Consistency-Preserving Deep Reinforcement Learning for Computation Offloading and Resource Allocation in MEC Environment

Huilin Li
The University of Tokyo
Tokyo, Japan
huilinli@hal.ipc.i.u-tokyo.ac.jp

Hiroshi Nakamura
The University of Tokyo
Tokyo, Japan
nakamura@hal.ipc.i.u-tokyo.ac.jp

Hideki Takase
The University of Tokyo
Tokyo, Japan
takasehideki@hal.ipc.i.u-tokyo.ac.jp

***Abstract*—Multi-access edge computing (MEC) has emerged as a promising paradigm for supporting computation-intensive and latency-sensitive services by deploying edge servers closer to user terminal devices (UTDs). However, the limited computation and communication resources in MEC environments make the joint optimization of computation offloading and resource allocation highly challenging. This complexity arises from the strong coupling among optimization variables and the interdependence of users' decisions. To address this, we propose a consistency-preserving deep reinforcement learning (CPDRL) method to reduce the latency and energy consumption of all UTDs, while ensuring the consistency between UTDs' computation and communication resource demands. We first formulate the system as a mixed-integer nonlinear programming (MINLP) problem. Then, we propose a novel optimization algorithm by modifying the network architecture, post-processing the predicted actions, and designing a task-specific reward function and loss formulation. Extensive simulations demonstrate that our approach derives training stability, converges to superior feasible solutions, while achieving high execution efficiency and reliability.**

***Index Terms*—multi-access edge computing, computation offloading, resource allocation, deep reinforcement learning, mixed-integer nonlinear programming.**

I. Introduction

The rapid emergence of novel communication services has imposed tremendous pressure on user terminal devices (UTDs) [1]. Multi-access edge computing (MEC) addresses this challenge by deploying edge servers closer to UTDs, enabling computation offloading that provides additional computing and storage resources while alleviating local burdens [2]. However, since computation and communication resources are limited, it is necessary to jointly optimize computation offloading and resource allocation to achieve system-wide efficiency. This joint optimization is typically formulated as a mixed-integer nonlinear programming (MINLP) problem, which is difficult to solve with classical methods [3], [6].

Deep reinforcement learning (DRL) imposes no strict requirements on the problem form and has emerged as an effective approach for solving such optimization problems [4]. Nevertheless, existing DRL-based solutions often treat computation offloading and resource allocation either separately or only loosely coupled, which frequently results in infeasible decisions. For example, assigning transmission power without actually offloading, or requesting edge computation resources while retaining tasks locally, not only wastes scarce system resources but also contaminates the replay buffer with misleading samples. Such inconsistencies enlarge the exploration space, degrade sample efficiency, and may even cause instability or non-convergence in training. Therefore, preserving consistency among optimization variables during learning is essential to ensure both convergence and deployability of the resulting strategies. Here, consistency-preserving refers to maintaining the alignment between users' offloading decisions and resource allocation, ensuring that only the necessary computation resources are assigned during the optimization process.

To achieve joint optimization of computation offloading and resource allocation while preserving the strong coupling among optimization variables, we propose a consistency-preserving deep reinforcement learning (CPDRL) method. During training, the algorithm enforces consistency between UTDs' computation and communication resource demands, avoiding infeasible variable combinations. This reduces the exploration space, lowers computational complexity, and prevents invalid solutions from contaminating the replay buffer, thereby improving both convergence speed and training stability. The main contributions are summarized as follows:

- We investigate a single-server multi-user scenario in the MEC environment. We formulate the joint optimization of computation offloading and resource allocation as a MINLP problem. The objective is to minimize the total time delay and energy consumption of the UTDs by adjusting variables such as offloading decision, bandwidth, transmission power, and local computing ability, while satisfying the delay requirements of each UTD.
- We propose a CPDRL method. By modifying the network architecture, post-processing the predicted actions, and introducing reward and penalty terms into the reward function and network loss, we enforce the consistency among optimization variables during the training process and improve the convergence.

- We conduct extensive simulations under various parameter settings and compare the proposed method with several classical baselines. The results demonstrate the effectiveness and reliability of the proposed approach.

The remainder of the paper is organized as follows: Section II introduces the related works. The system model and problem formulation are given in Section III. Section IV presents the proposed method. Section V demonstrates the simulation results and discussions. Finally, we conclude this paper in Section VI.

