Deep Reinforcement Learning for Resource Scheduling in Edge Devices

Abhishek Kerketta [1], Amogh Dubey [2], Kunal Mahajan [3], Rajarshi Mahapatra [4]
[1,2,4] Department of Electronics and Communication Engineering, IIIT-Naya Raipur, Chhattisgarh, India
[3] Department of Data Science and Artificial Intelligence, IIIT-Naya Raipur, Chhattisgarh, India

***Abstract*—Massive heterogeneous industrial devices communicating with clouds can lead to high latency and demand for high network bandwidth. Edge computing in edge devices can mitigate unacceptable processing delays and alleviate network strain, but limited resources in edge computing can pose a challenge. This paper aims to leverage deep reinforcement learning to develop an effective communication and resource allocation strategy. DRL has been proposed as a solution for resource scheduling on edge devices, where the allocation of computing power and network bandwidth is optimized based on the current state of the environment and the rewards or penalties associated with different actions. DRL uses deep neural networks to represent the agent's decision-making process, allowing for real-time decision-making in resource-constrained environments. This approach has several potential benefits, including reduced latency and energy consumption, improved quality of service for edge devices, and the ability to handle complex and dynamic environments.**

I. Introduction

This paper explores how deep reinforcement learning (DRL) can be used to optimize resource allocation and communication strategies in edge computing environments. We begin with an introduction to the challenges posed by massive heterogeneous industrial devices communicating with clouds, including high latency and demand for high network bandwidth. We then discuss how edge computing in edge devices can mitigate these issues, but limited resources in edge computing can pose a challenge.

Previous research has explored various approaches to resource allocation in edge computing environments. One common approach is to use heuristics or rule-based methods to allocate resources based on predefined policies or thresholds. For example, some studies have proposed using load-balancing algorithms to distribute workloads across multiple edge devices, while others have suggested using dynamic voltage and frequency scaling (DVFS) [1] techniques to adjust the power consumption of edge devices based on workload demands. Several studies have explored the use of DRL for resource allocation in edge computing environments. For example, Li, Y., Mao. proposed [2] [19] a DRL-based approach for task offloading in mobile edge computing systems, demonstrating that their approach outperformed traditional methods such as random selection and greedy algorithms. Similarly, Zhang et al. developed a DRL-based approach for resource allocation in fog computing systems [3], showing that their approach could improve system performance while reducing energy consumption compared to traditional methods. While these approaches can be effective in certain scenarios, they often require significant manual tuning and may not be able to handle complex and dynamic environments. In recent years, machine learning techniques such as deep reinforcement learning (DRL) with continuous action environments have emerged as promising solution for resource allocation in edge computing environments.

To address this challenge, we propose leveraging DDPG [7] as a solution for resource scheduling on edge devices, where the allocation of computing power and network bandwidth is optimized based on the current state of the dynamic environment and the rewards or penalties associated with different actions. DDPG uses actor-critic deep neural networks [8] to represent the agent's decision-making process, allowing for real-time decision-making in resource-constrained dynamic environments. We explain how DDPG uses deep neural networks to represent the agent's decision-making process, allowing for real-time decision-making in resource-constrained dynamic environments [20]. This approach has several potential benefits, including reduced latency and energy consumption, improved quality of service for edge devices, and the ability to handle complex and dynamic environments.

Overall, this paper provides a comprehensive overview of how DDPG can be used to address resource allocation challenges in edge computing dynamic environments, offering insights into both theoretical concepts and practical applications. By leveraging machine learning techniques such as DDPG, we can make real-time decisions that improve system performance and efficiency while reducing energy consumption and improving the quality of service for edge devices. these studies demonstrate the potential of DDPG for optimizing resource allocation in edge-computing dynamic environments.

