Deep Reinforcement Learning-based Task Offloading and Resource Allocation in MEC-enabled Wireless Networks

Seble Birhanu Engidayehu, Tahira Mahboob, and Min Young Chung, *Member, IEEE*
Department of Electrical and Computer Engineering,
Sungkyunkwan University,
2066 Seobu–ro, Jangan–gu, Suwon–si,
Gyeonggi–do, 16419, Republic of Korea,
eshet@g.skku.edu, {tahira,mychung}@skku.edu

***Abstract*—Mobile edge computing (MEC) has recently become an enabling technology for mobile operators that are offering a diverse set of services. These services require extensive storage, energy, and computation resources. However, user devices (UDs) have resource constraint to meet the requirements of such services. To tackle the contradiction between resource-constrained UDs and computationally intensive services, MEC have been proposed. MEC servers provide task execution services for UDs. On receiving a service request from the UDs, a MEC server within the network may dynamically allocate computation and memory resources for the task execution. As the MEC servers have limited capacity, efficient utilization of MEC resources is necessitated. Also, it is challenging to find an optimal solution for efficient resource allocations due to different task requirements for a diverse set of services offered to users and dynamicity in wireless networks. To address these problems, we propose a partial task offloading and resource allocation scheme to maximize user task completion within a tolerable time period while minimizing energy consumption. In this paper, we convert the formulated optimization problem to a markov decision process (MDP) and then propose a solution based on the deep deterministic policy gradient (DDPG) algorithm. The performance results show that the proposed method completes a greater number of tasks within a tolerable delay and reduces the energy consumption in the network, compared to those of other conventional schemes.**

***Index Terms*—Deep deterministic policy gradient (DDPG), mobile edge computing (MEC), partial offloading, resource allocation.**

I. Introduction

With a rapid increase in usage of Internet–of–things (IoT) devices, the number and types of resource intensive and delay–sensitive mobile applications have grown exponentially. Applications, such as, augmented reality (AR) and virtual reality (VR), have high demands for memory, compute power, and energy for successful execution. To alleviate the resource constraints faced by user devices (UDs), mobile edge computing (MEC)–based task offloading strategies have been proposed. The MEC servers are deployed close to user point–of–attachment, so that UDs can migrate their task to them for execution [1].

In recent years, MEC resource management has been widely studied topics [2]. As MEC servers have limited capacity, it is important to optimize task offloading and resource management. In [3], the MEC server processes the data related to human emotion that is received from the sensors installed on the IoT devices. To minimize the energy and time consumption of the MEC server and the sensors, a computation resource allocation algorithm has been introduced. The existing researches are based on classical optimization methods to find optimal offloading decisions and resource allocations in single and multiple user systems [4]. Conventional optimization techniques may suffer from instability and sub–optimal solutions due to the dynamic wireless channel, as well as the heterogeneity of tasks generated by UDs. Therefore, it is challenging to meet the resource requirements of different types of tasks.

However, learning based solutions can adopt this variation. In [7], Ren et al. used deep reinforcement learning (DRL) for dynamic power allocation. However, the delay requirement for UDs was considered fixed. According to most studies, the user device's task-related information such as tolerable delay, data size, and computation resources (CPU cycles) was considered fixed [7]. To overcome problems in the previous works, we propose a partial task offloading and resource allocation technique that enables multiple user devices (UDs) with varying data size, computation resources (CPU cycles), and tolerable delay requirements to offload part of the task to the MEC server and then compute the remaining portion locally. To meet the requirements of wireless devices and the dynamic nature of wireless channels, we convert the formulated problem using the markov decision process (MDP), and propose a deep deterministic policy gradient (DDPG)–based algorithm to maximize the number of tasks completed within a tolerable time period and minimize the total energy consumption.

The rest of this paper is organized as follows. In Section II, a system model is presented. Section III provides a detailed explanation of the DDPG-based task offloading and resource

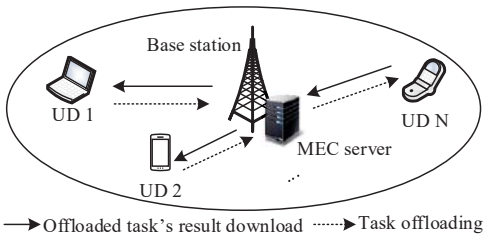

Fig. 1. Architecture of the MEC Network

allocation scheme. The Simulation results are presented in Section IV followed by conclusions in Section V.