II. Related work

The joint optimization of computation offloading and resource allocation in MEC networks has long been a research hotspot. Due to the heterogeneous data types of optimization variables and the complexity of delay and energy consumption calculations, this problem is typically modeled as a non-convex optimization problem. A common approach to solving it is through a non-convex problem transformation. In [5], the authors used an equivalent transformation to convert the resource allocation problem into a convex form and solved it using the Karush-Kuhn-Tucker (KKT) conditions. However, such transformations require strict structural assumptions and equivalence proofs, limiting their generality and practicality. Moreover, the strong coupling among optimization variables presents another major challenge. In [6], the authors addressed this issue by decomposing the original joint optimization problem into two separate stages and performing iterative updates. While this method handles the coupling effectively, it may increase computing complexity and makes it difficult to guarantee convergence during iteration.

DRL places no strict requirements on the form of the optimization problem and has become a popular approach for solving such problems in recent years. In [7], the authors proposed a DRL approach combined with multiple deep neural networks to obtain the optimal solution in complex action spaces. In [8], the authors further considered the mutual influence among different users and proposed a multi-agent DRL method. However, these works do not address the issue of consistency among optimization variables during the optimization process. Properly handling this consistency is critical for improving algorithm performance, yet current research on this topic remains limited.

III. System model

We consider a single-server multi-user communication scenario, as illustrated in Fig. 1. Within a rectangular area, there is one edge server S and n UTDs. The edge server is located at the center of the area and its coverage includes the entire region. The UTDs are randomly distributed within the rectangular area and are denoted by $N = \{1, 2, 3, \ldots, n\}$.

Each UTD n has a computation task to be executed, represented by a tuple (D_n, C_n, τ), where D_n denotes the data size in bits, C_n denotes the number of CPU cycles required to process one bit of data in cycles/bit, and τ represents the maximum tolerable delay. In this study, we focus on the binary

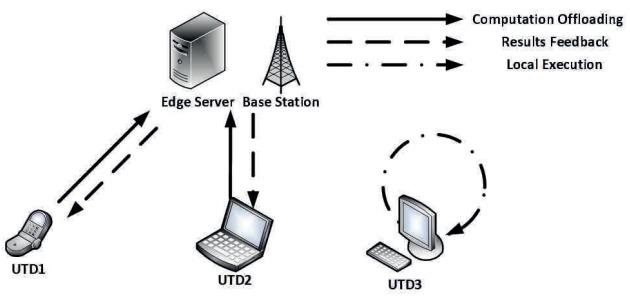

Fig. 1: System model

offloading, so a computation task cannot be partitioned and must be executed either entirely at the edge server or entirely locally. The offloading decision matrix is denoted by $A = \{a_1, a_2, \ldots, a_n, n \in N\}$, where each element a_n takes one of the following two values:

- $a_n = 0$: UTD n executes the task locally.
- $a_n = 1$: UTD n offloads the task to the edge server S.

Depending on the offloading decisions of the UTDs, the computation tasks can be executed in two different modes.

A. Local computation

When a UTD chooses to execute the computation task locally, it needs to allocate computation resources for the task. We quantify the computation resources using CPU frequency and denote the local computation resource allocation matrix by $f = \{f_1, f_2, \ldots, f_n, n \in N\}$, where f_n denotes the CPU frequency allocated by UTD n in Hz. Based on the above definitions, the local computation delay can be expressed as:

$$T_n^l = \frac{D_n C_n}{f_n} \tag{1}$$

The energy consumption of local computation depends on the CPU frequency and the number of CPU cycles required by the task, and is calculated as follows:

$$E_n^l = \kappa D_n C_n f_n^2 \tag{2}$$

where κ denotes the energy consumption coefficient for local computation [9].

B. Edge computation

When a UTD chooses to offload its computation task to the edge server, it must first transmit the required data to the edge server. After executing the task, the edge server returns the result to the UTD. Therefore, edge computation consists of two stages: data transmission and task execution.

1) Data transmission: During communication with the edge server, the UTD must occupy communication resources in the network and configure appropriate transmission parameters. The bandwidth allocation matrix is denoted by $B = \{b_1, b_2, \ldots, b_n, n \in N\}$ and the power allocation matrix is denoted by $P = \{p_1, p_2, \ldots, p_n, n \in N\}$. b_n and p_n

denote the channel bandwidth occupied by UTD n and the transmission power used during communication, respectively. In this study, we assume that orthogonal frequency division multiple access (OFDMA) is used, so inter-user interference is not considered [10]. According to Shannon's theorem, the data transmission rate is calculated as follows:

$$R_n = b_n \log(1 + \frac{p_n g_n}{\sigma^2}) \tag{3}$$

where σ^2 represents the power of Gaussian white noise. g_n denotes the channel gain, which depends on the distance between the UTD n and the edge server:

$$g_n = distance^{\alpha} \tag{4}$$

where α is the path loss factor.