II. System Model

A. *Overview*

We consider a multi-user heterogeneous network comprising Micro Edge Stations (MES) or Edge Servers, as depicted in Fig. 1. Let $M = [M_0, M_1, ...M_m]$ represent the set of all edge server, where M0 is the first MES. The set of mobile devices is denoted as $U = [U_0, U_1, ...U_n]$. Each MES is equipped with an edge server to provide computational resources in close proximity to the end devices. Each device in the system is responsible for performing a task that is both computationally demanding and time-sensitive. It is imperative that the task is completed within a specific deadline to ensure efficient system performance. The devices will offload their tasks to nearby edge servers for edge computing. In this study, we assume that devices are capable of offloading and migrating their tasks to a nearby MES.

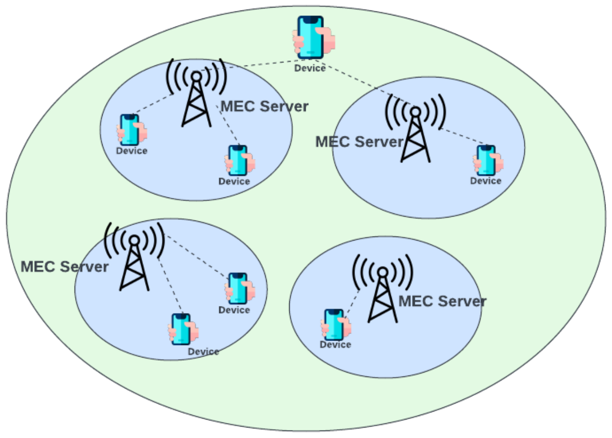


Fig. 1. Network Topology.

B. Mobile User

The Heterogeneous network consists of n mobile devices and supports the offloading and processing of tasks in six stages. Users' requests are transmitted through the network using a transmission model. To generate mobility data, the CRAWDAD [5] data set provides GPS location information for mobile devices.

*1) **Offloading stages**:* In a mobile edge computing system, a task can be offloaded from a mobile user device to an edge server for processing. The offloading process can be divided into several stages. In the first stage, the task is initiated by the mobile user device and sent to the edge server. This marks the beginning of the offloading process. The second stage involves the requested task being on its way to the edge server. Once the requested task reaches the edge server, it enters the third stage, where it is processed. At this stage, the size of the requested task has increased due to additional processing and computation.

After the processing is complete, the requested task is sent back to the mobile user device in the fourth stage. The size of the task at this stage is significantly reduced. It consists only of output results and metadata. The fifth stage marks the end of the offloading process, where the mobile user device disconnects from the edge server. The fifth stage is the default stage of mobile devices.

However, in some cases, the requested task may need to be migrated to another edge server for further processing. This is referred to as the sixth stage of the offloading process.

*2) **Transmission Model**:* In this context, we consider a cellular network consisting of a set U of users and a single-edge server. Users are capable of offloading their computational tasks to the edge server, which processes them. The physical layer transmission follows the classical information theory's Shannon-Hartley theorem [4], which relates the channel capacity, expressed in bits per second, to the bandwidth and signal-to-noise ratio (SNR). The transmission rate for user n can be calculated as:

$$R_i = B_i \log_2 \left(1 + \frac{h_i P_i}{N_0 B_i}\right) \tag{1}$$

Here, B_i represents the bandwidth of user i, P_i is the transmission power of user i, hi is the channel gain between user i and the edge server, and N0 denotes the noise power spectral density.

*3) **Request**:* The proposed methodology for offloading and migrating tasks to an edge server in a distributed edge computing system utilizes a request object. This object represents a user's request to offload a task or migrate an existing task to an edge server. The request is denoted as $R_i = (U_i, M_j,\ \mathrm{s})$, where U_i and M_j correspond to the respective identifiers of the user device and edge server, and s represents the current stage of the user device.

The use of request objects allows for the implementation of a distributed approach to offloading and migrating tasks in a cellular network. By tracking the state of each request, the system can determine when to initiate various stages of the offloading and migration processes.

*4) **Mobility**:* The generation of user mobility data is an essential component in the study of edge computing systems. To accomplish this, the GPS location of mobile devices dataset provided by the CRAWDAD [5] is used. It is assumed that the user's movement follows a linear pattern, where the user moves in a straight line from one location to another. This allows for the prediction of the user's location over time. To make accurate predictions, a time step t is considered between the initial location and the next location.By using this approach, the user's mobility pattern can be simulated, and edge computing systems can be analyzed and optimized accordingly.

