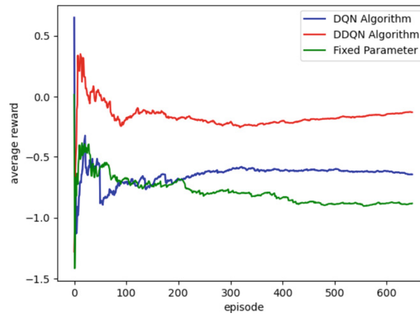

Fig. 8. The average reward of different methods

5 Conclusion

In this paper, we investigate a computational offloading strategy framework based on Double DQN, which maximizes the overall energy efficiency under transmission power and transmission delay constraints for each UE. Our approach builds upon the conventional method where the reward function is derived solely from immediate action without considering its impact on the future. Next, we employ the DQN algorithm, which leverages past learning experiences and takes into account future influences in the current action decision. However, the issue of overestimation of action values hampers the overall convergence performance and sum of rewards. Thanks to the framework of DQN, the Double DQN separates the selected actions from the target Q-value generation, thereby achieving more efficient edge computing offloading in the Mobile Edge Computing system. Extensive simulation results confirm the effectiveness of the computational offloading strategy based on DDQN. In future work, we will extend this model to networks with multiple MEC servers while experimenting with policy-based reinforcement learning algorithms to consider resource allocation.

Acknowledgments. This research was supported in part by the Inner Mongolia Science and Technology Key Project No. 2021GG0218, ROIS NII Open Collaborative Research 23S0601, and in part by JSPS KAKENHI Grant No. 21H03424.

References

1. Liu, J., Zhang, Q.: Offloading schemes in mobile edge computing for ultra-reliable low latency communications. IEEE Access **6**, 12825–12837 (2018). https://doi.org/10.1109/ACCESS.2018.2800032
2. Yang, J., Shah, A.A., Pezaros, D.: A survey of energy optimization approaches for computational task offloading and resource allocation in MEC networks. Electronics **12**(17), 3548 (2023). https://doi.org/10.3390/electronics12173548
3. Landers, M., Doryab, A.: Deep reinforcement learning verification: a survey. ACM Comput. Surv. **55**(14s), Article 330, 31 (2023). https://doi.org/10.1145/3596444
4. Kumaran, K., Sasikala, E.: Learning based latency minimization techniques in mobile edge computing (MEC) systems: a comprehensive survey. In: 2021 International Conference on System, Computation, Automation and Networking (ICSCAN), Puducherry, India, pp. 1–6 (2021). https://doi.org/10.1109/ICSCAN53069.2021.9526410
5. Liu, C.-F., Bennis, M., Poor, H.V.: Latency and reliability-aware task offloading and resource allocation for mobile edge computing. In: 2017 IEEE Globe com Workshops (GC Wkshps), Singapore, pp. 1–7 (2017). https://doi.org/10.1109/GLOCOMW.2017.8269175
6. Dab, B., Aitsaadi, N., Langar, R.: Q-learning algorithm for joint computation offloading and resource allocation in edge cloud. In: 2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM), pp. 45–52. IEEE (2019)
7. Huang, L., Feng, X., Zhang, C., et al.: Deep reinforcement learning-based joint task offloading and bandwidth allocation for multi-user mobile edge computing. Digit. Commun. Netw. **5**(1), 10–17 (2019)
8. Liang, Y., He, Y., Zhong, X.: Decentralized computation offloading and resource allocation in MEC by deep reinforcement learning. In: 2020 IEEE/CIC International Conference on Communications in China (ICCC), pp. 244–249. IEEE (2020)
9. Liang, S., Wan, H., Qin, T., et al.: Multi-user computation offloading for mobile edge computing: A deep reinforcement learning and game theory approach. In: 2020 IEEE 20th International Conference on Communication Technology (ICCT), pp. 1534–1539. IEEE (2020)
10. Wu, Y.C., Dinh, T.Q., Fu, Y., et al.: A hybrid DQN and optimization approach for strategy and resource allocation in MEC networks. IEEE Trans. Wireless Commun. **20**(7), 4282–4295 (2021)
11. Li, C., Xia, J., Liu, F., et al.: Dynamic offloading for multiuser muti-CAP MEC networks: a deep reinforcement learning approach. IEEE Trans. Veh. Technol. **70**(3), 2922–2927 (2021)
12. Gan, S., Siew, M., Xu, C., et al.: Differentially Private Deep Q-Learning for Pattern Privacy Preservation in MEC Offloading (2023). arXiv preprint arXiv:2302.04608
13. Silver, D., Huang, A., Maddison, C.J., et al.: Mastering the game of go with deep neural networks and tree search. Nature **529**(7587), 484–489 (2016)
14. Al-Absi, M.A., Al-Absi, A.A., Sain, M., et al.: Moving ad hoc networks—a comparative study. Sustainability **13**(11), 6187 (2021)
15. Jiang, P., Ergu, D., Liu, F., et al.: A review of Yolo algorithm developments. Procedia Comput. Sci. **199**, 1066–1073 (2022)
16. Nath, S., Li, Y., Wu, J., et al.: Multi-user multi-channel computation offloading and resource allocation for mobile edge computing. In: ICC 2020–2020 IEEE International Conference on Communications (ICC), pp. 1–6. IEEE (2020)
17. Silver, D., et al.: Mastering the game of go with deep neural networks and tree search. Nature **529**(7587), 484–489 (2016)
18. Hao, W., Yang, S.: Small cell cluster-based resource allocation for wireless backhaul in two-tier heterogeneous networks with massive MIMO. IEEE Trans. Veh. Technol. **67**(1), 509–523 (2017)

19. Zeng, H., Zhang, M., Xia, Y., et al.: Decoupling the depth and scope of graph neural networks. Adv. Neural. Inf. Process. Syst. **34**, 19665–19679 (2021)
20. Dennis, A.K.: Raspberry Pi Computer Architecture Essentials. Packt Publishing Ltd., Birmingham (2016)