It is worth noting that, the size of result data after task execution is relatively small. Therefore, the downlink data transmission process is not considered in this study. The data transmission delay is given by:

$$T_n^t = \frac{D_n}{R_n} = \frac{D_n}{b_n \log(1 + \frac{p_n g_n}{\sigma^2})} \tag{5}$$

The energy consumption of data transmission process is:

$$E_n^t = p_n T_n^t = \frac{p_n D_n}{b_n \log(1 + \frac{p_n g_n}{\sigma^2})} \tag{6}$$

2) Task execution: After receiving the computation request and the required data from the UTD, the edge server processes the task using its own computation resources. The total computation resources of the edge server are denoted by F. When computation requests are received simultaneously from multiple UTDs, the edge server allocates its resources equally to them. The task execution delay is calculated as follows:

$$T_n^e = \frac{D_n C_n \sum\limits_{n \in N} a_n}{F} \tag{7}$$

The optimization objective of this study is to minimize the overall delay and energy consumption of all UTDs. Therefore, the energy consumption during the task execution phase is not included in the objective function.

C. Problem formulation

In summary, the total delay of a UTD is given by:

$$T_n = (1 - a_n) T_n^l + a_n (T_n^t + T_n^e) \tag{8}$$

The total energy consumption of a UTD is given by:

$$E_n = (1 - a_n) E_n^l + a_n E_n^t \tag{9}$$

The joint optimization problem of computation offloading and resource allocation is formulated as follows:

$$\begin{aligned} objective: \quad & \min_{A,B,f,P} \sum_{n \in N} (\lambda_1 T_n + \lambda_2 E_n) \\ \text{s.t.} \quad & C1: T_n \le \tau, \forall n \in N \\ & C2: \sum_{n \in N} b_n \le B \\ & C3: 0 \le f_n \le f_{max}, \forall n \in N \\ & C4: 0 \le P_n \le P_{max}, \forall n \in N \end{aligned} \tag{10}$$

The optimization objective is to minimize the weighted sum of delay and energy consumption for all UTDs, λ_1 and λ_2 denote the weights of delay and energy consumption respectively and can be adjusted according to the specific requirements of different communication services. The optimization variables include the offloading decision, channel selection, power allocation, and local computation resource allocation of the UTDs. Constraint C1 indicates that the total delay of UTD can not exceed its maximum tolerable delay. Constraint C2 indicates that the bandwidth allocated to all the UTDs can not exceed the total bandwidth in the network. Constraints C3 and C4 indicate that the local computation ability and transmission power of the UTDs can not exceed their capacity.

D. Consistency relationship

The offloading decisions of UTDs directly affect the manner of resource allocation, and a strong consistency relationship exists between the two, as follows:

$$\begin{cases} b_n = 0, p_n = 0, f_n \neq 0 & \text{if } a_n = 0, \\ b_n \neq 0, p_n \neq 0, f_n = 0 & \text{if } a_n = 1. \end{cases} \tag{11}$$

When a UTD offloads its task, it requires no local computation resources but must allocate transmission power, as well as network bandwidth and edge server resources. Ignoring the consistency among optimization variables during the optimization process can significantly degrade the algorithm's ability to explore feasible solutions and may hinder its convergence.

IV. Proposed method

A. Problem analysis

According to the problem formulation in the previous section, the joint optimization problem of computation offloading and resource allocation is a MINLP problem, which is difficult to solve using conventional mathematical methods. Moreover, the interdependence of optimization variables further increases the problem's complexity.

DRL imposes no strict constraints on the form of the optimization problem, making it a suitable tool for addressing the challenges outlined above. Traditional DRL algorithms lack explicit control over their network outputs, which leads to a large number of infeasible solutions. By customizing the action space, adjusting the network architecture, and designing appropriate reward functions and loss formulations, we propose a novel CPDRL method.