C. Edge Server

The mobile edge station (MES) or Edge Server plays a crucial role in data processing and computation tasks at the network's edge. Due to the inherent limitations of resources and bandwidth, the capacity of an Edge Server to handle tasks and connections is finite. In addition to this, Edge Servers also undertake the responsibility of migrating tasks within the network, further emphasizing the importance of their role in the overall network infrastructure.

*1) **Location of MES**:* To compute the appropriate location for each edge server in a distributed edge computing system based on the mean location of user devices connected to each edge server [11], all users are initially divided into M groups, which correspond to the number of mobile edge stations (MESs). Therefore,

$$\text{No. of group's } (G) = \left\lceil \frac{N}{M} \right\rceil$$

Each group is assigned a mobile edge station (MES). Location data is read and collected from each user dataset within a group, and the data are stacked to create a comprehensive location dataset. The mean location of user devices within each group is then used to determine the location of the respective MES. Hence the coordination of Edge server will be

$$x_i = \frac{\sum_{j=iG}^{(i+1)G}\left(\sum_{t=0}^{t} x_{t,j}\right)}{G} \quad y_i = \frac{\sum_{j=iG}\left(\sum_{t=0}^{(i+1)G} y_{t,j}\right)}{G}$$

Where x_i and y_i correspond to the respective location of the edge server, and G represent the number of groups.

*2) **Process Handling**:* In edge computing, the edge server plays a crucial role in handling the tasks offloaded by users. However, the server has limited resources that can only handle a certain number of tasks until the resources are exhausted. The server also has a maximum bandwidth, and each connection between the server and the user takes some transition bandwidth from the total available bandwidth. Therefore, the maximum number of devices connected to the server or the maximum number of processes that the server can handle will depend on the resources and bandwidth available.

As a result, it is important to manage the tasks efficiently and optimize the use of the available resources to ensure that the server can handle as many tasks as possible without being overwhelmed. Proper management of resources and bandwidth can help prevent server crashes and ensure that users receive a seamless and uninterrupted experience when offloading their tasks to the edge server.

*3) **Task Migration**:* The migration process is an essential function in edge computing to handle resource utilization and workload distribution. The migration process occurs when the user device is in the sixth stage, and the task needs to be offloaded to another edge server [17]. When the user sends a request for migration, the edge server checks whether the available bandwidth between the current and the requested edge server is sufficient to handle the user's migration. If the available bandwidth is enough, the migration process takes place in three steps. In the first step, the migration process starts and offloads the user's task to another edge computer. In the second step, the user's task keeps migrating until the migration process is complete. Finally, in the third step, the user is registered to the new edge server, the pre-state is released, and the transmission process continues.

*D. **Problem Formulation***

*1) **State Space**::* At the start of each time slot, the mobile edge station (MES) monitors the system states of heterogeneous wireless networks. The state of the system, denoted by s_t, is a comprehensive object containing all the relevant information about the environment that the agent requires to make an informed decision. The state s is defined as

$$s_t = \{R_{j'} B_j O_j, S_j\} \tag{2}$$

$R_j = [\mathrm{R1}, \mathrm{R2}, \ldots, \mathrm{Rm}]$: Represents the available computing resources of each of the m edge servers.
$B_j = [\mathrm{B1},\ \mathrm{B2}, \ldots, \mathrm{Bm}]$: Represents the available migration bandwidth of each connection between edge servers.
$O_j = [\mathrm{O1}, \mathrm{O2}, \ldots, \mathrm{On}]$: Represents the offloading target of each mobile user.
$S_j = [\mathrm{S1},\ \mathrm{S2}, \ldots, \mathrm{Sn}]$: Represents the location of each mobile user.

