Optimization of Single-user Task Migration based on Improved DDPG

CAO NING[1], HE YANG[2], HU CAN[1]
[1]College of Computer Science and Software Engineering,
Hohai University,
Nanjing,
CHINA

[2]College of Information Science and Engineering,
Hohai University,
Nanjing,
CHINA

Abstract: - Aiming at the problems of slow convergence and unstable convergence of traditional reinforcement learning algorithms in minimizing computational cost on edge servers with random task arrivals and time-varying wireless channels, an improved DDPG algorithm (IDDPG) was proposed. The Critic network structure of DDPG was replaced by the Dueling structure, which converged faster by splitting the state value function into an advantage function and a value function. The update frequency of the Critic network was adjusted to be higher than that of the Actor-network to make the overall training more stable. The Ornstein- Uhlenbeck noise was added to the actions selected through the Actor-network to improve the algorithm exploration ability, and the action noise size was set in segments to ensure the stability of convergence. Experimental results show that, compared with other algorithms, the IDDPG algorithm can better minimize the computational cost and has a certain improvement in the convergence speed and convergence stability.

Key-Words: - deep reinforcement learning; edge computing; task offloading; strategy optimization; network structure; algorithm optimization.

Received: July 21, 2023. Revised: May 7, 2024. Accepted: June 22, 2024. Published: July 17, 2024.

1 Introduction

With the rapid development of artificial intelligence and fifth-generation mobile communication technology, a large number of computationally intensive tasks have emerged on mobile devices, [1]. Examples include online 3D games, face recognition, and augmented or virtual reality, all of which are limited by the limited computational capabilities, [2]. Additionally, in the Internet of Things and intelligent transportation systems, [3], wireless terminal devices with information transmission capabilities need to preprocess massive amounts of sensory data. To improve user experience quality, Mobile Edge Computing (MEC) technology, [4], has been proposed to bridge the gap between the limited computational capabilities of terminal devices and the massive computational demand. Task offloading, as a core technology of MEC, can offload computational tasks from mobile devices to MEC servers close to base stations (BS). Rational offloading strategies can not only minimize transmission latency but also reduce the transmission energy consumption of MEC servers, efficiently completing massive computational tasks. To achieve better user experience and higher transmission efficiency, the research on task offloading strategies in MEC has been widely studied.

For short-term optimization on quasi-static channels, literature [5], studied the optimal offloading and resource allocation on software-defined ultra-dense networks (SD-UDN). Literature [6], studied from the perspective of controlling the CPU working frequency, aiming to minimize computational energy consumption and execution time. Literature [7], discussed the trade-off between energy and delay in environments with sensitive delay and limited energy, and further improved the allocation of computational resources and communication. Literature [8], improved the performance of MEC further by using emerging technologies such as wireless power transfer and non-orthogonal multiple access. For stochastic task arrivals and time-varying wireless channels, dynamic joint control of radio and computational resources has also been studied. In the single-user scenario, literature [9], considered the energy consumption in

single-antenna mobile devices as the research objective, optimizing the time and power required for offloading. Literature [10], used heuristic algorithms to solve the dynamic task offloading problem in simple scenarios. Markov decision process (MDP) can also be used for dynamic control analysis and design of computation offloading, [11]. In addition, literature [12], demonstrated how to learn dynamic task offloading strategies through reinforcement learning (RL) algorithms without prior knowledge of the system. However, traditional RL algorithms can only solve small-scale problems, and they are often powerless in the face of increased complexity and dimensionality of the state space, [13]. The association between deep nonlinear network structures and reinforcement learning makes Deep Reinforcement Learning (DRL) have end-to-end perception and control, thus solving the problem of space explosion, [14]. In the MEC system, online resource allocation and scheduling based on DRL algorithms have been studied. Literature [15], designed an online offloading algorithm to maximize the weighted sum computation rate in a wireless power supply system. Literature [16], proposed a DRL-based task offloading strategy to select a MEC server for offloading and determine the offloading rate. Literature [17], presented a policy computation offloading algorithm based on deep Q-network (DQN), where mobile devices learn the optimal task offloading and energy allocation based on the task queue state, energy queue state, and channel quality, aiming to maximize long-term utility. Literature [18], proposed using Deep Deterministic Policy Gradient (DDPG) as the offloading strategy algorithm to solve the allocation problem of energy consumption and delay in a single-user scenario with stochastic task arrivals and time-varying wireless channel models, aiming to minimize computational cost. Literature [19], proposed the ECOO algorithm based on DDPG for optimizing candidate networks, solving the problem of stochastic task offloading. Literature [20], used DDPG to jointly optimize service cache placement, task offloading decisions, and resource allocation.

In summary, in the single-user scenario, the current dynamic computation offloading strategies based on DRL only utilize traditional reinforcement learning algorithms. However, these algorithms, such as Deep Q-Network (DQN) and Deep Deterministic Policy Gradient (DDPG), suffer from slow convergence and unstable convergence, leading to high computational cost and latency in the system as the base station collects information and allocates it to the user. Therefore, for a single-user MEC system, a more efficient task offloading strategy is needed.