B. CPDRL method

1) Markov decision process (MDP) fundamentals: In reinforcement learning, the optimization process is abstracted as a MDP, where each state transition corresponds to an exploration of the feasible solution space [11]. An agent's interaction with the environment is typically represented as a quadruple (s_t, a_t, r_t, s_{t+1}), comprising the current state, action, reward, and next state.

In this study, the state space is composed of the time delay and energy consumption of all UTDs. The action space

includes each UTD's offloading decision, allocated bandwidth, transmission power, and local computing frequency.

The reward is composed of the objective function, constraint conditions, and the consistency relationships among optimization variables. Given the current state and action, the distribution of the future state is independent of past history, forming a MDP.

2) Algorithm overview: Based on the deep deterministic policy gradient (DDPG) algorithm [12], we propose a consistency-preserving deep reinforcement learning algorithm, whose structure is illustrated in Fig. 2. Through actor network output control and the innovative design of reward and loss functions, we ensure the consistency relationship between optimization variables during training.

The overall procedure of CPDRL is presented in Algorithm 1. At each step, the actor network generates a predicted action based on the current state. Here We modify the actor network by employing offloading decisions as masks to enforce consistency among resource allocation variables in the actor network output. Next, noise is added to the predicted action to enhance exploration. The action then interacts with the environment, yielding a reward and leading to a transition to the next state. Each experience is stored in the replay buffer as a tuple (s_t, a_t, r_t, s_{t+1}). The Q value used for updating the networks corresponds to the immediate reward resulting from the chosen action. Consistency penalty terms are incorporated into both the reward and the actor's loss functions, enabling the algorithm to autonomously learn the consistency relationships. In each episode, the best objective function value and the corresponding action are recorded.

The weights of the critic network are updated by minimizing the following loss function:

$$\mathcal{L}_{\text{critic}} = \frac{1}{N}\sum_{i}(y_i - Q(s_i, a_i|\theta^Q))^2 \tag{12}$$

The parameters of the target networks are updated using a soft update mechanism:

$$\theta^{Q'} \leftarrow \ \eta^Q + (1-\eta)\theta^{Q'} \tag{13}$$

$$\theta^{\mu'} \leftarrow \eta\theta^{\mu} + (1-\eta)\theta^{\mu'} \tag{14}$$

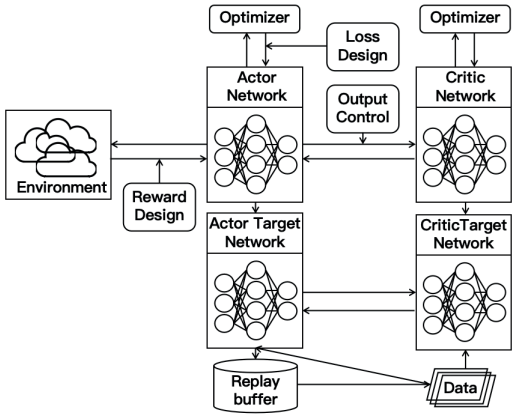

Fig. 2: The structure of CPDRL

Algorithm 1 CPDRL Algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$.
Initialize replay buffer R.
for episode $= 1, M$ **do**
  Initialize a random noise $\mathcal{N}$ for action exploration.
  Receive initial observation state s_1.
  for $step = 1, T$ **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$.
    Perform consistency-preserving adjustment.
    Execute a_t, observe reward r_t and new state s_{t+1}.
    Store tuple (s_t, a_t, r_t, s_{t+1}) in R.
    Sample a random batch of (s_i, a_i, r_i, s_{i+1}) from R.
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$.
    Update critic using eq. (12)
    Update the actor policy using eq. (18)
    Update the target networks using eq. (13) and eq. (14)
  end for
  Record the best objective value and action.
end for

3) Action output control: We modify the actor network architecture, as shown in Fig. 3. The input to the actor network is the state, and the output action is divided into four components: the offloading decision A, channel selection B, local computing frequency f and power allocation P.

In this study, we adopt the tanh function as the activation function in the output layer, resulting in continuous values within the range [-1,1]. Therefore, the offloading decision A is first binarized using the straight-through estimator (STE), as shown below [13]:

$$\begin{cases} \tilde{a_n} = \text{round}(\text{sigmoid}(\beta a_n^{raw})) \\ a_n = \tilde{a_n} + (\text{sigmoid}(\beta a_n^{raw}) - \text{stop\_grad}(\text{sigmoid}(\beta a_n^{raw}))) \end{cases} \tag{15}$$

a_n^{raw} is the raw output of the actor network. β is a sharpness factor to push the sigmoid output closer to 0 or 1. $round()$ performs binary discretization (0 or 1). $stop\_grad()$ prevents gradients from being propagated through its argument. $\tilde{a_n}$ is the discrete action used during the forward pass. a_n is the final differentiable surrogate used during training.