*2) **Action Space**:* The action space is the set of all valid actions or choices available to an agent as it interacts with the environment. The action taken by the agent is based on the system's environment and encompasses three main components: offloading decision-making, computation resource allocation, and migration bandwidth allocation. The action a_t is defined as a tuple containing these three components.

$$a_t = \{R_{i'} B_{i'} O_i\} \tag{3}$$

$R_i = [\mathrm{R1}, \mathrm{R2}, \ldots, \mathrm{Rn}]$: Represents the computing resources each mobile user's task needs to use.
$B_i = [\mathrm{B1},\ \mathrm{B2}, \ldots, \mathrm{Bn}]$: Represents the migration bandwidth of each mobile user's task needs to occupy.
$O_i = [\mathrm{O1}, \mathrm{O2}, \ldots, \mathrm{On}]$: Represents the offloading target of each mobile user.

*3) **Reward Function**:* After an action is taken, the system computes immediate reward $R^{imm}\left(s_{t'} a_t\right)$. The objective of the proposed methodology is to manage resource allocation within limited bandwidth [13]. Therefore, the reward will be the total processed tasks in each step. The objective of the DRL model is to maximize the cumulative total reward, given by

$$R = \max E\left[\sum_{t=0}^{T-1} \gamma R^{imm}\left(s_{t'} a_t\right)\right] \tag{4}$$

where $\gamma \in [0,1]$ is the discount factor. If all tasks can be completed within their stringent delay constraints, the agent gets a total reward of R . Otherwise, the agent receives a penalty.

III. Proposed Methodology

*A. **Deep Deterministic Policy Gradient (DDPG)***

Deep Q learning algorithm that learns to make optimal decisions by approximating the action-value function using a deep neural network. By taking into account the state s, the neural network predicts state-action values $\mathrm{Q}\left(s, a_i\right)$, where the number of predicted Q-values is equivalent to the number of actions available in the environment [7]. Consequently, this technique is applicable to discrete environments that can benefit from Deep Q-Learning's capacity to approximate the optimal Q-values for each possible action in a given state. [8]

It is not possible to straightforwardly apply Q-learning to continuous action spaces because in continuous spaces finding the greedy policy requires optimization at every timestep; this optimization is too slow to be practical with large, unconstrained function approximators and nontrivial action spaces.

Instead, here we used an actor-critic approach based on the DPG algorithm

We exploit the advanced DRL algorithm-Deep Deterministic Policy Gradient (DDPG), to solve the resource scheduling and bandwidth allocation problem to maximize the total processed tasks in each step [8]. DDPG consists of three modules: primary network, target network, and replay memory, as shown in Fig. 2 . Primary and secondary networks both consist of actor DNN $\pi\left(s_t \mid \theta_\pi\right)$ and primary critic DNN Q $\left(s_t, a_t \mid \theta_\pi\right)$

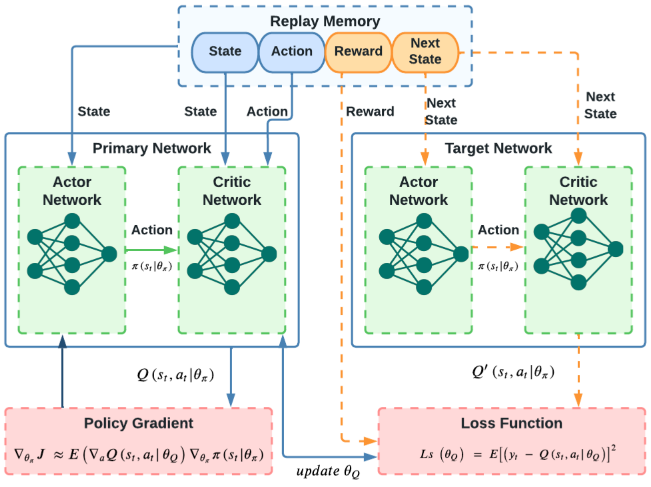


Fig. 2. DDPG Structure

- **Replay Memory**
  It is a component of the Deep Deterministic Policy Gradient (DDPG) algorithm, which is used to solve continuous control problems. It is a buffer that stores past transitions in tuple R_B