In this paper, a system consisting of a single-user MEC server and a base station connected to it is constructed, where tasks arrive randomly and the user's channel conditions are time-varying. The mobile user independently learns a dynamic computation offloading strategy based on local observations of the system. An improved Deep Deterministic Policy Gradient algorithm, called IDDGP, is proposed, which operates in a continuous action space and enables more efficient local execution and task offloading power control, to minimize computational cost in the long term.

2 System Model

This article presents a system consisting of a BS(with N antennas, a MEC server, and a group of mobile users. The system adopts a discrete-time model, where each operation $m \in \{1, 2, ..., M\}$is set to a fixed time interval τ_0. The time slots are indexed by $t \in \{0, 1, ..., T\}$. Due to the varying channel conditions and task arrivals at each t, it is necessary to calculate the proportion of local execution and computation offloading to balance the average energy consumption and latency of task processing, thus achieving cost minimization.

2.1 Network Model

For each time slot t, the received signal $\boldsymbol{X}(t)$ at the base station can be represented as follows:

$$X(t) = \sum_{m=1}^{M} h_m(t)\sqrt{p_{o,m}(t)}s_m(t) + n(t) \quad (1)$$

where $p_{o,m}(t) \in [0, P_{o,m}]$ represents the transmission power when user m migrates its task, $s_m(t)$ represents the $N \times 1$ dimensional transmitted signal, and $n(t) \sim \mathcal{CN}(0, \sigma_R^2 I_N)$ is an additive Gaussian white noise vector with variance σ_R^2.

$\boldsymbol{h}_m(t)$ is used to characterize the correlation between user m and time slot t, and its definition can be found in [21]. $\boldsymbol{H}(t) = [\boldsymbol{h}_1(t), \cdots, \boldsymbol{h}_m(t)]$ represents the $N \times M$ channel matrix between the mobile users and the base station. The base station detector can represent the channel matrix's pseudoinverse as $\mathbf{H}^{\dagger}(t) = \left(\mathbf{H}^H(t)\mathbf{H}(t)\right)^{-1}\mathbf{H}^H(t)$. If the m-th row of $\mathbf{H}^{\dagger}(t)$ is represented as $\mathbf{g}_m^H(t)$, then $\mathbf{g}_m^H(t)\mathbf{X}(t) = \sqrt{p_{o,m}(t)}s_m(t) + g_m^H(t)\boldsymbol{n}(t)$ represents the signal detected by the base station for user m. Therefore, the corresponding signal-to-noise ratio can be derived as shown in (2).

$$\gamma_m(t) = \frac{p_{o,m}(t)}{\sigma_R^2\|\boldsymbol{g}_m(t)\|^2} = \frac{p_{o,m}(t)}{\sigma_R^2\left[\left(\boldsymbol{H}^H(t)\boldsymbol{H}(t)\right)^{-1}\right]_{mm}} \quad (2)$$

2.2 Computation Model

We assume that the applications are fine-grained, [22]. The definitions of $d_{l,m}(t)$ and $d_{o,m}(t)$ can be found in reference [21], while the definition of $B_m(t)$ is described in [20]. The relationship between $B_m(t)$ and the task computation load is given by (3) as shown in [20].

$$B_m(t+1) = \left[B_m(t) - \left(d_{l,m}(t) + d_{o,m}(t)\right)\right]^+ + a_m(t) \tag{3}$$

Where $B_m(0) = 0$, and $a_m(t)$ represents the arrival task load.

The definitions of $P_{l,m}(t)$ and Lm can be found in [3]. By using the DVFS technique, [23], to adjust the chip voltage, the CPU frequency of the local device is determined as shown in (4), [20]:

$$f_m(t) = \sqrt[3]{p_{l,m}(t)/k} \tag{4}$$

where k represents the effective switching capacitance of the chip.

The amount of local data processed by mobile device m at time t is given by (5):

$$d_{l,m}(t) = \tau_0 f_m(t) L_m^{-1} \tag{5}$$

The amount of migration data processed by mobile device m at time t is given by (6):

$$d_{o,m}(t) = \tau_0 W \log_2\left(1 + \gamma_m(t)\right) \tag{6}$$

where W is the system bandwidth.

3 DRL-based IDDPG

This paper proposes a DRL-based IDDPG algorithm that enables mobile devices to learn computation offloading policies dynamically, minimizing the computational cost of energy consumption and buffer delay in the MEC system. The proposed strategy observes the environment from its perspective and selects actions to allocate power for local execution and computation offloading. The following sections define the DRL framework, followed by an introduction to the DDPG algorithm and the Dueling network architecture. Finally, the proposed IDDPG algorithm is presented.

3.1 Deep Reinforcement Learning

State space: It is assumed that the state of the mobile device is determined by its local observations of the system. At the beginning of time interval t, the mobile device updates the queue length of the data buffer using (3) and estimates the incoming uplink channel vector $\boldsymbol{h}_m(t)$ by receiving the signal-to-noise ratio $\gamma_m(t-1)$ from the previous wireless communication with the MEC server. The current state of the mobile device is represented as follows:

$$s_{m,t} = \left[B_m(t), \phi_m(t-1), \boldsymbol{h}_m(t)\right] \tag{7}$$

where $\phi_m(t) = \frac{\gamma_m(t)\sigma_R^2}{p_{o,m}(t)\|\boldsymbol{h}_m(t)\|^2}$.

