

Towards a Dynamic Computation Offloading Mechanism with Twin Delayed DDPG in Edge Computing

Aiichiro Oga and Bernady O. Apduhan(✉)

Department of Information Science, Kyushu Sangyo University, Fukuoka, Japan
k20rs023@st.kyusan-u.ac.jp, bob@is.kyusan-u.ac.jp

Abstract. A potential pitfall of edge computing is that processing delays occur when the load is concentrated on a single edge server. To alleviate this issue, computation offloading technology has been proposed to solve this problem. In this research, we aim to realize an efficient computation offloading for edge server load balancing and real-time processing by an offloading mechanism using Twin Delayed DDPG (TD3) algorithm. In the preliminary experiment, the offload destination was determined by considering the location information, load ratio, and CPU utilization of the offload destination edge server, and using DDPG reinforcement learning algorithm. In the main experiments, we used the TD3 reinforcement learning algorithm. The experiments results show that higher episodic reward and learning stability can be achieved by using TD3 with the optimal noise parameters applied compared to DDPG.

Keywords: Computation Offloading · Edge Computing · Twin Delayed DDPG (TD3)

1 Introduction

1.1 Background of Research

In recent years, the Internet of Things (IoT), a technology that connects objects in various industries to the Internet, such as home appliances, industrial robots, and factory line maintenance, in addition to Internet terminals such as PCs and smartphones, is being utilized. Due to this influence, the number of such Internet-connected objects (IoT devices) is expected to further increase in the future. Cloud computing is currently being used as the technology to process the large amount of data generated by IoT devices. Cloud computing is a service that allows users to use software and applications over a network on an on-demand basis. However, as the number of IoT devices being used increases, data transmission capacity grows, and there are problems such as communication delays caused by the squeeze on communication bandwidth and the difficulty in responding to real-time requests due to the distance to the cloud data center.

Edge computing is a technology that has been attracting attention to addressing such issues. Edge computing is a technology in which an edge server is installed near

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2023
O. Gervasi et al. (Eds.): ICCSA 2023 Workshops, LNCS 14107, pp. 133–149, 2023.
https://doi.org/10.1007/978-3-031-37114-1_10

a device, and the edge server processes data and tasks emitted by the device on behalf of the device. Since edge computing allows processing to be performed near the device, it allows for more real-time processing than cloud computing via the Internet and can reduce communication delays between the device and the cloud server. In addition, since processing can be performed solely on the edge server, it is possible to reduce the amount of communication by sending only the processing results to the cloud server without overloading the network. However, the problem of processing delays caused by the concentration of load on a single edge server has been cited. To solve this problem, a technique called computation offloading was considered.

Computation offloading is a technology that reduces the load on the main system by transferring part of the processing to an external system [6–10]. Research is being conducted to solve the edge computing problem by transferring the processing of one edge server to another edge server to reduce the load.

1.2 Research Objective

The objective of this study is to achieve efficient computation offloading, which is a real-time processing of tasks and load balancing of edge servers by an offloading mechanism using deep reinforcement learning to solve the processing delays on edge servers. We will verify whether the offload mechanism using deep reinforcement learning learns optimal offloading decisions and behaves in a way that increases rewards.

The paper is organized as follows. In Sect. 2, we provide a brief review of related technologies. Section 3 cites related research, while Sect. 4 describes the system model and experiment environment. Sections 5 and 6 describe experiments with DDPG and TD3, respectively. Lastly, Sect. 7 discusses our concluding remarks and cites future work.

2 Review of Related Technologies

2.1 Deep Deterministic Policy Gradient(DDPG)

Deep Deterministic Policy Gradient (DDPG) is a deep reinforcement learning algorithm for handling continuous action spaces. DDPG is based on the deterministic policy gradient method because it is a method that applies deep reinforcement learning to the deterministic policy gradient method [1, 2]. While general policy learning methods maximize the expected reward, DDPG is characterized by maximizing the total estimated reward. The learning model of DDPG is shown in Fig. 1. First, the Actor outputs actions from the state, and Critic evaluates Actor's actions with the state and Actor's actions as input and outputs the value of the actions. Then, based on the evaluation from Critic, Actor updates the strategy and selects an action. By repeating these flows, the optimal behavior that can obtain more reward is learned.

In DDPG, the behavioral value function (Q function) is updated using the following target values. This equation represents the sum of the reward at time step t and the discounted future Q value. The future Q value is obtained from the target Critic network using as input the next states_(t + 1) and the action by the target policy in that state.

The DDPG Eq. 1:

$$Q_{target}(t) = r_t + \gamma Q_{target}(s_{t+1}, \pi'(s_{t+1})) \tag{1}$$

where,

- $\mathbf{r}_t$ is the reward at time step t
- γ is the discount rate
- $\mathbf{Q_{target}}$ $\mathbf{(s_{(t+1)},\pi'(s_{(t+1)}))}$ is the **Q** value of the action of the target policy $\boldsymbol{\pi'(s_{(t+1)})}$ in the next state's$_{\mathbf{(t+1)}}$ by the target Critic network.

Critic's loss function $\mathbf{L_{critic}}$ is then defined as the mean squared error between the **Q** value $\mathbf{Q(s_t,a_t)}$ predicted by the network and its target value $\mathbf{(r_t + \gamma Q_{target}(s_{(t+1)},\mu_{target}(s_{(t+1)}))}$.