$$R_B =< s_t, a_{t'} r_{t+1}, s_{t+1} >$$

  The buffer is then sampled randomly to form mini-batches for training the neural network. By storing and reusing these transitions for exploration and exploitation, the DDPG algorithm can learn from past experience and reduce the correlation between samples, leading to improved stability and convergence of the algorithm.
- **Actor DNN**
  The Actor DNN is responsible for selecting the best action given the current state of the system, Its primary role is to approximate the optimal deterministic policy $\mu(s)$ in a continuous action space [18]. The Actor DNN takes the current state s_t of the system as input and outputs the optimal action a_t to take.
- **Critic DNN**
  The Critic DNN is used to evaluate how good the selected action is. The Critic DNN learns from the replay buffer and updates its parameters to minimize the difference between the predicted Q $\left(s_t, a_t\right)$ and the optimal Q $\left(s_{t'} a_t\right)$ obtained through the Bellman equation.
- **Primary Network**
  The primary network in DDPG refers to the actor-network, which learns the optimal policy for the agent. The primary network is trained using a combination of policy gradient and Q-learning techniques and receives feedback from the critic network to improve its policy.
- **Target Network**
  The target network in DDPG is a copy of the primary network that is used to provide stable targets for the Q-learning update of the primary network. Target networks provide stable targets for the Q-learning update, which leads to more stable training and better convergence of the policy.

B. Algorithm

In this section, we will explore the workflow of our proposed DDPG-based resource scheduling approach, which involves several key steps such as algorithm, training, and testing. We will also discuss the various components of our approach and how they work together to optimize resource allocation in edge computing environments.

Algorithm 1 DDPG

```
Initialize replay buffer $R_B$
Randomly Initialize $Q$ and $\mu$ with weight $\theta_Q$ and $\theta_\mu$
Initialize $Q'$ and $\mu'$ with weight $\theta_{Q'} \leftarrow \theta_Q$ and $\theta_{\mu'} \leftarrow \theta_\mu$
for $iteration = 1, 2, \ldots$ do
    Reset environment to generate state $s$
    for $epoch = 1, 2, \ldots$ do
        Select action $a$ according to state $s$
        Add noise $\eta$ for action exploration
        $s', r \leftarrow stepfunction(a, s)$
        Update critic network
        Update actor-network
        Update target network
        $R_B \leftarrow< s, a, r, s' >$
        $s \rightarrow s'$
    end for
end for
```

*1) **Exploration Noise:*** DDPG noise exploration is a technique used to add stochasticity or randomness to the agent's actions in order to encourage exploration and prevent the agent from getting stuck in a suboptimal policy. In DDPG, Ornstein-Uhlenbeck noise is often added to the selected actions to explore the continuous action space more efficiently [12]. Another approach is to add noise to the policy parameters directly. By gradually decreasing the amount of noise over time, the agent can converge to an optimal policy.

*2) **DDPG step function:*** The action is chosen based on the initial state s, and a certain amount of noise is added to the action to encourage exploration. Once the action is chosen, the DDPG algorithm executes the step function, which comprises several key steps. First, the step function calculates the immediate reward based on the chosen action and the current state of the environment. This reward serves as feedback to the agent and influences its subsequent behaviour. Second, the step function alters the environment based on the chosen action at, generating a new state s.

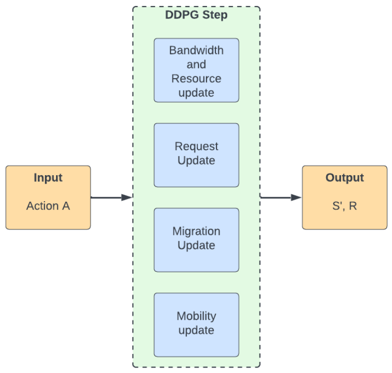


Fig. 3. Step function for Resource and Bandwidth allocation.