Action space: The mobile device selects an action $a_{m,t}$ based on the observed state $s_{m,t}$, as represented by (8).

$$a_{m,t} = \left[p_{l,m}(t), p_{o,m}(t)\right] \tag{8}$$

Reward function: The definition of the relationship between energy consumption and delay can be found in [24]. Based on this relationship, the reward function $r_{m,t}$, received by the mobile device after time slot t is defined as follows:

$$r_{m,t} = -\omega_{m,1}\left(p_{l,m}(t) + p_{o,m}(t)\right) - \omega_{m,2} B_m(t) \tag{9}$$

Where $\omega_{m,1}$ and $\omega_{m,2}$ are non-negative weighting factors, and $\omega_{m,1} = 1 - \omega_{m,2}$. The meaning of the reward function $r_{m,t}$ is the negative weighted sum of the total energy consumption and the length of the task buffer queue at time t. The weighted sum of energy consumption and delay represents the computational cost required for task offloading. The optimization objective of this paper is to minimize the computational cost, which is equivalent to maximizing the reward. By setting different values for the weighting factors $\omega_{m,1}$ and $\omega_{m,2}$, dynamic adjustments can be made between the energy consumption and task execution delay in the task migration strategy.

3.2 DDPG

DDPG, [25], is a deterministic policy algorithm that utilizes deep learning techniques and is based on the Actor-Critic algorithm. This algorithm uses deep neural networks to establish approximate functions. It directly generates deterministic actions from the Actor-network, evaluates the actions using the Critic network, and guides the Actor network in selecting the next action. Additionally, DDPG maintains a set of parameters for both the Actor and Critic networks, which are used to calculate the expected value of action values for improved stability and enhancement of the Critic's policy guidance level. The networks that use the backup parameters are referred to as target networks, and their corresponding parameters are updated with small increments each time. Another set of parameters,

corresponding to the Actor and Critic, is used to generate actual interactive actions and calculate the respective policy gradients, and these parameters are updated after each learning iteration. This dual-parameter setup aims to reduce non-convergence occurrence due to the guidance of approximate data. The specific usage scenarios for these four networks are as follows:

Actor-network: The Actor network iteratively updates its parameter θ. It generates specific actions based on the current state s and interacts with the environment to generate s', r.

Target Actor network: The target Actor network periodically copies the parameters $\theta \rightarrow \theta'$. It selects the optimal exploratory action a' based on the subsequent state s' provided by the environment.

Critic network: The Critic network iteratively updates its parameter ω. It calculates the current action value corresponding to the state s and the generated action a.

Target Critic network: The target Critic network periodically copies the parameters $\omega \rightarrow \omega'$. It calculates the target action value based on the subsequent state s' and a'.

DDPG uses soft updates, where a portion of the parameters is updated proportionally each time to ensure training stability. The update expression is shown as follows:

$$\begin{aligned} w' &\leftarrow \tau w + (1-\tau) w' \\ \theta' &\leftarrow \tau \theta + (1-\tau) \theta' \quad \tau \ll 1 \end{aligned} \tag{10}$$

In action selection, DDPG does not use the ε-greedy approach as in DQN. Instead, it adds a certain amount of noise $\mathcal{N}$ to the chosen action A to increase randomness and learning coverage. The expression is shown as follows:

$$A = \pi_\theta(s) + \mathcal{N} \tag{11}$$

where $\pi_\theta(s)$ represents the action policy on the Actor-network.

For the current Critic network, the mean square error loss function is defined as shown in (12):

$$J(\omega) = \frac{1}{m} \sum_{j=1}^{m} \left(y_j - Q(\phi(s_j), a, \omega)\right)^2 \tag{12}$$

where $y_j = r_j + \gamma Q'(\phi(s_j'), \pi_{\theta'}(\phi(s_j')), \omega')$ serves as the Critic target Q-value. The current network parameters ω are updated using gradient backpropagation.

For the current Actor-network, the loss gradient is defined as shown as follows:

$$J(\theta) = -\frac{1}{m} \sum_{j=1}^{m} Q(s_j, a_j, \omega) \tag{13}$$

All parameters θ of the current Actor network are updated through gradient backpropagation using a neural network.

3.3 Dueling Network Architecture

In reinforcement learning, the agent needs to estimate the value for each state. However, for many states, the agent does not need to change its action to adapt to the new state. Therefore, evaluating the value of such state-action pairs could be more efficient and meaningful, [9]. The Dueling network architecture separates the state value and the action advantage to evaluate, avoiding unnecessary action evaluations and enabling more accurate estimation of Q-values with faster convergence speed.

This paper incorporates the Dueling network structure into the DDPG Critic main network and target network, called the Dueling-Critic network. The Dueling-Critic network separately evaluates the observed state value and the action advantage transmitted from the Actor-network. This network allows finding stable policies in continuous action spaces and speeds up convergence. The structure of the Dueling-Critic network in this paper is illustrated in Figure 1.

State-action value function $Q^\pi(s,a)$ represents the expected return value when action a is chosen by policy π in state s, while state value function $V^\pi(s)$ represents the expected return value generated by policy π in state s. The difference between the two represents the advantage of choosing action a in state s, as defined in [9].