The DDPG Eq. 2:

$$L_{critic} = \frac{1}{N} \sum (r_t + \gamma Q_{target}(s_{t+1}, \mu_{target}(s_{t+1})) - Q(s_t, a_t))^2 \tag{2}$$

where,

- $\mathbf{r_t}$ is the reward at time step t.
- γ is the discount rate
- $\mathbf{Q_{target}}$ is the target Critic network.
- $\mathbf{s_{(t+1)}}$ is the next state.
- $\boldsymbol{\mu_{target}}$ is the target Actor network (target policy)
- $\mathbf{a_t}$ is the action at time step t
- $\mathbf{Q(s_t,a_t)}$ is the Q value generated by the current Critic network.
- $\boldsymbol{\Sigma}$ represents the sum over all samples in the batch, and **N** is the batch size.

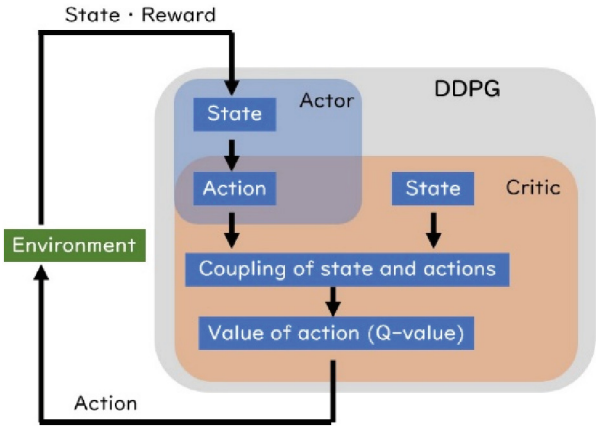


Fig. 1. The DDPG learning model.

2.2 The Twin Delayed Deep Deterministic Policy Gradient(TD3)

Twin Delayed Deep Deterministic Policy Gradient (hereinafter referred to as TD3) is one of the deep reinforcement learning algorithms in continuous action space. TD3 is an extension of Deep Deterministic Policy Gradient (hereinafter referred to as DDPG) and is designed to deal with the problem of overestimation of action value functions with noise [1]. The algorithm estimates the minimum value of the action value function using two independent networks of Critic and suppresses overestimation. In addition, to achieve smooth policy updates in the action space, noise is added to the actions from the target policy network, and the noise is clipped to stabilize the actions.

The learning model of TD3 is shown in Fig. 2. In the learning process of TD3, Actor generates actions from the state, and two Critics evaluate the value of the actions using the state and Actor's actions as inputs. In addition, the minimum value estimates obtained from the two Critic networks are used to update the policy. By delaying the timing of policy update, TD3 ensures that the learning of the value function is stable before updating the policy, thereby improving the learning stability. Through these original innovations, TD3 can achieve higher performance and stability in the continuous action space.

In TD3, the behavioral value function (Q function) is updated using the following target values. This equation represents the sum of the reward at time step t and the discounted future Q-value (the minimum of the outputs of the two target Critic networks). The future Q-value is obtained from each target Critic network using as input the next state and the action taken by the target policy in that state.

The TD3 Eq. 1:

$$Q_{target}(t) = r_t + \gamma min_{i=1,2}(Q_{i,target}(s_{t+1}, \pi'(s_{t+1}))) \tag{3}$$

where,

- $\mathbf{r_t}$ is the reward at time step t
- γ is the discount rate
- $\mathbf{min_{i=1,2}}$ $(\mathbf{Q_{i,\ target}}$ $(\mathbf{s_{t+1}}, \boldsymbol{\pi}'(\mathbf{s_{t+1}})))$ are the **Q** values of the target policy $\boldsymbol{\pi}'$ $(\mathbf{s_{t+1}})$ actions in the next state $\mathbf{s_{t+1}}$ by the target Critic network respectively.

Critic's loss function L_{critic} is then defined as the goal that minimizes the squared error between the expected Q value for the current state and action and the sum of the actual reward and the expected Q value in the next state.

The TD3 Eq. 2:

$$L_{critic} = \sum (r_t + \gamma min_{i=1,2}(Q_{target}(s_{t+1}, \mu_{target}(s_{t+1}) + \mathrm{N}(0, \sigma))) - Q(s_t, a_t))^2 \tag{4}$$

where,

- $\mathbf{r_t}$ is the reward at time step t
- $\boldsymbol{\gamma}$ is the discount rate
- $\mathbf{min_{i=1,2}}$ $(\mathbf{Q_{target}}$ $(\mathbf{s_{t+1}}$,$\boldsymbol{\mu}_{\mathbf{target}}$ $(\mathbf{s_{t+1}})$ + $\mathbf{N(0,\sigma)}))$ is the Q value of two target Critic networks for the action of target policy μ_{target} (s_{t+1}) plus noise N (samples from a

normal distribution with mean 0 and clipping value σ) in the following state. The smaller Q values of the two target Critic networks for the action of the target policy μ_{target}.

- $\mathbf{Q(s_t,a_t)}$ is the Q-value that the current Critic network predicts for state s_t and action a_t.

