Frontiers in Computing and Intelligent Systems
ISSN: 2832-6024 | Vol. 6, No. 3, 2023

Computation Offloading and Resource Allocation in Mobile Edge Computing-enabled IoT Network

Hao Yang *, Huifu Zhang, Ziyi Gong

School of Computer Science and Engineering, Hunan University of Science and Technology, Xiangtan, Hunan, 411201, China
*** Corresponding author:** Hao Yang (Email: yanghaohnust@163.com)

Abstract: Mobile edge computation (MEC)-enabled Internet of Things (IoT) network have gained significant attention from academia and industry due to its ability to provide ultra-low latency computation services for several IoT applications such as VR/AR, smart city and online gaming, etc. This article explores the challenge problem of jointly addressing computation offloading and resource allocation (CORA) in a time-varying MEC-enabled IoT network. Firstly, we propose a multiuser and multiserver MEC-enabled IoT architecture based on centralized management for the decision-making process of CORA. Secondly, we formulate a task execution delay minimization problem considering energy consumption and resource constraints. It is challenging to solve this problem using traditional methods due to the coupling between the variables, as well as the dynamic nature of the network. To this end, we convert the original problem into a Markov Decision Process (MDP). Then, an entropy-based deep reinforcement learning algorithm (EDRL) with strong exploration capabilities and stability is used to learn the dynamic CORA strategy. Finally, extensive simulation experiments have indicated that our proposed EDRL method exhibits superior learning ability and stability compared to the DDPG. Except for the exhaustive method, EDRL outperforms other baselines with respect to execution delay and energy consumption.

Keywords: Computation Offloading; Mobile Edge Computing; Resource Allocation; Entropy; Deep Reinforcement Learning; Internet of Things (IoT).

1. Introduction

The convergence of the Internet of Things (IoT) and wireless communication technology has enabled a wide range of mobile services and applications, such as virtual/ augmented reality(V/AR), smart city, online gaming, etc. [1, 2]. With the progressive implementation of 5G cellular networks [3] and the emergence of 6G networks [4] these mobile applications will increasingly penetrate various aspects of our lives. However, it is impractical to directly execute these computation-intensive mobile services on resource-constrained IoT user devices (UDs). Researchers have introduced Mobile Cloud computation (MCC) to address this issue, which involves offloading the computational UDs' tasks to remote central cloud servers for processing [5]. Compared to IoT UDs, these cloud servers in MCC provide more abundant computation resources and greater storage capacity [6]. Nevertheless, due to the deployment of cloud servers far from end users, this approach results in high transmission delay, which is not feasible for delay-sensitive mobile applications. Furthermore, in the process of computation offloading in MCC, the increasing data volume puts significant pressure on wireless network transmission [7]. Fortunately, Mobile Edge computation (MEC) emerges as a new computation paradigm that can alleviate these challenges. Therefore, the MEC-enabled IoT network also comes into being.

In the task processing of IoT UDs, two issues require attention: firstly, determining the necessity of offloading tasks for processing on the MEC server; and secondly, allocating the appropriate resources to UDs that require task offloading. It is evident that due to the presence of binary offloading variables and resource allocation variables, CORA in MEC is considered as a Mixed Integer Nonlinear Programming (MINLP) problem, which is also NP-hard[15]. Some methods, such as the Alternating Direction Method of Multipliers (ADMM)-based approach [16] and heuristic methods[17–19], have been proposed to try to address the MINLP problem. While these methods can reduce algorithm complexity, they require numerous iterations to achieve local optimality. Once certain factors in the system change, such as bandwidth or the number of IoT UDs connecting to MEC servers, the optimal offloading and resource allocation problem needs to be solved again. Therefore, these methods are unsuitable for dynamic MEC network scenarios.

Deep learning (DL) has become a key tool for addressing various problems in machine learning. Many cutting-edge applications, such as autonomous vehicles and natural language processing, rely on DL to predict resource requirements and optimize their allocation[20]. In [21,22], the authors utilize the powerful feature extraction capabilities of DL to learn offloading strategies in MEC systems. However, DL presents challenges in MEC systems due to the need for large labeled datasets to train neural networks. In contrast, reinforcement learning (RL) does not rely on a dataset for model training, as the agent interacts with the dynamic MEC system and learns an optimal strategy without prior knowledge. Typical RL algorithms like SARSA and Q-learning store tuples containing state, action, and value in a Q-table. However, with an increasing number of states and actions, the methods for manipulating the Q-table become inefficient and may lead to the curse of dimensionality. Deep Reinforcement Learning (DRL) [23,24]addresses the issue by approximating the Q-table using a neural network combined with reinforcement learning techniques.

In this study, a MEC network framework is devised featuring an Edge Manager Node (EMN). The EMN is introduced to enhance CORA management efficiency in the MEC system, as well as reduce the overhead for UDs. Specifically, we design a problem of CORA, considering

constraints on computation resources, communication resources, and energy consumption, with the objective of minimizing execution delay. This problem is non-convex within each time interval. To address this issue, inspired by reference [25], we propose an entropy-based deep reinforcement learning (EDRL) algorithm for CORA in the MEC system. This algorithm incorporates an entropy term into the reinforcement learning reward to enhance its robustness and stability. Our primary contributions are summarized as follows:

1) We advocate a MEC architecture based on centralized management for the decision-making process of CORA. In this system, UDs can interact in real-time with the edge nodes to obtain reasonable decisions regarding CORA, which can reduce unnecessary computational overhead caused by making offloading decisions in each IoT UD.