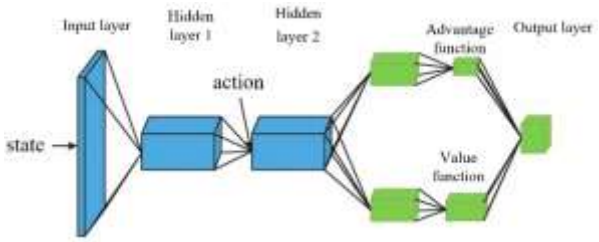

Fig. 1: Dueling-Critic network architecture

The output of the Dueling-Critic network consists of the state value $V(s, \omega, \alpha)$ and the action advantage $A(s, a, \omega, \beta)$. ω represents the parameters of the Dueling-Critic network, while α and β represent the parameters of the value function and the action advantage function , [9]. The output of the deep Q-network with the competitive network structure is shown as follows:

$$Q(s, a, \omega, \alpha, \beta) = V(s, \omega, \alpha) + A(s, a, \omega, \beta) \tag{14}$$

Since the network directly outputs Q values without explicitly knowing the state value V and

action advantage A, to ensure interpretability, the action advantage A is centrally processed, which improves optimization stability while maintaining performance. The modified Q value is defined as follows:

$$Q(s,a,\omega,\alpha,\beta)=V(s,\omega,\alpha)+\left(A(s,a,\omega,\beta)-\frac{1}{|\mathcal{A}|}\sum_{a'}A(s,a',\omega,\beta)\right) \quad (15)$$

3.4 Delayed Policy Updates

In DDPG, the Critic network extracts a portion of experiences $\{(s_j,a_j,r_j,s_j')\},j=1,2,\cdots,m$ from the replay buffer B_m as the current Q-values for calculation. Then, the mean squared error is computed to estimate the Q-values, and the current network parameters of the Critic are updated through gradient backpropagation. These updated parameters are then transferred to the target network at a fixed interval τ. Similarly, the Actor network shares the same network structure as the Critic, so it synchronizes the parameter updates with the Critic at the fixed interval τ.

However, synchronous updates may cause issues when the Actor-network reaches its optimal point during training while the Q-values of the Critic have been updated. In this case, the Q-values here could be more optimal. The Actor network can only continue searching for new optimal points, which increases the training duration. In the worst case, the Actor-network may get trapped in suboptimal points and fail to find the correct optimal point, leading to poor training of both the Critic and Actor networks and making it difficult for the model to converge.

This paper introduces asynchronous updates for the Critic and Actor networks by setting τ_1 and τ_2 as the soft update coefficients for the Actor and Critic networks, respectively, and adjusting the update frequency of the Critic network to be faster than the Actor-network. The purpose is to stabilize the Critic before the Actor proceeds with the next learning step. Delayed updates not only reduce unnecessary repeated updates but also minimize accumulated errors in multiple updates.

3.5 Noise Segmentation

Reinforcement learning often struggles with the exploration-exploitation trade-off. If an agent always selects actions based on the maximum Q-value, it can hardly learn the optimal policy in complex environments. This is because such an action selection strategy lacks exploration, and some actions may never be explored, leading to getting stuck in suboptimal situations. Common methods to address the exploration-exploitation dilemma include:

1. ε-greedy: With a probability of (1-ε), the agent greedily selects the action that corresponds to the currently perceived maximum Q-value, and with a probability of ε, it randomly selects an action from all available options.
2. Decaying ε-greedy: As time progresses, the probability of selecting a random action ε decreases.
3. Uncertainty-based exploration: When the value of an action is uncertain, the agent has a higher probability of choosing that action.
4. Information value-based exploration: Approximating the value and function and constructing a model based on the information state to sample and approximate the solution.
5. Adding Gaussian noise or Ornstein-Uhlenbeck (OU) exploration noise to action selection, where OU noise is defined in [5], to enhance exploration.

In this paper, due to the small-time granularity of the system, the agent selects OU exploration noise. To improve the efficiency of exploration, this paper segments the magnitude of the OU noise, allowing it to have different levels of exploration at different stages of training. In the early stages of training, the agent needs to explore a wide range of actions in unknown environments, so a higher level of exploration is required. As the agent interacts with the environment and improves its learning ability, the exploration level gradually decreases with the number of steps.

This paper provides the settings for the noise magnitude as shown in Table 1.