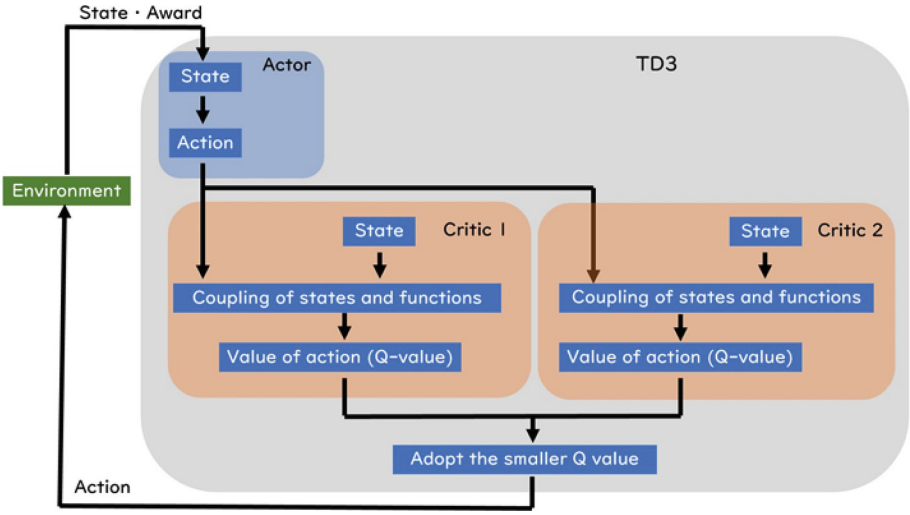


Fig. 2. The TD3 learning model.

3 Related Research

Yunzhao Li, et.al. [3] developed a system using the deep reinforcement learning algorithm DDPG for the problem of computational offloading with unbalanced edge server load, aiming to optimize computational offloading decisions. The system using the DDPG algorithm was compared with the Deep Q Network (DQN) algorithm and the Asynchronous Advantage Actor-Critic (A3C) algorithm to demonstrate the superiority of the system using the DDPG algorithm. This study was able to show the superiority of the system using the DDPG algorithm over the Deep Q Network (DQN) algorithm and the Asynchronous Advantage Actor-Critic (A3C) algorithm on a mobile edge computing environment.

Our research considers data consolidated into tasks from non-mobile devices, such as surveillance cameras and sensors, which can be offloaded to another edge server as the need arises.

Ide, S. [3] conducted research aimed at developing a high-performance edge computing system using the deep reinforcement learning algorithm DDPG to achieve both load balancing and real-time processing in edge computing. In developing the high-performance edge computing system, they investigated the problems that edge computing faces, studied methods to overcome the problems of conventional and related

methods, verified the operation of the proposed method and evaluated its effectiveness. Likewise, they conducted a comprehensive review of both domestic and overseas related research papers. His research assumes that the optimal offload destination from a mobile to each edge server is determined based on the location information of the mobile (e.g., smartphone) and each edge server.

While Tsurumaru, R, et al.[5] share the same main research objectives and used DDPG algorithm with [4], his research attempts to determine the optimal offload destination from one edge server to another based on CPU, memory, and hard disk utilization in addition to location information. Whereas, our research adopted the same experiment environment and conditions as [5] but used TD3 instead of DDPG to evaluate and learn insights on the characteristics of these two deep reinforcements learning algorithms.

4 System Model and Experiment Environment

4.1 System Model

In this study, we assume that tasks possessed by edge server 0 are offloaded to multiple external edge servers, as shown in Fig. 3. The tasks possessed by edge server 0 are data/tasks offloaded from fixed devices such as surveillance cameras and sensors. Edge server 0 and the other edge servers are different edge servers. Edge server 0 only offloads its own tasks to the other edge servers, and no tasks are sent from the other edge servers, except the results of the tasks which were offloaded by server 0 to the concerned servers.

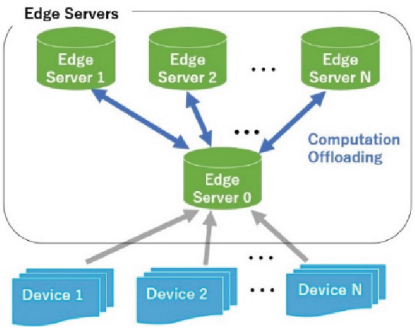


Fig. 3. Data consolidation and tasks offloading between edge servers.

4.2 Proposed Methodology

To perform efficient computation offloading from Edge Server 0, which possesses the task, to multiple external edge servers, we proposed a task offloading mechanism that makes continuous action decisions assuming a single-agent scenario with Broker Server as the agent. The interaction between the Broker Server and the environment is shown in Fig. 4.

To achieve complex decision making of the agent, a deep reinforcement learning algorithm Deep Deterministic Policy Gradient (DDPG) based on the Actor-Critic model is employed. The DDPG algorithm is capable of learning dynamic load-balancing policies because it is characterized by its ability to deal with continuous action spaces.

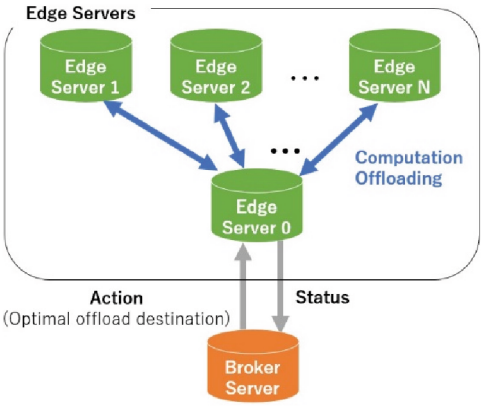


Fig. 4. Interaction between the broker server and concerned edge server.