II. System Model

As shown in Fig.1 we consider a network consisting one base station embedded with a MEC server and n UDs. There are T time slots, each of σ duration. The UDs generate service requests, referred to as tasks, at each time slot t, where $t \in \mathcal{T} = \{1, 2, \ldots, T\}$. The task information can be represented by $\Omega_{n,t} = \{B_{n,t}, C_{n,t}, D_{n,t}\}$, where $B_{n,t}$, $C_{n,t}$, $D_{n,t}$ are the data size in bits, the required CPU cycle per bit, and the tolerable delay in ms. We assume that the generated task can be partitioned in accordance with the offloading ratio $\alpha_{n,t} \in [0, 1]$. We employ a partial offloading scheme, in which the MEC server, after receiving a task execution request from UDs, determines the optimal portion of task $\alpha_{n,t}$ to be computed at both the UD n and the MEC server. Based on the offloading decision, the UD migrate $\alpha_{n,t}B_{n,t}$ portion of their tasks to the MEC server, while $(1 - \alpha_{n,t})B_{n,t}$ portion is computed at the UD.

Furthermore, we assume that the system uses orthogonal frequency division multiple access (OFDMA), which eliminates interference between UDs by allocating separate subchannels. The UDs can transmit tasks to the base station (embedded with a MEC server) using

$$r_{n,t} = \omega_{n,t} W \log_2\left(1 + \frac{p_n \zeta_{n,t}}{\delta^2}\right), \tag{1}$$

where W is the total bandwidth between MEC server and all UDs, $\omega_{n,t} \in [0, 1]$ is the fractional bandwidth allocated to the UD n, $\zeta_{n,t}$ is the channel gain of UD n, δ is noise power and p_n is the transmit power of UD n.

A. Task execution model

At each time slot t, UDs n generate resource-intensive tasks represented by $\Omega_{n,t}$. In the partial offloading approach, the MEC server is capable of computing a portion of the task and the UDs execute the subsequent portion of the task. In the following sections, we outline MEC execution and UD local execution.

1) MEC Execution: If the MEC server decides to execute $\alpha_{n,t}$ part of UD's task, UD n offloads that portion via the allocated bandwidth. The transmission delay and energy consumption can be calculated by

$$T_{n,t}^{off} = \frac{\alpha_{n,t} B_{n,t}}{r_{n,t}}, \tag{2}$$

$$E_{n,t}^{off} = p_n T_{n,t}^{off}. \tag{3}$$

The MEC server executes the received task bits using its computational resources. The MEC server's task processing delay can be determined by using

$$T_{n,t}^{MEC} = \frac{\alpha_{n,t} B_{n,t} C_{n,t}}{\left(\frac{c_m}{N}\right)}, \tag{4}$$

where c_m is the computational resource of the MEC server. The computational resource is equally shared among all UDs. The total time consumption of the offloaded task can be calculated by combining (2) and (4). Thus, we have

$$T_{n,m,t} = T_{n,t}^{MEC} + T_{n,t}^{off}. \tag{5}$$

2) Local Computing: The UD executes $(1 - \alpha_{n,t})B_{n,t}$ portion of the task locally. Consequently, the time and energy consumption related to local execution can be calculated as

$$T_{n,t}^{loc} = \frac{(1 - \alpha_{n,t}) B_{n,t} C_{n,t}}{c_n}, \tag{6}$$

$$E_{n,t}^{loc} = (1 - \alpha_{n,t}) B_{n,t} C_{n,t} \varrho c_n^2, \tag{7}$$

where ϱ and c_n are the energy consumption coefficient and computational capacities of each UD, respectively. In partial offloading, UDs and MEC server execute the task simultaneously. Accordingly, the total delay can be calculated as follows

$$T_{n,t}^{tot} = \max\left\{T_{n,m,t}, T_{n,t}^{loc}\right\}. \tag{8}$$

The total energy consumption can be computed using (3) and (7).

$$E_{n,t}^{tot} = E_{n,t}^{off} + E_{n,t}^{loc}. \tag{9}$$

We assume that the data downloaded from the MEC server onto the UD is small in size and is downloaded quickly. Thus, the energy and time required to download are therefore not taken into account [8].