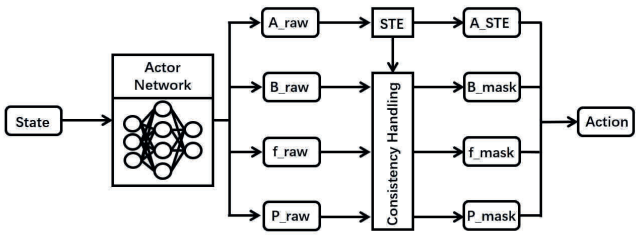

Fig. 3: Actor network design of CPDRL

Next, the binarized offloading decision is used as a mask to element-wise multiply the remaining components of the action:

$$\begin{cases} b_n = \tilde{a_n} \cdot b_n^{raw} \\ f_n = (1 - \tilde{a_n}) \cdot b_n^{raw} \\ p_n = \tilde{a_n} \cdot b_n^{raw} \end{cases} \tag{16}$$

Finally, the four components are concatenated to obtain the final output of the actor network.

4) Reward and actor loss design: We introduce bonus and penalty terms into both the reward function and the actor loss, enabling the algorithm to autonomously learn the consistency relationships and get feasible solutions. The reward function is defined as follows:

$$\begin{aligned} \textit{reward} = &- \sum_{n \in N} T_n - \sum_{n \in N} E_n \\ &- \left(\sum_{n \in N} T_n - \tau \right) - \left(\sum_{n \in N} b_n - B \right) \\ &+ \textit{consistency\_bonus} \end{aligned} \tag{17}$$

The first two terms reflect the objective function, where lower delay and energy consumption yield higher reward. The third and fourth terms correspond to constraint C1 and C2, violation of these constraints results in a reduction in the reward. The fifth term represents the number of UTDs whose strategies satisfy the consistency relationships, serving as a bonus term to guide the algorithm's exploration direction.

The actor loss is modified based on the standard DDPG algorithm by incorporating a penalty term for consistency violations. It is calculated as follows:

$$\mathcal{L}_{\text{actor}} = -\mathbb{E}_{s \sim \mathcal{D}} \left[Q(s, \mu(s)) \right] + \lambda_{\text{c}} \cdot \mathcal{L}_{\text{consistency}} \tag{18}$$

$$\mathcal{L}_{\text{consistency}} = \frac{1}{N} \sum_{n=1}^{N} \left[a_n \cdot (f_n)^2 + (1 - a_n) \cdot \left(b_n^2 + p_n^2 \right) \right] \tag{19}$$

C. Complexity analysis

Let $\mathcal{O}(L_a)$ and $\mathcal{O}(L_c)$ denote the time complexity of a single forward and backward pass of the actor and critic networks, respectively. In each step, the actor computes an action from the current state with a complexity of $\mathcal{O}(L_a)$. The complexity of noise injection and post-processing is $\mathcal{O}(L_a+N)$. The environment interaction based on the selected action incurs a complexity of $\mathcal{O}(N)$. The complexity of sampling from the replay buffer and updating the networks is $\mathcal{O}(L_b(L_a+L_c)+L_p)$, where L_b is the batch size and L_p is the total number of network parameters. Therefore, the overall time complexity of the proposed CPDRL algorithm over M episodes with T steps is: $\mathcal{O}(MT(N + L_b(L_a + L_c)) + L_p)$.

V. Evaluation

A. Simulation settings

The simulation scenario is configured within a 300×300 m rectangular area comprising one centrally located edge server and multiple randomly distributed UTDs. In the proposed CPDRL algorithm, both the actor and critic networks consist of an input layer, two fully connected layers, and an output layer. The detailed parameter settings are shown in Table I.

We compared CPDRL with the following methods in terms of reward, execution time, critic loss, and objective value, where the objective value is defined as the sum of delay and energy consumption of all UTDs:

- The exhaustive method (TEM): An enumeration method that yields a near-optimal solution.
- DDPG algorithm: A classic DRL method.
- STE-DDPG algorithm: DDPG algorithm with STE control in actor network without penalty control.
- ASTE-DDPG algorithm: DDPG algorithm with STE and action control in actor network without penalty control.
- Random offloading method (ROM): The communication and computation resources are randomly allocated.
- Fully offloading method (FOM): All UTDs offload computation tasks with random communication resources.
- Local execution method (LEM): All UTDs execute computation tasks locally with random computation resource.