4.3 Experiment Environment

See (Tables 1, 2 and 3)

Table 1. Machine Specifications

OS	Windows10 Home
CPU	Intel(R) Core(TM) i7-9700K CPU@3.60GHz
GPU	NVIDIA GeForce RTX2070 SUPER
Memory	16.00 GB
Storage	464 GB

5 Experiments with DDPG

5.1 Experiment Objective

In this experiment, we will assume an environment that is close to a real-world situation. That is, the edge server that owns the tasks makes the offloading decisions on which destination servers to offload and confirm whether the offloading mechanism can make the optimal offload decision.

Table 2. Software used in the experiments.

Package	Anaconda
Programming language	Python (ver3.7.1)
Integrated Development Environment (IDE)	PyCharm Community
Library	TensorFlow(ver2.3)

Table 3. Simulation Parameters

Parameter	Value
Learning Steps	3000steps
Data Size	50MB
Migration Bandwidth	1GB/s
No. of Tasks	90
No. of Edge Servers	10
Maximum number of simultaneous edge servers	10

5.2 Experiment Methods

In addition to the task size used in the preliminary experiments and numerical data on the location of each randomly generated edge server, numerical data on the CPU utilization, memory utilization, and hard disk utilization of randomly generated offload destination edge servers were input to the agent. The optimal offload destination is determined based on the utilization rate, memory utilization rate, hard disk utilization rate, load factor, and location information of each edge server. The experiments were conducted on a virtual edge computing environment. Here, the system administrator will set the parameters such as edge servers to be deployed in the environment and task size to be generated in advance (Table 4).

5.3 Results and Discussion

In addition to the location information and load ratio of each edge server, the system was tested to see if it could make optimal offload destination decisions considering the CPU utilization, memory utilization, and hard disk utilization of the offload destination edge servers. Tasks are assumed to be those generated by fixed devices such as surveillance cameras and sensors. We defined the reward value to increase or decrease depending on the number of tasks processed, as well as to provide an additional reward if the offloading decision is made to an edge server with the lowest CPU utilization, memory utilization, and hard disk utilization (less than 50%).

Figure 5 shows the average of the results of 10 experiments for the value of the reward per episode obtained from the training in this experiment. The rewards increased with each successive episode. The learning was terminated when the reward was about

Table 4. DDPG Model Parameters

Parameter	Value
No. of Layers	5
Activation Function	ReLu Function
Actor Learning Rate	1.0×10^{-4}
Critic Learning Rate	2.0×10^{-4}
Discount Rate	0.9
Soft Update Renewal Rate	1.0×10^{-2}
Experience Replay Buffer Size	1.0×10^{4}
Batch Size	32

202,000, because the learning was set to terminate when the maximum reward value was not updated in the last 30 episodes, and the learning was completed (Table 5 and Fig. 6).

Table 5. Summary of results of 10 studies using DDPG

Average number of overall episodes	308.30
Mean value of rewards for the last 30 episodes	199393.80
Median reward of last 30 episodes	199330
Mode of reward of last 30 episodes	198203
Standard deviation of the reward of the last 30 episodes	2534.86
Range of the reward of the last 30 episodes (difference between the maximum and minimum values)	10005
Number of times the mean value of the reward of the last 30 episodes exceeded 198000	7 times

The average amount of reward in the last 30 episodes was 199393.80, with an average number of episodes of 308. Sometimes the study ended with a small number of episodes near 100 and other times with many episodes near 500, and the nature of the graph of averages shows a variation in rewards in the latter part of the study where the population of data is small.

It can be observed that the value of rewards obtained from the environment increases as the episodes progress. This suggests that an appropriate offloading policy is being learned. The increase in the reward value indicates that the system can process many tasks, and that it is successfully learning an appropriate offload policy considering the CPU utilization, memory utilization, and hard disk utilization of the edge server where the offload is to be performed. It is thought that the system can successfully perform

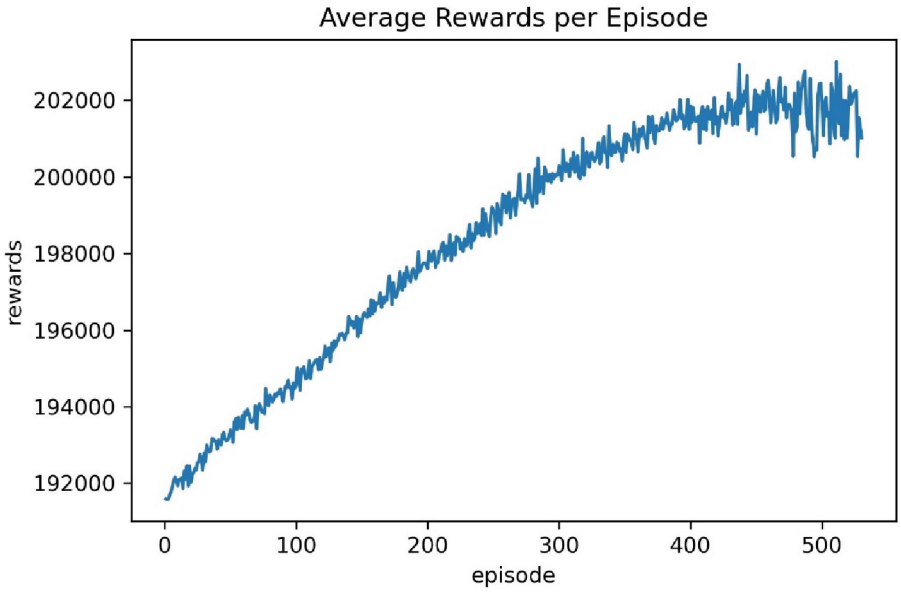


Fig. 5. Average reward of 10 times per episode for the reward of learning results using DDPG (Vertical axis: average amount of reward/horizontal axis: number of episodes)