B. Problem Formulation

We propose an efficient task offloading and resource allocation scheme. We aim to reduce the total energy consumption while maximizing the number of tasks completed within a tolerable time period. Thus, the formulated optimization problem is derived as

$$\max_{\omega,\alpha} \sum_{t=1}^{T}\sum_{n=1}^{N} \lambda_1 H_{n,t} - \lambda_2 E_{n,t}^{tot} \tag{10}$$

$$\text{s.t.} \quad \alpha_{n,t} \in [0,1], \forall\ n \in \mathcal{N}, t \in \mathcal{T}, \tag{11}$$

$$\omega_{n,t} \in [0,1], \forall n \in \mathcal{N}, t \in \mathcal{T}, \tag{12}$$

$$T_{n,t}^{tot} \leq D_{n,t}, \forall n \in \mathcal{N}, t \in \mathcal{T}, \tag{13}$$

$$\sum_{n\in\mathcal{N}} \omega_{n,t} \leq 1, \forall t \in \mathcal{T}, \tag{14}$$

where constraint (11) represents the offloading ratio of each UD at time slot t. Constraint (12) specifies the amount of bandwidth allocated to each device. As indicated by constraint (14), the sum of the fraction of bandwidth resources allocated for each user device must be less than the available link bandwidth of the MEC server. In general, the total delay to complete each task should not exceed each device's maximum delay tolerance. $H_{n,t}$ represents a flag, which indicates whether the constraint (13) is satisfied or not .

III. Problem solution

In this section, we present the formulated problem as a markov decision process (MDP) and propose a DDPG-based solution to allocate bandwidth and offload tasks at the MEC server efficiently.

A. Markov Decision Process definition

MDP consists of the following key elements:

1) State: This includes the channel gain $\phi_{n,t}$ and the task information of UD n at time slot t. Therefore, the state space can be described as: $z_t \in \mathbf{Z} = \{\phi_{n,t}, \ldots, \phi_{N,t}, \Omega_{n,t}, \ldots, \Omega_{N,t}\}$.

2) Action: The state changes depending on which actions are taken. In this case, the offloading decision is $\alpha_t = \{\alpha_{1,t}, \ldots, \alpha_{N,t}\}$ and resource allocation is $\omega_{n,t} = \{\omega_{1,t}, \ldots, \omega_{N,t}\}$. Thus, the action space is $a_t = \{\alpha_{1,t}, \ldots, \alpha_{N,t}, \omega_{1,t}, \ldots, \omega_{N,t}\}$.

3) Reward: In the state Z_t, the agent executes each possible action a_t and is rewarded with $R_t(Z_t, \alpha_t)$. To maximize the reward, the agent executes as many tasks as possible within tolerable delay, while minimizing the energy consumption required to complete each task. As a consequence, the reward is positively correlated with the number of tasks completed, but negatively correlated with the amount of energy consumed. The immediate reward, $R_t(Z_t, a_t)$, can be calculated using

$$R_t(Z_t, a_t) = \sum_{n\in\mathcal{N}} \lambda_1 H_{n,t} - \lambda_2 E_{n,t}^{tot}, \tag{15}$$

where $H_{n,t}$ indicates whether constraint (13) is satisfied or not. Thus, if constraint (13) is satisfied, $H_{n,t} = 1$, otherwise $H_{n,t} = 0$. Consequently, the expected return can be defined as

$$R_e = \max_{a_t} \mathbb{E}\Big[\sum_{t=1}^{T} \gamma^{t-1} R_t\Big], \tag{16}$$

where $0 < \gamma \leq 1$ is a discount factor on reward R_t.

Algorithm 1 DDPG method

```
Random initialization of the actor-critic weight paramters for the evaluation and the target network: φ, θ, φ′, and θ′
Initialize replay buffer M
for episode = 1,Ep do
    Receive initial observation Z_t,
    for time = 1,T do
        Select action a_t by evaluation network, i.e., a_t = μ(Z_t|w) + n_o
        Make an action a_t, obtain reward R_t and next observation state Z_{t+1}
        Save transition (Z_t, a_t, R_t, Z_{t+1}) in M
        Sample a random mini-batch transitions B (Z_i, a_i, R_i, Z_{i+1}) from R
        Update evaluation critic and actor parameters based on loss function and policy gradient.
        Update the target actor-critic network parameters using (22) and (21), respectively.
    end for
end for
```

B. DDPG method

Due to the non-convexity of problems such as ours, using reinforcement learning (RL) methods is imperative. One of the most widely used techniques for tackling RL problems is Q-learning. In Q-learning, all actions and state pairs are executed, and values are stored in Q-tables. Upon convergence, the agent selects actions with max $Q_\theta(s, a)$ in the Q-table. As a result, Q-learning provides an optimal solution only for small action-state spaces.