- **Resource and Bandwidth update**
  In order to facilitate the efficient allocation of resources in a mobile edge computing system, tables are utilized to store the resource and bandwidth values of each server. These tables allow for easy access to important information regarding the available resources and bandwidth of each server. As mobile users progress through the various stages of the offloading process, they will demand resources from the servers. This demand for resources triggers updates to the corresponding values in the tables. By keeping the tables up-to-date, the system can ensure that resources are allocated efficiently and effectively, leading to optimal system performance.
- **Request update**
  The offloading stage of a user plays a crucial role in the efficiency of edge computing. When a user is initially disconnected, their offloading stage is set to the fifth stage. However, as soon as they connect to the edge server, their stage will be set to zero. The system utilizes the request update process to move the user to the next stage and update the system state accordingly. Once the user reaches stage six, the edge server will perform a migration update to ensure the smooth functioning of the system.
- **Migration Update**
  The migration update is responsible for managing the migration of users from one edge computing resource to another based on factors such as resource availability and user demand. It operates by checking the user's current edge computing resource location and comparing it to the destination location. It maintains the migration process by preparing for migration, starting and continuing migration, and finally registering the user to the target edge. If there is enough capability on the target edge, the function registers the user on the target edge and updates the request ID and state.
- **Mobility Update**
  In a mobile edge computing system, the movement of the mobile user is an important factor that needs to be taken into account. To facilitate this, the time step is increased by one after each episode. Additionally, the new location of the mobile user is calculated after each episode based on their movement pattern. This movement is determined by utilizing data from a CRAWDED dataset. After every iteration, the mobile user reaches the next GPS location in the dataset, which is used to update the user's location within the system.

*3) **Primary Network Training**:*

- **Actor:** The proposed policy can be defined as a mathematical function, with the parameter of the policy denoted by. This function performs a mapping of the present system state to an action or a probability distribution that encompasses all possible actions. In other words, optimal action is taken by taking argmax over the Q-values of all actions.

$$\mu(s) = \operatorname{argmax}_a Q(s,a)$$
$$\mu(s) = \pi\left(s_t \mid \theta_\pi\right)$$

  Here, $\mu(s)$ represents a proto-actor action. Furthermore, $\pi\left(s_t \mid \theta_\pi\right)$ denotes the deterministic policy that is produced by a Deep Neural Network (DNN) model.
  The primary actor network updates the network parameter θ_π using the sampled policy gradient of the loss function J_μ, that is

$$J_\mu = E\left[Q\left(s_{t'}\mu\left(s_i\right)\right)\right]$$
$$\nabla_{\theta^H J_\mu} = E\left[\nabla_\mu Q\left(s_{t'}\mu\left(s_i\right)\right)\nabla_{\theta^\mu}\mu\left(s_i\right)\right]$$

- **Critic:** The critic network in the primary network evaluates the chosen action's performance based on the action-value function $Q(s,a)$. The calculation of the action-value function is determined by the Bellman optimality equation and can be represented as

$$Q\left(s_{t'}\mu\left(s_t\right)\right) = E\left[R^{imm}\left(s_{t'}\mu\left(s_t\right)\right) + \gamma\max_a Q\left(s_{t'}\mu\left(s_t\right)\right)\right]$$

  To calculate the optimal action-value pair, the primary critic deep neural network (DNN) takes both the current state (s_t) and the next state (s_t') as inputs.
  The primary critic DNN updates the network parameter by minimizing the loss function

$$J_Q = E\left[\left(y_t - Q\left(s_{t'}\mu\left(s_t\right)\right)\right)^2\right]$$

  Where y_t is the target value and can be by

$$y_t = R^{imm}\left(s_{t'}\mu\left(s_t\right)\right) + \gamma Q_{trag}\left(s'_{t'}\mu_{trag}\left(s_t\right)\right)$$

  Where $Q_{\text{trag}}\left(s'_{t'}\mu_{\text{trag}}\left(s_t\right)\right)$ is the action value pair value form target network policy. The primary critic network updates network parameters using the sampled policy gradient of the loss function J_Q, that is

$$\nabla\theta_Q L_s = E\left[2\left(y_t - Q\left(s_{t'}\mu\left(s_t\right)\right)_{\nabla\theta_Q} Q\left(s_{t'}a_t\right)\right]\quad (5)$$

*4) **Twin Q Network**:* Twin Q networks modify the traditional Q-networks used in reinforcement learning algorithms, such as the Deep Deterministic Policy Gradient (DDPG) algorithm. In traditional Q-networks, a single neural network approximates the action-value function, which estimates the expected cumulative reward for each action in a given state. However, in Twin Q networks, two separate Q-networks are used to estimate the action-value function independently.