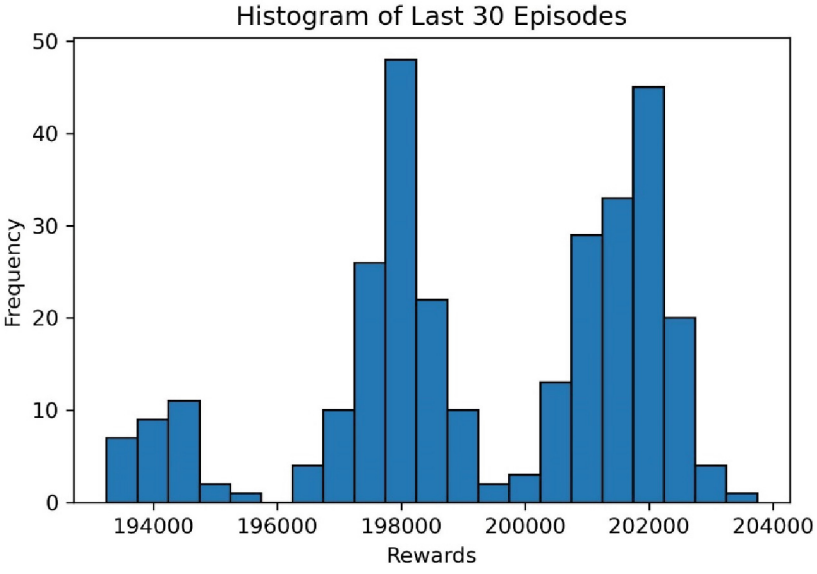


Fig. 6. Histogram summarizing the amount of reward for the last 30 episodes of the 10 learning results using DDPG

learning considering the CPU utilization, memory utilization, and hard disk utilization of the offload destination edge server.

As can be seen from the histogram, the most frequent reward for the last 30 episodes was around 198000, followed by 202000, with a slight outlier appearing around 194000. It is clear that the learning process is relatively stable. Thereafter, we will use 198000, which was the most frequent final reward for DDPG, as a baseline to see if the TD3 reward exceeds this value.

6 Experiments with TD3

We conducted experiments on the same simulation environment using the Twin Delayed DDPG reinforcement learning algorithm.

6.1 The TD3 model parameters

See Table 6.

Table 6. TD3 model parameters

Parameter	Value
No. of layers	5
Activation function	ReLu function
Actor Learning rate	1.0×10^{-4}
Critic Learning rate	2.0×10^{-4}
Discount rate	0.9
Soft Update Renewal rate	1.0×10^{-2}
Experience Replay Buffer size	1.0×10^{4}
Batch size	32
Policy Network Delay Update Interval	2
Clipping value of action noise	0.5–0.6
Standard deviation of action noise	0.2–0.3

6.2 Experiments and Results

To search for the optimal noise parameters, we set the clipping value of the action noise to 0.5 or 0.6 and the standard deviation of the action noise to 0.2 or 0.3. Rewards increase as the episodes progress. The learning was completed when the amount of reward was between 202,000 and approximately 204,000 without updating the previous maximum reward in the last 30 episodes, and when the reward was judged to be stable and high. The following are the learning results for each noise parameter.

6.2.1 TD3 (Noise Clipping Value 0.5, Standard Deviation of Noise 0.2)

See (Table 7, Figs. 7 and 8).

Table 7. Summary of the results of 10 training sessions with TD3 (noise = 0.5/0.2)

Average number of overall episodes	266.50
Mean value of rewards for the last 30 episodes	198383.69
Median reward of last 30 episodes	199278.5
Mode of reward of last 30 episodes	199267
Standard deviation of the reward of the last 30 episodes	3232.94
Range of the reward of the last 30 episodes (difference between the maximum and minimum values)	11131
Number of times the mean value of the reward of the last 30 episodes exceeded 198000	6 times