In contrast, deep Q-learning (DQN) overcomes the dimensionality limitation of Q-learning for the discrete action space case, which leads to enhanced data efficiency and precision. Similar to Q-learning, DQN is suitable only for a finite action state space. Thus, to apply DQN in continuous action space, the action would need to be discretized, which increases the dimensionality of action space exponentially and reduces precision [9]. As a result of the above factors, DQN is not able to effectively handle problems involving continuous action such as ours. Due to the high dimension and continuous state-action space of the problem (10), we employed the deep deterministic policy gradient (DDPG) method.

The DDPG framework consists of an evaluation network (EN) and a target network (TN). The TN is an exact copy of EN, which helps to trace the learned networks slowly. This prevents the networks from becoming unstable during learning [9][1]. In addition, both networks contain actor-critic networks. During the training process, the correlation between transitions can reduce the convergence rate. Therefore, we adopted a replay buffer technique in which each transition (Z_t, a_t, R_t, Z_{t+1}) is saved in the replay buffer M. Following

[1]We can study the DDPG algorithm in details in [9].

that, mini-batches of size B are randomly selected from M to update the evaluation actor-critic network. The evaluation actor network generates continuous actions $a_t = \mu(z_t|\phi) + n_o$ by analyzing the policy $\mu(Z_t|\phi)$ and exploration noise. Also, ϕ denotes the actor weight parameter and a n_o represents the ornstein-uhlenbeck noise [10]. The main objective is optimizing the expected reward given by:

$$J(\phi) = \mathbb{E}\big[Q(Z,a)|Z = Z_t, a_t = \mu(Z_t|\phi)\big]. \tag{17}$$

The evaluation critic network uses Q-function to evaluate the policy $\mu(Z_t|\phi)$. Based on the Bellman equation, the updated Q-function can be defined as:

$$y_t = R_t + \gamma Q'(Z_{t+1}, \mu'(Z_{t+1}|\phi')|\theta'), \tag{18}$$

where ϕ' and θ' are the weight parameters of the target actor and critic network, respectively. The evaluation actor and the evaluation critic network weight parameters are updated according to a policy gradient, $\nabla_\phi J$ and loss function $L(\theta)$ [9].

$$\nabla_\phi J = \mathbb{E}_{Z_t}\big[\nabla_\phi \mu(z|\phi) \nabla_a Q(Z,a|\theta)\big]\Big|_{a=\mu(Z|\phi)}, \tag{19}$$

$$L(\theta) = \mathbb{E}_{Z_t}\big[y_t - (Q(Z_t, a_t|\theta))^2\big]. \tag{20}$$

In addition, the weight parameters ϕ' and θ' for the target actor and critic network, respectively, are updated based on the soft update rule[2].

$$\phi' \leftarrow \omega_a \phi + (1 - \omega_a)\phi', \tag{21}$$

and

$$\theta' \leftarrow \omega_c \theta + (1 - \omega_c)\theta'. \tag{22}$$

Algorithm 1 shows the summary of DDPG method. From lines 1 to 2 the evaluation and target network weight parameters and replay buffer M are initialized. In every episode, the agent obtains a new observation state Z_t and in each t, the agent chooses an action a_t to perform, executes it, and receives a reward R_t. Then it obtains its newly acquired observation state Z_{t+1}. Following this, in lines 8 to 9 experience transitions are stored in the replay buffer M, and a random set of transitions of size B are selected from M. In lines 10 to 11, the target and evaluation networks are updated.

IV. Simulation Results

We consider a base station located in the center of a 300m x 300m area with 13 UDs distributed uniformly within this area. We consider a log distance path loss model and the small-scale fading coefficient is assumed as a Rayleigh random variable, while the large-scale shadowing is computed based on the log-normal distribution with a standard deviation of 10 dB and a zero mean shadowing model [11]. A MEC server in the base station has a spectrum resource of 10MHz and a computation resource of 18GHz. The transmission power of each UD is set to 0.1W. UD has a 1GHz computational capacity. The UD selects a task consisting of random values of data size, CPU cycles, and tolerable delay. The ranges for data size, CPU cycles, and tolerable delay are set as $[1000, 1200]$Kb, $[1100, 1200]$CPU cycles per bit, and $[500, 1500]$ms, respectively [12].