2) We explore the proposed dynamic MEC system and formulate CORA as an MINLP problem, aiming to minimize task computation latency while considering energy consumption, power resources, and computation resources. It is challenging to solve this problem by traditional methods due to the large solution space and dynamic nature of MEC.

3) The original problem is converted into a Markov Decision Process (MDP). Then, an entropy-based Deep Reinforcement Learning (EDRL) is proposed to handle this model. EDRL enhances learning exploration and stability by introducing entropy during the learning process, achieving stable and efficient CORA strategies.

4) Extensive simulation experiments are conducted to validate the efficiency of our proposed EDRL. Simulation results demonstrate that the EDRL method exhibits superior learning ability and stability compared to the DDPG during the process of CORA. In addition, except for the exhaustive method, EDRL outperforms other baselines with respect to execution delay and energy consumption.

The rest of the article is shown below. Section 2 describes the proposed MEC system mode and the research problem. Section 3 describes the entropy-based DRL solution method. In Section 4, we provide the parameter settings for the simulation experiments and baselines and analyze the outcomes of the experiments. Finally, Section 5 encompasses the conclusions drawn from this article.

2. System Network Model and Problem Description

2.1. System Network Model

Our study focuses on a MEC-enabled IoT network depicted in Figure 1, which is composed of numerous IoT UDs, multiple edge computing nodes, and an edge control node. Let $\mathcal{N} = \{1,2,\dots,N\}$ represents the set of IoT UDs, and $\mathcal{M} = \{1,2,\dots,M\}$ denotes the set of edge nodes in the considered MEC system. Every IoT UD in the MEC system, equipped with limited computational capabilities, has the option to perform its computation tasks either locally or offload them to an edge node for processing. Additionally, each edge node corresponds to a small cellular base station (scBS) for communication with the user devices, and each scBS incorporates an MEC server to process tasks generated by the IoT UDs. Different from existing research, we introduce an EMN into the MEC network, responsible for executing CORA algorithms and sending the computation results to target UD. Similar to regular edge nodes, the EMN also has an scBS that deploys MEC servers for managing CORA. Specifically, to simplify the problem, we assume that the EMN has robust computation and storage resources and can parallelly process requests for CORA from different IoT UDs. In the MEC network framework, CORA schemes are handled as follows. Firstly, within each time period, each UD generates a task. Secondly, the EMN gathers system information, including the attributes of computation tasks, the remaining bandwidth resources of scBS, and the available computation resources of the MEC servers. After obtaining the relevant system information, the edge control node will execute the corresponding CORA algorithm. Based on the computation results received from the EMN, IoT UDs can undertake appropriate offloading actions.

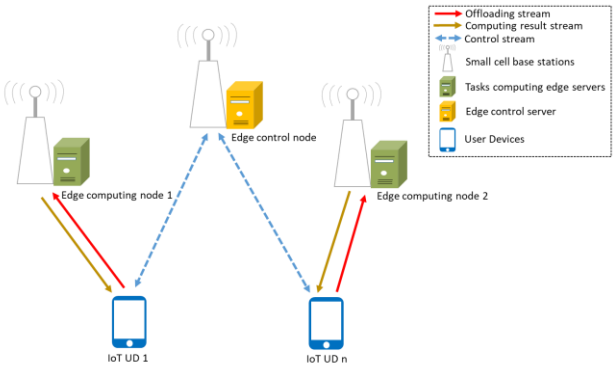

Figure 1. The MEC-enabled IoT network model with multiuser and multiserver

To characterize the MEC system, we discretize time into fixed-length time periods, denoted as $t \in \{1,2,\dots,T\}$.Within each time period, every IoT UD generates only one computation-intensive task. Furthermore,we employ a commonly used task model to establish the definition of execution tasks, which can be represented as $\Omega_n = \{\delta_n, \zeta_n, f_n^{local}, \tau_n\}$. Here, $\delta_n$represents the volume of data in the execution task, $\zeta_n$represents the total count of CPU cycles necessary to execute the task Ω_n ,and f_n^{local} represents the computation capability of IoT UD n. τ_n denotes task execution deadline for performing task Ω_n.The implication is that the task's execution time should not exceed a certain threshold, regardless of whether it is executed locally or on an MEC server. It should be noted that this study exclusively focuses on local task execution or offloading to MEC servers, without considering additional offloading to remote centralized clouds or other ENs. In this paper, within each time period, dynamically arriving computational tasks $\Omega_n$from UD are different, with varying computation intensities and input task sizes. To reduce execution delay and energy consumption, a task Ω_n can be processed locally or offloaded to an EN node.In addition, this study specifically addresses the scenario of binary offloading, where tasks cannot be further subdivided. An variable $x_{n,m}$ is applied for defining the offloading decision for the computational task Ω_n of UD n, where $x_{n,m} = 0$ represents the task being processed locally on UD n, and otherwise, the task $\Omega_n$is offloaded and executed on the EN m.