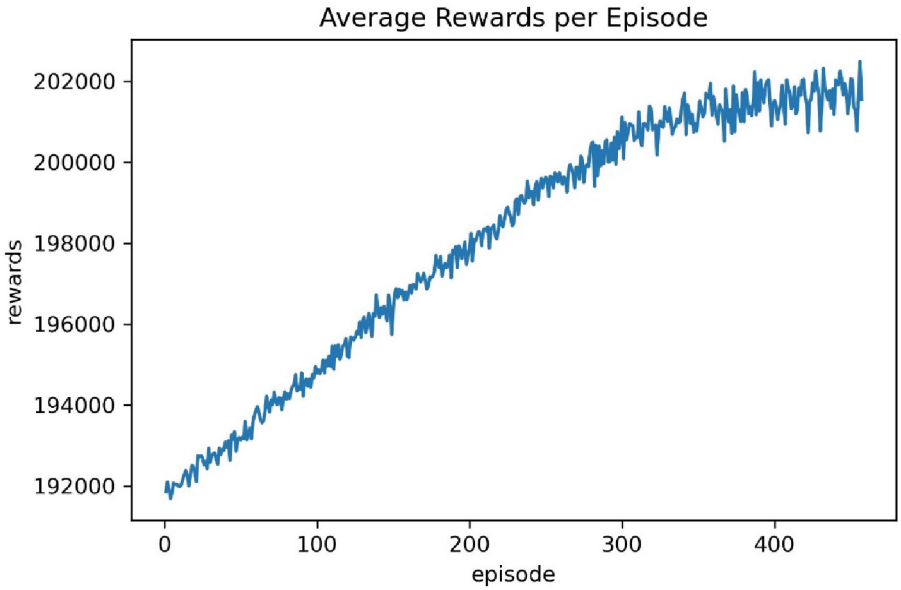


Fig. 7. Reward for learning results using TD3 (noise = 0.5/0.2), Average reward for 10 times per episode (vertical axis: average amount of reward/horizontal axis: number of episodes)

The learning was performed with a clipping value of 0.5 for noise and a standard deviation of 0.2 for noise, with an average reward value of 198383.69 for the last 30 episodes and an average of 266.50 for the number of episodes. The histogram shows a noticeable mountain of learning that was terminated early. It is inferred that the noise was too small, and the search was not sufficient, resulting in a locally optimal solution.

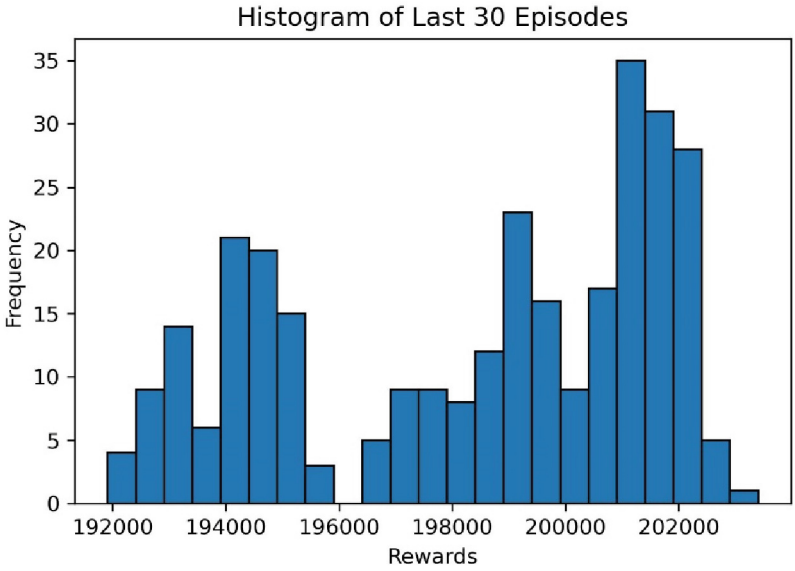


Fig. 8. Histogram summarizing the amount of reward for the last 30 episodes of the 10 learning results using TD3 (noise = 0.5/0.2)

The most frequent reward amount was around 201000, which is an improvement compared to DDPG, but the number of times it was below 198000, the DDPG's most frequent value, increased, raising questions about its stability.

6.2.2 TD3 (Clipping Value of Noise 0.5, Standard Deviation of Noise 0.3)

See Table 8 and Figs. 9 and 10.

Table 8. Summary of 10 training results using TD3 (noise = 0.5/0.3)

Average number of overall episodes	351.00
Mean value of rewards for the last 30 episodes	199606.51
Median reward of last 30 episodes	201058.5
Mode of reward of last 30 episodes	201104
Standard deviation of the reward of the last 30 episodes	3239.91
Range of the reward of the last 30 episodes (difference between the maximum and minimum values)	11842
Number of times the mean value of the reward of the last 30 episodes exceeded 198000	9 times

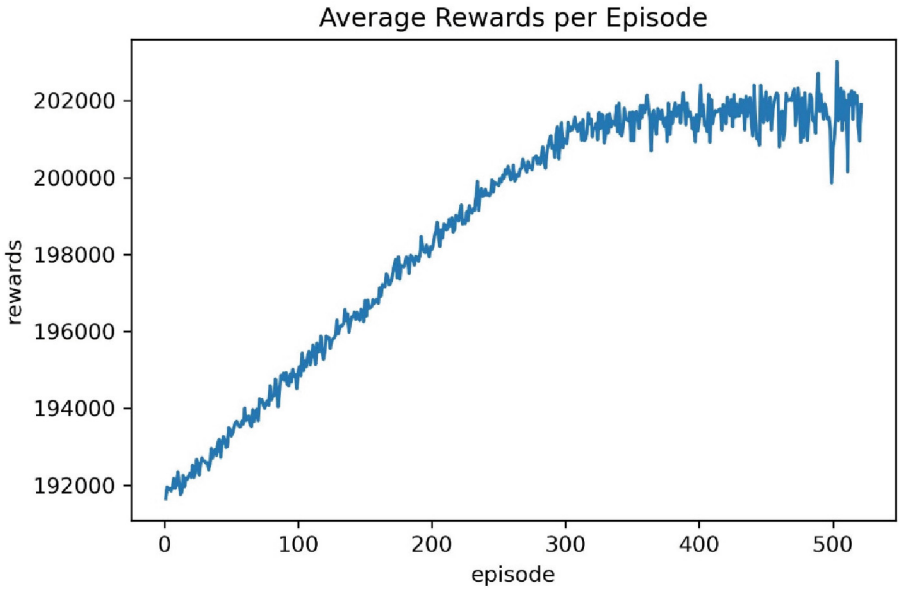


Fig. 9. Reward for learning results using TD3 (noise = 0.5/0.3), Average reward for 10 times per episode (vertical axis: average amount of reward/horizontal axis: number of episodes)