B. Results and discussion

1) Effectiveness validation: Fig. 4 and Fig. 5 compare the objective values and execution time of CPDRL with the exhaustive method. As the number of UTDs increases, the objective values of both methods rise, with CPDRL closely matching the exhaustive method and achieving near-optimal solutions. Meanwhile, CPDRL's execution time remains almost constant, whereas that of the exhaustive method grows exponentially. Fig. 6 further shows that CPDRL outperforms all classical baselines by achieving the lowest objective value.

2) Convergence and stability validation: Fig. 7 and Fig. 8 demonstrate the effectiveness of CPDRL in improving convergence and training stability. Fig. 7 shows that CPDRL consistently achieves higher rewards than the baselines, while Fig. 8 indicates that its critic loss has the smallest variance, highlighting superior stability and convergence.

TABLE I: Simulation Parameter Settings

Simulation Parameters	Value
The computing ability of UTDs	1-2GHz
The computing ability of edge server	15 GHz
The local data size of UTDs	200-400 KB
The CPU cycles required per data in local training	500-1000 cycles/bit
The channel bandwidth	5MHz
The noise power	10^{-13}
The maximum tolerance time delay	0.5s
The maximum transmission power of UTDs	1W
The path loss exponent	-3
The energy exponent	1×10^{-28}
The learning rate of actor network	1×10^{-4}
The learning rate of critic network	1×10^{-4}
The units number of dense layer in actor network	$(24, 16)$
The units number of dense layer in critic network	$(24, 16)$
The standard deviation of Gaussian noise in CPDRL	1.0
The soft update coefficient in CPDRL	0.005
The discount factor in CPDRL	0.85
The batch size in CPDRL	64
The capacity of replay buffer in CPDRL	100000

3) Cross-Scenario Performance: We evaluate the proposed algorithm under different communication scenarios. As shown in Fig. 9, more UTDs lead to higher total delay and energy consumption. Fig. 10 shows that greater bandwidth improves transmission rates during offloading, thereby reducing delay and energy consumption. Fig. 11 demonstrates that higher edge server capacity further decreases delay and energy consumption. Overall, the proposed algorithm performs well across all scenarios, consistent with theoretical expectations.