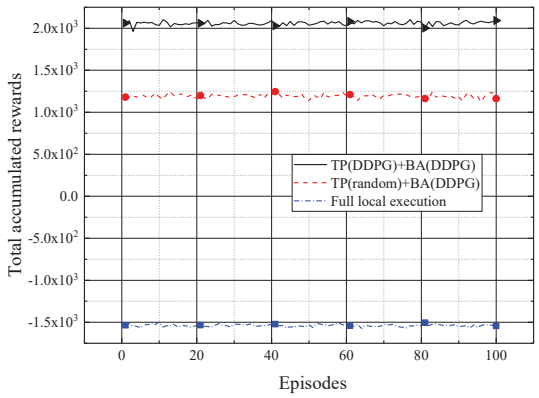

Fig. 2. Total accumulated rewards versus episode, UD = 13, tolerable delay = $[500, 1100]ms$

The actor network consists of two hidden layers. There are 256 and 128 neurons in the first and the second layer, respectively. The critic networks consist of 1024 and 512 neurons in the first and the second layer, respectively. Also, using extensive experimental settings, we evaluated the proposed scheme with a reasonable set of parameters that gave us a feasible network performance. For the input and hidden layers, we used ReLU activation functions and for the output layer, we used tanh activation functions[3]. After training with 4000 episodes, we tested the model with 100 episodes, each consisting of 200 steps. Replay buffers can store up to 10000 experience transitions and 32 mini-batches at a time. The learning rate for actor and critic network is set 0.001 and 0.01, respectively. We compared the proposed DDPG-based model with the following schemes.

1) Scheme 1-random task partitioning (TP) and DDPG–based bandwidth allocation (BA) (TP(random)+BA(DDPG)).
2) Scheme 2-(Full local execution): Tasks executed locally at UD.
3) The proposed scheme-(TP(DDPG)+BA(DDPG)): The task offloading decisions and bandwidth allocation is based on DDPG.

In Fig. 2, we compare the total rewards over 100 testing episodes. Each episode contains 200–time slots, so the total reward earned in the episode is the sum of the rewards earned in all 200-time slots. With the proposed scheme, the total reward accumulated in each episode is higher than with the other schemes. The proposed scheme accumulated approximately

[2]Soft updating refers to updating the target network frequently and slowly [9].

[3]The tanh activation function results in higher values of gradient during the training phase leading to a higher update values of trainable weights in the deep networks.

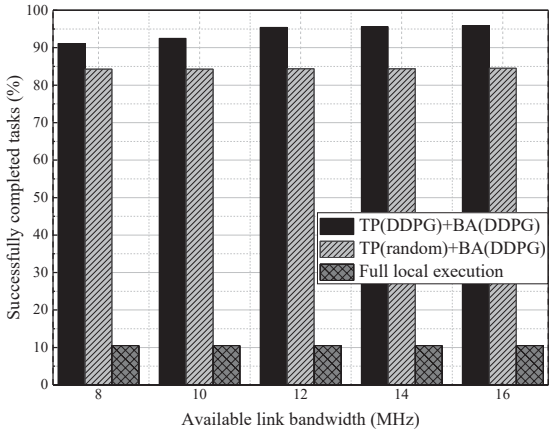

Fig. 3. Successfully completed task versus available link bandwidth between the MEC server and UD (MHz)

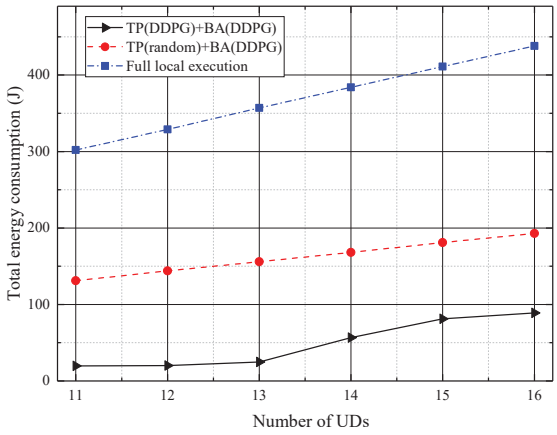

Fig. 4. Total energy consumption (J) versus the number of UDs

42.08% and 174.69% more rewards than schemes 1 and 2, respectively.