Table 1. Parameter configuration

Index	Step number	Noise magnitude
1	0<Step<60000	noise_sigma=0.2
2	60000≤Step<150000	noise_sigma=0.12
3	150000≤Step<300000	noise_sigma=0.08
4	300000≤Step<400000	noise_sigma=0.03

3.6 IDDPG

Algorithm 1: IDDPG

Input: Actor current network θ, Actor target network θ', Critic current network ω, Critic target network ω', discount factor γ, soft update coefficients τ_1 and τ_2, the batch size for gradient descent m, value function network parameters α, advantage function network parameters β, maximum iterations for target network T_{max}

Output: The optimal parameters for the current Actor network θ and the current Critic network ω

1 Randomly initialize the Actor-network and the Dueling-Critic network, initialize the weights of the target networks $\theta'\leftarrow\theta$, $\omega'\leftarrow\omega$, $\alpha'\leftarrow\alpha$, $\beta'\leftarrow\beta$, initialize the experience replay buffer B_m, and initialize the exploration noise process $\Delta\mu$

2 for *episode=1* to *episode_max* do

3 Initialize state s
4 for $t=1$ to T_{max} do
5 Observe the current state s and select an action a by using the current policy network in conjunction with the segmented exploration noise $\Delta\mu$
6 Take action a, receive reward r, and then observe the next state s'
7 Collect the set of (s, a, r, s') for the current execution and store it in the buffer B_m
8 Randomly sample m experiences $\{(s_j, a_j, r_j, s_j')\}$ from B_m to create a mini-batch
9 Update the Dueling-Critic network by minimizing the loss function L using the sampled mini-batch

$$L = \frac{1}{m}\sum_{j=1}^{m}\left(r_j + \gamma Q'\left(\phi(s_j'),\pi_{\vartheta'}\left(\phi(S_j')\right),\omega',\alpha,\beta\right) - Q(\phi(s_j),a_j,\omega,\alpha,\beta)\right)^2$$

10 Update the Actor-network using the sample gradient policy

$$J(\theta) = -\frac{1}{m}\sum_{j=1}^{m} Q(s_j,a_j,\omega,\alpha,\beta)$$

11 Soft update the Dueling-Critic target network and the Actor target network ($\tau_1\tau_2<<1$).

$$\theta' \leftarrow \tau_1\theta + (1-\tau_1)\theta'$$
$$\omega' \leftarrow \tau_2\omega + (1-\tau_2)\omega'$$
$$\alpha' \leftarrow \tau_2\alpha + (1-\tau_2)\alpha'$$
$$\beta' \leftarrow \tau_2\beta + (1-\tau_2)\beta'$$

12 end for
13 end for

In the DRL-based IDDPG problem model, MEC observes the environment and obtains an initial state space s as (7). These high-dimensional states are processed through deep neural networks to output Q-value functions or policies. As an edge server, the MEC selects actions a for local offloading power and task migration power based on the current policy network and segmented exploration noise $\Delta\mu$ as shown in (8). Then, the reward value r based on (9) is returned to the MEC, and the next state s' is observed. The MEC collects the set (s, a, r, s') for the current execution and stores it in the buffer B_m. Afterward, a mini-batch of m experiences is randomly sampled from B_m, and the Dueling-Critic network $Q(s, a, \omega, \alpha, \beta)$ is updated by minimizing the loss function using the sampled gradient. The Actor network is updated using the sample gradient. The MEC receives the reward value r corresponding to each action selection, and trains and improves the network model to output the optimal policy that maximizes the reward, which means minimizing the MEC task offloading computational cost.

4 Simulation

To assess the practicality of the IDDPG algorithm in a single-user MEC model. experimental simulations were conducted in an environment with Windows 10, CPU 8500, GTX 1080, Python 3.7, and TensorFlow 1.5. Four algorithms were compared: Greedy local offloading (GD-Local), [20], Greedy migration offloading (GD-Offload), [20], Discrete action DQN, [11], Continuous action DDPG, [25].

4.1 Setup

In the MEC system, the time interval τ_0 is set to 1 ms. At the beginning of each episode, the channel vectors $\boldsymbol{h}_m(0) \sim \mathcal{CN}(0, \boldsymbol{h}_0(d_0/d_m)^a I_N)$ are initialized, where the reference distance d_0=1 m, the path loss exponent α=3, and d represents the distance between the mobile device and the MEC server, which is set to 100m in this paper. The channel correlation coefficient p_m=0.95, the bandwidth W=1 MHz, the maximum transmission power of the mobile device $p_{o,m}$=2 W, the noise power σ_R^2=10^{-9} W, the local execution parameter k=10^{-27}, the CPU cycle L_m=500 cycles/bit, the CPU cycle frequency F_m=1.26 GHz, and the maximum power for local execution $p_{l,m}$=2 W.

To validate the effectiveness of the proposed algorithm, the same neural network architecture and parameters are used for DQN, DDPG, and ID-DPG. The number of neurons in the hidden layers is set to 400 and 300, respectively. In IDDPG, the last hidden layer of the dueling-critic network is split into an advantage function and a value function. ReLU is used as the activation function for the hidden layers of the neural network, and the actions outputted by the Actor-network are mapped using a sigmoid function. The neural network parameter settings are shown in Table 2.

Table 2. Network parameter setting

Index	Parameter	Value
1	Actor learning rate	0.001
2	Critic learning rate	0.0005
3	Actor soft update coefficient τ_1	0.001
4	Critic soft update coefficient τ_2	0.15
5	Noise parameter θ	0.12
6	Noise parameter σ	2.5×10^5

4.2 Training

The distance between the mobile device and the BS is set to d_1=100. As shown in Figure 2, Figure 3 and Figure 4, the training process for dynamic offloading in the single-user scenario is conducted with different values of ω_1. In Figure 2, the left plot corresponds to ω_1=0.5, and the right plot corresponds to ω_1=0.8, where *episode_max*=2000,

T_{max}=200. Each curve represents the average value obtained from running the numerical simulation 10 times.