The twin Q-network architecture is used in the DDPG algorithm to address issues with overestimating Q-values in the traditional Q-network approach. Using two separate Q-networks, DDPG can mitigate overestimation by taking the minimum value between the two Q-values estimated by each network.

$$min(Q_{\theta 1}, Q_{\theta 2})$$

where, $Q_{/theta1} and Q_{/theta1}$ represent Q value form two seprate Q networks.Overall, the twin Q-network approach in DDPG is an effective way to improve the performance and stability of reinforcement learning algorithms.

*5) **Target Network Training**:* The target network can be considered as a previous version of the primary network, featuring distinct parameters (θ_π^T and θ_Q^T). During each iteration, the actor parameters (θ_π^T) and critic parameters (θ_Q^T) are updated using the following equation:

$$\theta_\pi^T = \omega\theta_\pi + (1-\omega)\theta_\pi^T \tag{6}$$

$$\theta_Q^T = \omega\theta_Q + (1-\omega)\theta_Q^T \tag{7}$$

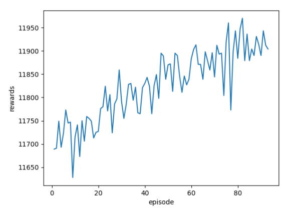
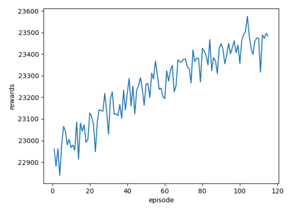
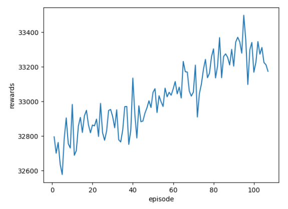
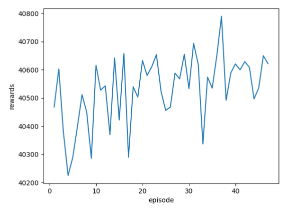
Where, $\omega \in [0, 1]$ The soft update rate is denoted by 'ω'. The target network is updated by gradually blending the weights of the main network with the weights of the target network, using the soft update rate ω'.

The parameter θ_π^T and θ_Q^T is used to calculate $Q_{\text{trag}}\ (s_{t'}\mu_{\text{trag}}\ (s_t))$. By using the target network to estimate the target Q-value, the DDPG algorithm is able to reduce the variance in the updates of the critic network and ensure more stable learning.

IV. Results

In this study, we examine a network topology that consists of M edge servers and N mobile users. The location of the edge servers is generated and mobile users are distributed according to the CRAWDAD dataset, which provides real-world traces of mobile users' movements. The maximum transmission power of mobile users, Pi, is set to $250\ \mathrm{mW}$. To model the wireless channel between mobile users and edge servers, we adopt the channel gain models presented in the 3GPP standardization [14]. The maximum bandwidth available at each edge server is set to $1000\mathrm{MBps}$, and the maximum computational resource available at each edge server is set to 63MBps. These parameters are chosen to reflect the realistic conditions of modern mobile networks and edge computing systems. The offloading task involves the VOC SSD300 object detection model [15], with a 300-dimensional input image transmitted as $300 \times 300 \times 3$ bytes. The computational complexity of the CNN model is 4 bytes per second per image, resulting in a processing requirement of $300 \times 300 \times 3 \times 4$ bytes $/\mathrm{sec}$. The output includes both output and metadata, totalling 4 and 20 bytes per image, respectively.

TABLE I
SIMULATION RESULTS

Number of Clients	**Average Total processed task**	**Training History**
10	11910	
20	23449	
30	33257	
40	40584	

V. Conclusion

This paper has demonstrated the potential of deep reinforcement learning (DRL) for optimizing resource allocation and communication strategies in edge computing environments. By leveraging DRL, we can make real-time decisions in resource-constrained environments, leading to reduced latency and improved efficiency. Our proposed approach has several advantages over traditional resource allocation strategies, including the ability to handle complex and dynamic environments.