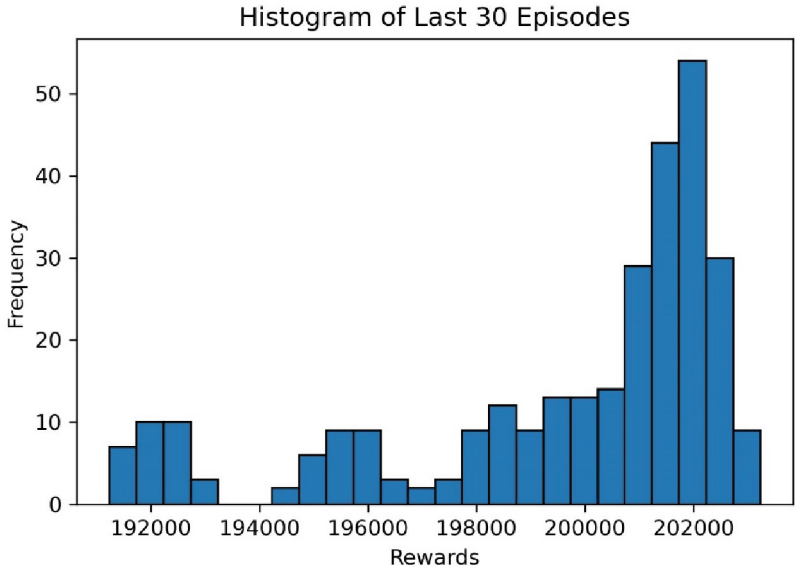


Fig. 10. The rewards of the learning results using TD3 (noise = 0.5/0.3), Histogram summarizing the amount of reward for the last 30 episodes of the 10

(Noise clipping value 0.5, standard deviation of noise 0.3), resulting in an average reward of 199606.51 for the last 30 episodes and 351 for the total number of episodes. As can be seen from the histogram, high episodic rewards were obtained consistently.

The average episode reward exceeded 198,000 9 out of 10 times, making it difficult to find outliers. In addition, the average reward was higher than that of DDPG in all comparisons of the number of episodes (200, 300, 400, and 500 episodes).

6.2.3 TD3 (Clipping Value of Noise 0.6, Standard Deviation of Noise 0.2)

See Table 9 and Figs. 11 and 12.

Table 9. Summary of 10 training results using TD3 (noise = 0.6/0.2)

Average number of overall episodes	314.20
Mean value of rewards for the last 30 episodes	199052.55
Median reward of last 30 episodes	200003
Mode of reward of last 30 episodes	200151
Standard deviation of the reward of the last 30 episodes	3457.62
Range of the reward of the last 30 episodes (difference between the maximum and minimum values)	12907
Number of times the mean value of the reward of the last 30 episodes exceeded 198000	7 times

(Noise clipping value of 0.6, standard deviation of noise of 0.2) was used for training, and the average amount of reward for the last 30 episodes was 19,952, and the overall episode count average was 314.20. As can be seen from the histogram.

The most frequent reward volume is around 203,000. Increasing the noise (clipping value of noise 0.5, standard deviation of noise 0.3) resulted in the mountains shifting to the right and higher rewards compared to (clipping value of noise 0.5, standard deviation of noise 0.3). However, it can also be observed that learning is more often terminated prematurely at 192000–194000. The number of times the average episodic reward exceeded 198000 was slightly less than 7 times. It is considered that when the noise is too large, although the rewards are high because of sufficient exploration, the stability of learning is reduced, and outliers are more likely to be obtained.

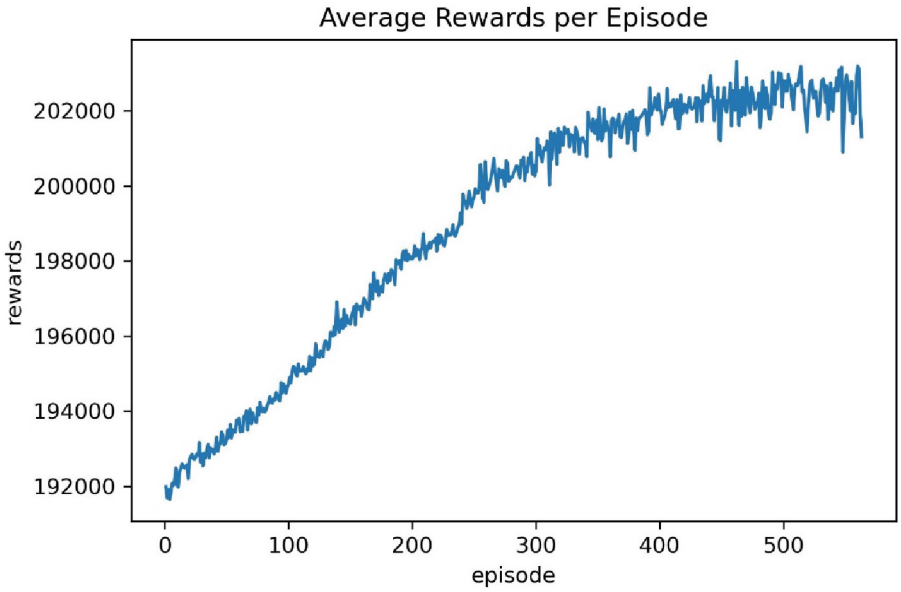


Fig. 11. Reward for learning results using TD3 (noise = 0.6/0.2),Average reward for 10 times per episode (vertical axis: average amount of reward/horizontal axis: number of episodes)