In each plot, two cases are shown, with task arrival rates λ=2.0 Mbps and λ=3.0 Mbps. When ω_1=0.5, which represents a balance between delay and energy consumption, although the average energy consumption in Figure 3(left) is not the lowest, the proposed IDDPG algorithm achieves higher average rewards in Figure 2(left) compared to the DDPG and DQN algorithms, by adjusting the average delay in Figure 4(left).

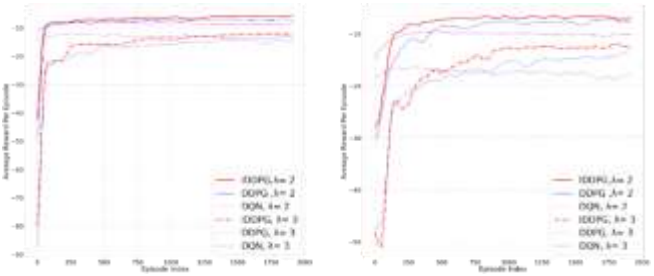

Fig. 2: Average reward values for different task arrival rates with different trade-off factors

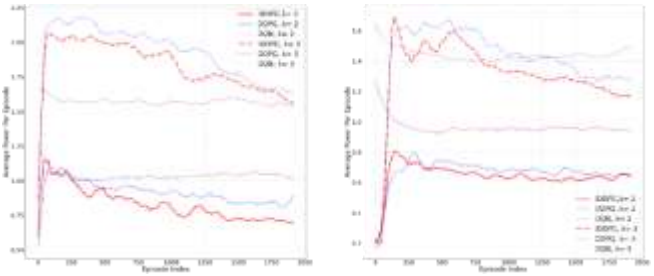

Fig. 3: Average energy consumption values for different task arrival rates with different trade-off factors

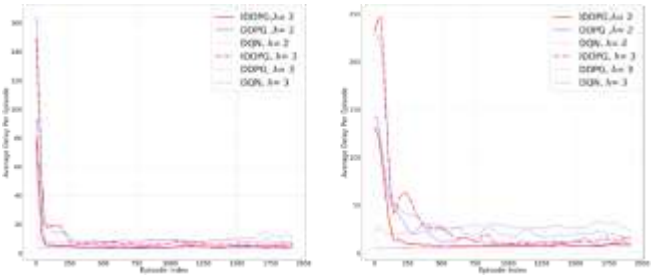

Fig. 4: Average delay values for different task arrival rates with different trade-off factors

When ω_1=0.8, which indicates a higher emphasis on energy consumption, the IDDPG algorithm achieves lower average energy consumption in Figure 3(right) compared to the compared algorithms. Although there is a sacrifice in average delay in Figure 4(right), the IDDPG algorithm still achieves higher average rewards in Figure 2(right) for task arrival rates λ=2.0 Mbps and λ=3.0 Mbps compared to the DDPG and DQN algorithms.

Through comparative experiments, it can be observed that the proposed IDDPG algorithm, through interaction between the user agent and the environment, achieves higher average rewards than the compared algorithms under different trade-off factors, and demonstrates greater stability. This indicates that for continuous control problems, the IDDPG algorithm is capable of better "exploration-exploitation" and more efficient maximization of rewards, i.e., minimizing computational costs, while learning the optimal computation offloading strategy.

The IDDPG algorithm demonstrates superior performance in terms of average rewards and convergence speed compared to the other algorithms, regardless of the task arrival rate (λ = 2.0 Mbps or λ = 3.0 Mbps) when the trade-off factor ω_1 is set to 0.8. Furthermore, the advantage of the IDDPG algorithm over the DDPG and DQN algorithms is even greater when compared to the weighting factor ω_1=0.5. This indicates that the IDDPG algorithm is more suitable for scenarios that prioritize power consumption. In situations with more complex task computations, the IDDPG algorithm demonstrates superior performance due to its reasonable action exploration and network structure optimization.

4.3 Test

After training for a total of 2000 episodes, we obtained dynamic computation offloading strategies learned by the IDDPG, DDPG, and DQN algorithms, respectively. To compare the performance of different strategies, we conducted numerical simulations in 100 episodes with a test step size of T=10000. The average values were taken from 10 simulations for each of the five task arrival rates: λ=1.0 Mbps, λ=1.5 Mbps, λ=2.0 Mbps, λ=2.5 Mbps, and λ=3.0 Mbps.

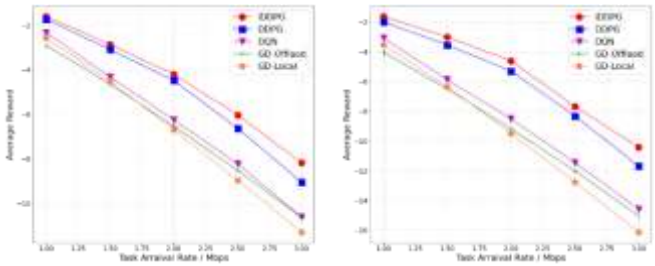

Fig. 5: Average reward value for different task arrival rates with different ω_1

In the scenario of ω_1 =0.5 in Figure 5(left), when λ=3.0 Mbps, the average reward of the DQN-based strategy is approximately equal to the greedy

migration offloading. This is because the DQN strategy has a limited number of discrete power levels, and although finer granularity in the action space discretization may lead to better performance, the number of operations increases exponentially with the degrees of freedom, making efficient exploration more challenging and thus reducing the performance of the DQN strategy.