Through the experimental results, we have shown that DRL can be an effective solution for resource scheduling on edge devices. Our approach offers several potential benefits, including reduced energy consumption, improved quality of service for edge devices, and the ability to handle diverse workloads.

Overall, this paper contributes to the growing body of research on using machine learning techniques for optimizing resource allocation in edge computing environments. We hope that our work will inspire further research in this area and lead to practical applications that improve the performance and efficiency of industrial systems.

References

[1] Youssef Ait El Mahjoub, Jean-Michel Fourneau, Hind Castel-Taleb, ”Performance Evaluation and Energy Consumption for DVFS Processor”, Performance Engineering and Stochastic Modeling, vol.13104, pp.165, 2021.

[2] Li, Y., Mao, Y., Zhang, J. Leng, S. (2018). Deep reinforcement learning for mobile edge computing offloading: A survey. IEEE Access, 6, 69011-69022.

[3] Liu, X., Zhang, Y., Chen, J., Liang, X. (2019). A deep reinforcement learning-based resource allocation approach for fog computing systems. IEEE Transactions on Industrial Informatics, 15(3), 1745-1754.

[4] Price, E., Woodruff, D.P. (2012). Applications of the Shannon-Hartley theorem to data streams and sparse recovery. In: ISIT, pp. 2446–2450 (2012)

[5] Tristan Henderson, David Kotz (2018). Data citation practices in the CRAWDAD wireless network data archive vol.13, pp.165.

[6] Mao, Y., You, C., Zhang, J., Huang, K., Letaief, K. B. (2017). A survey on mobile edge computing: The communication perspective. IEEE Communications Surveys and Tutorials, 19(4), 2322-2358.

[7] Sutton R.S and Barto A.G.(2018). Reinforcement Learning: An Introduction (2nd ed.). MIT Press.

[8] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, Daan Wierstra Continuous control with deep reinforcement learning 21(4), 3425-3451, 2019.

[9] Liu, X., Zhang, Y., Chen J. (2020). A deep reinforcement learning-based task offloading approach for mobile edge computing systems with energy harvesting. IEEE Transactions on Mobile Computing, 19(6), 1477-1489.

[10] Zhang, Y., Mao, Y., Letaief, K. B., Chen, J. (2019). Deep reinforcement learning for mobile edge computing: A survey. IEEE Communications Surveys and Tutorials, 21(3), 2224-2287.

[11] Satyanarayanan, M., Bahl, P., Caceres, R., Davies, N. (2009). The case for VM-based cloudlets in mobile computing. IEEE Pervasive Computing, 8(4), 14-23.

[12] Plappert, M., Houthooft, R., Dhariwal, P., Sidor, S., Chen, R. Y., Chen, X., Schulman, J. (2018). Parameter space noise for exploration. arXiv preprint arXiv:1706.01905

[13] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... and Petersen, S. (2015). Human-level control through deep reinforcement learning. Nature, 518(7540), 529-533.

[14] Dahlman, E., Parkvall, S., Skold, J. (2018). 5G NR: The Next Generation Wireless Access Technology. Academic Press.

[15] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C. Y., Berg, A. C. (2016). SSD: Single shot multibox detector. In European conference on computer vision (pp. 21-37). Springer.

[16] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D. (2016). Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971.

[17] Wang, X., Wen, Y., Chen, Z., Li, H. (2019). Task migration in edge computing: state of the art and research challenges. IEEE Internet of Things Journal, 7(7), 5440-5455.

[18] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A. (2015). Human-level control through deep reinforcement learning. Nature, 518(7540), 529-533.

[19] Ahmadvand, H., Foroutan, F., and Fathy, M. (2021). DV-DVFS: Merging Data Variety and DVFS Technique to Manage the Energy Consumption of Big Data Processing. ArXiv. /abs/2102.03751

[20] Zhao, J., and Yang, C. (2021). Deep Reinforcement Learning with Symmetric Prior for Predictive Power Allocation to Mobile Users. ArXiv. /abs/2103.13298