Figure 3 shows that the proposed scheme achieves superior performance in terms of successful task completion when the link bandwidth is varied from 8 to 16MHz, compared to the other schemes 1 and 2. For the proposed scheme and scheme 1, doubling the link bandwidth from 8 to 16 MHz results in a 5% and 0.24% increase in the percentage of tasks that are successfully completed. Since full local execution does not utilize the link bandwidth, the percentage of tasks completed remains unchanged. Using 16 MHz as the available link bandwidth, the proposed scheme completed 95.9% of offloaded tasks successfully. This is approximately 11.89% and 89.05% higher than schemes 1 and 2, respectively.

Figure 4 shows the amount of energy consumed by all the UDs in relation to the increasing number of UDs $[11, 16]$. When UD is 11, the proposed scheme has approximately 84.96% and 93.47% lower energy consumption than schemes 1 and 2, respectively. Likewise, when UD is 16, the proposed scheme has approximately 53.89%, 79.68% lower energy consumption than schemes 1 and 2.

V. Conclusion

In this paper, we proposed a partial task offloading and resource allocation scheme for multi–user MEC network. To reduce total energy consumption while maximizing the number of tasks completed within a tolerable time period, we leveraged a deep deterministic policy gradient (DDPG)–based solution. The MEC server received offloaded task requests with varying resource requirements from the UDs in the network. It then determined the optimal resource allocations and offloading decisions. The simulation results showed that the proposed method completes a greater number of tasks within a tolerable delay and reduces the energy consumption in the network, compared to those of other conventional schemes.

Acknowledgement

Min Young Chung is the corresponding author. This work was supported by the BK21 FOUR Project.

References

[1] X. Xu, B. Shen, X. Yin, M. R. Khosravi, H. Wu, L. Qi, and S. Wan, "Edge Server Quantification and Placement for Offloading Social Media Services in Industrial Cognitive IoV," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 4, pp. 2910–2918, 2021.

[2] L. A. Haibeh, M. C. E. Yagoub, and A. Jarray, "A Survey on Mobile Edge Computing Infrastructure: Design, Resource management, and Optimization Approaches," *IEEE Access*, vol. 10, pp. 27 591–27 610, 2022.

[3] Z. Yang, Y. Du, C. Che, W. Wang, H. Mei, D. Zhou, and K. Yang, "Energy–efficient Joint Resource Allocation Algorithms for MEC–enabled Emotional Computing in Urban Communities," *IEEE Access*, vol. 7, pp. 137 410–137 419, 2019.

[4] M. Chen and Y. Hao, "Task Offloading for Mobile Edge Computing in Software–defined Ultra–dense Network," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 3, pp. 587–597, 2018.

[5] Z. Sun and M. R. Nakhai, "An Online Learning Algorithm for Distributed Task Offloading in Multi–access Edge Computing," *IEEE Transactions on Signal Processing*, vol. 68, pp. 3090–3102, 2020.

[6] J. Wang, J. Hu, G. Min, A. Y. Zomaya, and N. Georgalas, "Fast Adaptive Task Offloading in Edge Computing based on Meta Reinforcement Learning," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 1, pp. 242–253, 2021.

[7] J. Ren and S. Xu, "DDPG–based Computation Offloading and Resource Allocation for MEC Systems with Energy Harvesting," in *Proc. 2021 IEEE 93rd Vehicular Technology Conference (VTC2021–Spring),* 2021, pp. 1–5.

[8] D. Sha and R. Zhao, "DRL–based Task Offloading and Resource Allocation in Multi–UAV–MEC Network with SDN," in *Proc. 2021 IEEE/CIC International Conference on Communications in China (ICCC)*, 2021, pp. 595–600.

[9] T. P. Lillicrap *et al.*, "Continuous Control with Deep Reinforcement Learning," 2015.

[10] G. E. Uhlenbeck and L. S. Ornstein, "On the Theory of the Brownian Motion," Phys. Rev., vol. 36, pp. 823–841, Sep 1930. [Online].

[11] D. S. Lakew, V. D. Tuong, N. -N. Dao and S. Cho, "Adaptive Partial Offloading and Resource Harmonization in Wireless Edge Computing-Assisted IoE Networks," in *IEEE Transactions on Network Science and Engineering*, doi: 10.1109/TNSE.2022.3153172.

[12] Z. Yang, C. Pan, K. Wang, and M. Shikh–Bahaei, "Energy efficient resource allocation in UAV–enabled mobile edge computing networks," *IEEE Transactions on Wireless Communications*, vol. 18, no. 9, pp. 4576–4589, 2019.