From Figure 5, it can be observed that as the task arrival rate increases, the average rewards of all the algorithms gradually decrease. This is because higher demands lead to increased computational costs (average rewards in this paper are defined as the negative value of costs). In the case of low arrival rates, such as λ=1.0 Mbps and λ=1.5 Mbps, the IDDPG algorithm shows little difference in average rewards compared to the DDPG algorithm but performs better than the DQN and other greedy algorithms. This is because the computational workload is small and does not require deep exploration in actions. The improved network structure and exploration noise advantage in IDDPG are not prominent. However, as the arrival rate increases, the IDDPG algorithm gradually demonstrates its advantages in terms of faster and more stable convergence.

In summary, the IDDPG algorithm outperforms the compared strategies in terms of average rewards for different task arrival rates under different trade-off factors. This indicates that the IDDPG strategy can allocate local execution power and task offloading power more reasonably based on observations of the environment in unknown information settings, thereby balancing energy consumption and latency and achieving the goal of minimizing long-term costs, highlighting the superiority of this strategy.

5 Conclusion

In this paper, an improved DDPG algorithm is proposed to address the slow convergence and instability issues in task offloading of traditional single-user MEC systems using reinforcement learning algorithms. The improved DDPG algorithm aims to minimize computational costs more efficiently. Through experimental validation, our proposed algorithm has been shown to outperform traditional DDPG, DQN, and other greedy strategies in terms of power consumption and buffer delay. The main innovation of this paper lies in replacing the Critic network structure of DDPG with a Dueling structure. This structure decomposes the state value function into the advantage function and value function, ensuring faster convergence speed and higher stability. Additionally, we set the update frequency of the Critic network higher than that of the Actor-network to stabilize the Critic network's training and improve the Actor network's action output. Furthermore, we segment the action noise, initially setting it at a higher value to ensure comprehensive exploration during the initial training stage. As the training progresses and stabilizes, we gradually reduce the noise level to ensure convergence stability.In our future work, we plan to incorporate multi-user collaboration into the IDDPG framework to further enhance task offloading in MEC systems and improve strategy performance.

References:

[1] SHI Weisong, ZHANG Xingzhou, WANG Yifan, ZHANG Qingyang. Edge computing: State-of-the-art and future directions. *Journal of Computer Research and Development*, 2019, 56 (1):69-89 (in Chinese).

[2] W. Shi, J. Cao, Q. Zhang, Y. Li and L. Xu, "Edge Computing: Vision and Challenges," in *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637-646, Oct. 2016, doi: 10.1109/JIOT.2016.257919 8

[3] K. Zhang, Y. Mao, S. Leng, Y. He and Y. ZHANG, "Mobile-Edge Computing for Vehicular Networks: A Promising Network Paradigm with Predictive Off-Loading," in *IEEE Vehicular Technology Magazine*, vol. 12, no. 2, pp. 36-44, June 2017, doi: 10.1109/MVT. 2017.2668838

[4] Y. Mao, C. You, J. Zhang, K. Huang and K. B. Letaief, "A Survey on Mobile Edge Computing: The Communication Perspective," in *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2322-2358, Fourthquarter 2017, doi: 10.1109/COMST.2017.2745201

[5] Maria J.P. Peixoto and Akramul Azim. 2021. Using time-correlated noise to encourage exploration and improve autonomous agents performance in *Reinforcement Learning. Procedia Comput. Sci.*, 191, C (2021), 85-92. https://doi.org/10.1016/j. procs.2021.07.014.

[6] M. Chen and Y. Hao, "Task Offloading for Mobile Edge Computing in Software Defined Ultra-Dense Network," in *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 3, pp. 587-597, March 2018, doi: 10.1109/JSAC.201 8.2815360.

[7] H. Guo, J. Liu, J. Zhang, W. Sun and N. Kato, "Mobile-Edge Computation Offloading for

Ultradense IoT Networks," in *IEEE Internet of Things Journal*, vol. 5, no. 6, pp. 4977-4988, Dec. 2018, doi: 10.1109/JIOT.2018.2838584.

[8] J. Zhang et al., "Energy-Latency Tradeoff for Energy-Aware Offloading in Mobile Edge Computing Networks," in *IEEE Internet of Things Journal*, vol. 5, no. 4, pp. 2633-2645, Aug. 2018, doi: 10.1109/JIOT.2017.2786343.

[9] S. Bi and Y. J. Zhang, "Computation Rate Maximization for Wireless Powered Mobile-Edge Computing With Binary Computation Offloading," in *IEEE Transactions on Wireless Communications*, vol. 17, no. 6, pp. 4177-4190, June 2018, doi: 10.1109/TWC.201 8.2821664.