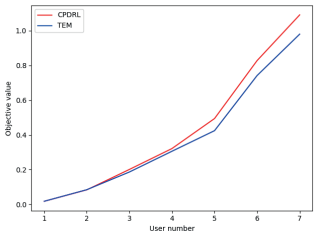

Fig. 4: Objective value comparison of CPDRL and TEM

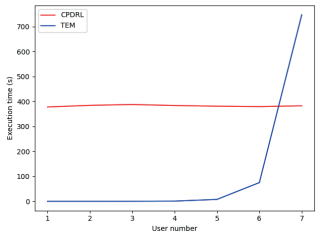

Fig. 5: Execution time comparison of CPDRL and TEM

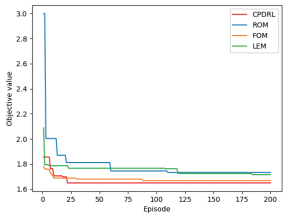

Fig. 6: Objective value comparison of CPDRL and baselines

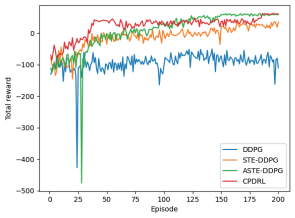

Fig. 7: Reward comparison of CPDRL and baselines

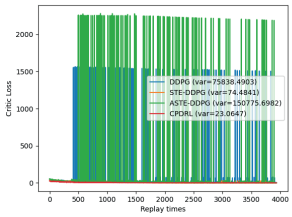

Fig. 8: Convergence comparision of CPDRL and baselines

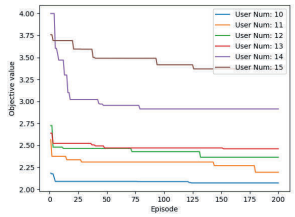

Fig. 9: CPDRL objective value vs. UTD number

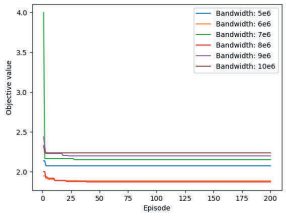

Fig. 10: CPDRL objective value vs. bandwidth

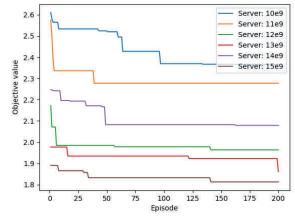

Fig. 11: CPDRL objective value vs. edge server ability

VI. Conclusion

In this study, we investigated the joint computation offloading and resource allocation problem in MEC environment. We proposed a consistency-preserving deep reinforcement learning algorithm that maintains the consistency among optimization variables during the training process. Simulation results demonstrate that the proposed algorithm can more efficiently obtain better feasible solutions while improving convergence performance. In future work, we plan to extend this approach to scenarios involving multiple edge servers.

Acknowledgment

This work was supported by JST, CREST Grant Number JPMJCR21D2 and SPRING Grant Number JPMJSP2108, Japan.

References

[1] K. Li, X. Wang, Q. He, Q. Ni, M. Yang and S. Dustdar, "Computation Offloading for Tasks With Bound Constraints in Multiaccess Edge Computing," in IEEE Internet of Things Journal, vol. 10, no. 17, pp. 15526-15536, 1 Sept.1, 2023.

[2] M. A. Hossain, W. Liu and N. Ansari, "Computation-Efficient Offloading and Power Control for MEC in IoT Networks by Meta-Reinforcement Learning," in IEEE Internet of Things Journal, vol. 11, no. 9, pp. 16722-16730, 1 May1, 2024.

[3] C. Li, Y. Gan, Y. Zhang and Y. Luo, "A Cooperative Computation Offloading Strategy With On-Demand Deployment of Multi-UAVs in UAV-Aided Mobile Edge Computing," in IEEE Transactions on Network and Service Management, vol. 21, no. 2, pp. 2095-2110, April 2024.

[4] H. Ding, Z. Zhao, H. Zhang, W. Liu and D. Yuan, "DRL Based Computation Efficiency Maximization in MEC-Enabled Heterogeneous Networks," in IEEE Transactions on Vehicular Technology, vol. 73, no. 10, pp. 15739-15744, Oct. 2024.

[5] B. Huang, Y. Zhou, X. Zhang, J. Chen and L. Shang, "Computation Offloading and Resource Allocation for Vehicle-Assisted Edge Computing Networks With Joint Access and Backhaul," in IEEE Access, vol. 12, pp. 110248-110259, 2024.

[6] 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.

[7] C. Shang, Y. Sun, H. Luo and M. Guizani, "Computation Offloading and Resource Allocation in NOMA–MEC: A Deep Reinforcement Learning Approach," in IEEE Internet of Things Journal, vol. 10, no. 17, pp. 15464-15476, 1 Sept.1, 2023.

[8] F. Chai et al., "Multi-Agent DDPG Based Resource Allocation in NOMA-Enabled Satellite IoT," in IEEE Transactions on Communications, vol. 72, no. 10, pp. 6287-6300, Oct. 2024.

[9] Z. Yang, M. Chen, W. Saad, C. S. Hong and M. Shikh-Bahaei, "Energy Efficient Federated Learning Over Wireless Communication Networks," in IEEE Transactions on Wireless Communications, vol. 20, no. 3, pp. 1935-1949, March 2021.

[10] L. Tan, Z. Kuang, L. Zhao and A. Liu, "Energy-Efficient Joint Task Offloading and Resource Allocation in OFDMA-Based Collaborative Edge Computing," in IEEE Transactions on Wireless Communications, vol. 21, no. 3, pp. 1960-1972, March 2022.

[11] S. Chouikhi, M. Esseghir and L. Merghem-Boulahia, "Energy-Efficient Computation Offloading Based on Multiagent Deep Reinforcement Learning for Industrial Internet of Things Systems," in IEEE Internet of Things Journal, vol. 11, no. 7, pp. 12228-12239, 1 April1, 2024.

[12] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver and D. Wierstra, "Continuous control with deep reinforcement learning," arXiv, 2019. [Online]. Available: https://arxiv.org/abs/1509.02971

[13] H. Le, R. K. Høier, C. -T. Lin and C. Zach, "AdaSTE: An Adaptive Straight-Through Estimator to Train Binary Neural Networks," 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), New Orleans, LA, USA, 2022.