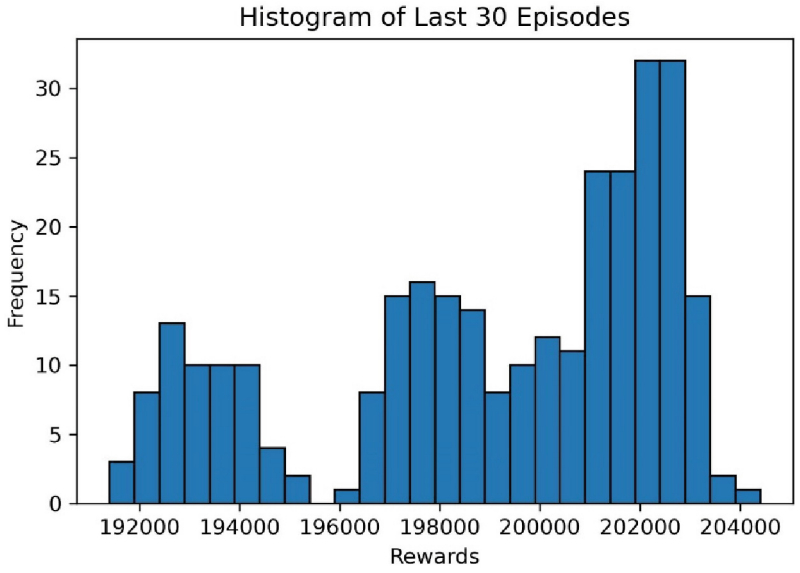


Fig. 12. The rewards of the learning results using TD3 (noise = 0.6/0.2), Histogram summarizing the amount of reward for the last 30 episodes of the 10

7 Concluding Remarks and Future Work

Increasing the parameters of clipping value and standard deviation of the behavioral noise increased the episodic reward but decreased the success rate of learning. From this, we infer that excessive noise impairs learning stability. The noise parameter that minimizes the standard deviation of the episode reward in the final 30 episodes is the combination of a clipping value of 0.5 for action noise and a standard deviation of 0.3 for action noise, which is estimated to be the optimal noise parameter in this experiment. Higher episodic reward and learning stability can be achieved by using TD3 with the optimal noise parameters applied, compared to DDPG.

In the conducted experiments, the task sizes were all set to the same value. However, it is assumed that in a real environment, there will be a variety of tasks and task sizes which differ from each other. To realize a more practical system, we believe it is necessary to conduct experiments with random task sizes. In addition, the experiments were conducted on a virtual edge computing environment, and actual edge servers were not used. Therefore, we consider it necessary to conduct experiments to verify the effectiveness of the offloading mechanism in a real environment.

References

1. Studying Reinforcement Learning from the Basics after All These Years, DDPG/TD3 Edition (Continuous Action Space.) https://qiita.com/pocokhc/items/6746df2eb9e7840e6814 (2022). (in Japanese)
2. Fujimoto, S., Hoof, H., Meger, D.: Addressing function approximation error in actor-critic methods. In: Proceedings of the 35th International Conference on Machine Learning (ICML) (2018). https://arxiv.org/abs/1802.09477
3. Li, Y., Qi, F., Wang, Z., Xiuming, Y., Shao, S.: Distributed edge computing offloading algorithm based on deep reinforcement learning. IEEE Access **8**, 85204–85215 (2020)
4. Ide, S.: Development of a Dynamic Offloading Mechanism for Edge Computing Using Deep Reinforcement Learning: Master Thesis, Graduate School of Information Science, Kyushu Sangyo University (2021). (in Japanese)
5. Tsurumaru, R., Apduhan, B.: Improving a dynamic offloading mechanism in edge computing using deep reinforcement learning. In: Proceedings 2023 Hinokuni Symposium (2023). (in Japanese)
6. Sadiki, A., Bentahar, J., Dssouli, R., En-Nouaary, A.: Deep Reinforcement Learning for the Computation Offloading in MIMO-based Edge Computing: TechRxiv (2021). https://doi.org/10.36227/techrxiv.16869119.v1
7. Li, S., Hu, X., Du, Y.: Deep Reinforcement Learning for Computation Offloading and Resource Allocation in Unmanned-Aerial-Vehicle Assisted Edge Computing. Sensors **21**, 6499 (2021). https://doi.org/10.3390/s21196499
8. J. Li, Z. Chen, X. Liu: Deep reinforcement learning for partial offloading with reliability guarantees. In: Proceedings of the 2021 IEEE ISPA/BDCloud/SocialCom/SustainCom, pp. 1027–1034 (2021)
9. Hazarika, B., Singh, K., Biswas, S., Li, C.-P.: DRL-based resource allocation for computation offloading in IoV networks. IEEE Trans. Indus. Inform. **18**(11), 8027–8038 (2022)
10. Xi, H., Huang, Y.: Deep reinforcement learning based offloading decision algorithm for vehicular edge computing. Peer J. Comput. Sci. **8**, e1126 (2022). https://doi.org/10.7717/peerj-cs.1126