[10] J. Kwak, Y. Kim, J. Lee and S. Chong, "DREAM: Dynamic Resource and Task Allocation for Energy Minimization in Mobile Cloud Systems," in *IEEE Journal on Selected Areas in Communications*, vol. 33, no. 12, pp. 2510-2523, Dec. 2015, doi: 10.1109/JSAC.2015.2478718.

[11] N. Janatian, I. Stupia and L. Vandendorpe, "Optimal resource allocation in ultra-low power fog-computing SWIPT-based networks," *2018 IEEE Wireless Communications and Networking Conference (WCNC)*, Barcelona, Spain, 2018, pp. 1-6, doi: 10.1109/WCNC.2018.8376974.

[12] M. Qin, L. Chen, N. Zhao, Y. Chen, F. R. Yu and G. Wei, "Power-Constrained Edge Computing With Maximum Processing Capacity for IoT Networks," in *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4330-4343, June 2019, doi: 10.1109/JIOT.2018.2875218.

[13] J. Liu, Y. Mao, J. Zhang and K. B. Letaief, "Delay-optimal computation task scheduling for mobile-edge computing systems," 2016 *IEEE International Symposium on Information Theory (ISIT)*, Barcelona, Spain, 2016, pp. 1451-1455, doi: 10.1109/ISIT.201 6.7541539.

[14] T. Q. Dinh, Q. D. La, T. Q. S. Quek and H. Shin, "Learning for Computation Offloading in Mobile Edge Computing," in *IEEE Transactions on Communications*, vol. 66, no. 12, pp. 6353-6367, Dec. 2018, doi: 10.1109/TCOMM.201 8 .2866572.

[15] Sutton R S, Barto A G. Reinforcement learning: An introduction. MIT press, 2018.

[16] Mnih, V., Kavukcuoglu, K., Silver, D. et al. Human-level control through deep reinforcement learning. *Nature*, 518, 529–533 (2015), https://doi.org/10.1038/nature14236.

[17] L. Huang, S. Bi and Y. -J. A. Zhang, "Deep Reinforcement Learning for Online Computation Offloading in Wireless Powered Mobile-Edge Computing Networks," in *IEEE Transactions on Mobile Computing*, vol. 19, no. 11, pp. 2581-2593, 1 Nov. 2020, doi: 10.1109/TMC.2019.2928811.

[18] M. Min, L. Xiao, Y. Chen, P. Cheng, D. Wu and W. Zhuang, "Learning-Based Computation Offloading for IoT Devices With Energy Harvesting," in *IEEE Transactions on Vehicular Technology*, vol. 68, no. 2, pp. 1930-1941, Feb. 2019, doi: 10.1109/TVT.2018.2890685.

[19] X. Chen, H. Zhang, C. Wu, S. Mao, Y. Ji and M. Bennis, "Optimized Computation Offloading Performance in Virtual Edge Computing Systems Via Deep Reinforcement Learning," in *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4005-4018, June 2019, doi: 10.1109/JIOT.2018.2876279.

[20] Chen, Z., Wang, X. Decentralized computation offloading for multi-user mobile edge computing: a deep reinforcement learning approach. *J. Wireless Com Network*, 2020, 188 (2020), https://doi.org/10.1186/s13638-020-01801-6.

[21] H. Lu, C. Gu, F. Luo, W. Ding, S. Zheng and Y. Shen, "Optimization of Task Offloading Strategy for Mobile Edge Computing Based on Multi-Agent Deep Reinforcement Learning," in *IEEE Access*, vol. 8, pp. 202573-202584, 2020, doi: 10.1109/ACCESS.2020.3036416.

[22] Y. Yang, K. Wang, G. Zhang, X. Chen, X. Luo and M. -T. Zhou, "MEETS: Maximal Energy Efficient Task Scheduling in Homogeneous Fog Networks," in *IEEE Internet of Things Journal*, vol. 5, no. 5, pp. 4076-4087, Oct. 2018, doi: 10.1109/JIOT.2018.2846644.

[23] Asghari, A., Sohrabi, M.K. Combined use of coral reefs optimization and multi-agent deep Q-network for energy-aware resource provisioning in cloud data centers using DVFS technique. *Cluster Comput*., 25, 119–140 (2022), https://doi.org/10.1007/s10586-021-03368-3.

[24] Shortle JF, Thompson JM, Gross D, et al. *Fundamentals of queueing theory*. New York: John Wiley & Sons, 2018:1-576.

[25] Xu, Yi-Han, et al. "Deep Deterministic Policy Gradient (DDPG)-Based Resource Allocation Scheme for NOMA Vehicular Communications." *IEEE Access*, Jan. 2020, pp. 18797–807, https://doi.org/10.1109/access.2020.2968595.

Contribution of Individual Authors to the Creation of a Scientific Article (Ghostwriting Policy)

The authors equally contributed in the present research, at all stages from the formulation of the problem to the final findings and solution.

Sources of Funding for Research Presented in a Scientific Article or Scientific Article Itself

No funding was received for conducting this study.

Conflict of Interest

The authors have no conflicts of interest to declare

Creative Commons Attribution License 4.0 (Attribution 4.0 International, CC BY 4.0)

This article is published under the terms of the Creative Commons Attribution License 4.0
https://creativecommons.org/licenses/by/4.0/deed.en_US