Table 1. Presents a summary of the key concepts used in this article

Concept	Description
$\mathcal{N}$	Set of user devices, indexed by n
$\mathcal{M}$	Set of edge nodes, indexed by m
T	The scale of time slot
Ω_n	Computational task of user device n
δ_n	Computational volume of the task Ω_n
ζ_n	CPU cycle count needed to perform the task Ω_n
f_n^{local}	Computational capability of user device n.
τ_n	Execution deadline of the task Ω_n
$x_{n,m}$	Offloading decision variable
$f_{n,m}^{MEC}$	Amount of computational resources assigned to the task $\Omega_n$by edge node m
$w_{n,m}$	Amount of bandwidth assigned to the task $\Omega_n$by edge node m
$R_{n,m}$	Data transfer rate between user device n and the edge node.
f_{MEC}	Computational capacity of the MEC servers

While offloading computational tasks to an EN can decrease task execution latency and energy consumption compared to local computation, offloading tasks also incur additional delay and energy costs due to data transfer. Therefore, when UD n offloads its task to an EN within a time slot, we take into account the transmission latency and energy consumption. Specifically, we neglect the communication cost between MEC server and scBS the and simplify the problem by not considering interference between neighboring cells. According to Shannon's theorem and referring to [26] and [27], the wireless transmission rate from UD n to EN m can be represented as:

$$R_{n,m} = w_{n,m}\log_2\left(1 + \frac{p_n g_n}{w_{n,m}\,\sigma}\right) \quad (1)$$

where p_n represents IoT UD n's transmission power when offloading the task Ω_n to EN m; σ denotes the noise power spectral density; $w_{n,m}$ represents the bandwidth allocated by EN m to UD n; g_n is the channel gain between IoT UD n and scBS.We assume that the mobility of the user device is not significant during offloading, so the channel gain of the user device is constant but may vary for different user devices. Based on the above description, the transmission delay associated with IoT UD n's decision to offload its execution task Ω_n to EN m can be represented as:

$$D_n^{tran} = \frac{\delta_n}{R_{n,m}} \quad (2)$$

Furthermore, the corresponding energy consumption for the task Ω_n can be obtained as follows:

$$E_n^{tran} = p_n D_n^{tran} \quad (3)$$

Typically, the data received by the IoT UD from the MEC server is significantly smaller in size compared to the transmitted data and has a higher download link rate [28]. Therefore, in this paper, we do not consider the transmission latency and energy consumption resulting from the computed results returned by the MEC servers of the edge nodes.

The task Ω_n can be executed either on the local IoT UD n or on the MEC server m.When $x_{n,m}= 0$, the computational task Ω_n of UD n will be processed locally. We use ε_n^{local} to represent the difference in energy consumption and computational capacity of heterogeneous IoT UDs. Consequently, the task latency can be represented as:

$$D_n^{local} = \frac{\zeta_n}{f_n^{local}} \quad (4)$$

And the local energy consumption of IoT UD n is calculated as follows:

$$E_n^{local} = \zeta_n \varepsilon_n^{local} \quad (5)$$

where $\varepsilon_n^{local} = 10^{-27}(f_n^{local})^2$ is related to the actual chip architecture[42].When $x_{n,m}= 1$, the task Ω_n of UD n is offloaded to EN m for processing. We utilize $f_{n,m}^{MEC}$ to define the computation resources allocated by EN m's MEC server to task Ω_n of UD n. Therefore, the task Ω_n' execution delay on the MEC server is written as:

$$D_n^{MEC} = \frac{\zeta_n}{f_{n,m}^{MEC}} \quad (6)$$

Furthermore, the energy consumption associated with offloading computational task Ω_n can be represented as:

$$E_n^{MEC} = P_{MEC}\, D_n^{MEC} \quad (7)$$

where P_{MEC} represents the power of the MEC server. According to equations (5) and (6), we can infer that when UD n is allocated more computation resources by EN m, the computation delay will be reduced, but the energy consumption will increase accordingly.

Similar to [26], we consider the energy cost of the idle mode for IoT UD n while waiting for the offloaded computation results, which can be calculated as:

$$E_n^{wait} = p_n^w D_n^{MEC} \quad (8)$$

where $p_n^w$is the power in idle mode for UD n. By combining equations (2), (3), (6), (7), and (8), the total delay and energy consumption for offloading computational task Ω_n of UD n within a given time slot can be easily obtained as follows:

$$D_n^{off} = D_n^{tran} + D_n^{MEC} \quad (9)$$

And

$$E_n^{off} = E_n^{tran} + E_n^{MEC} + E_n^{wait} \quad (10)$$

2.2. Problem Description

Based on the above content, we can calculate the delay cost of task Ω_n of UD n within a given time slot as:

$$D_n = (1 - x_{n,m})D_n^{local} + x_{n,m}D_n^{off} \quad (11)$$

And the corresponding energy consumption is:

$$E_n = (1 - x_{n,m})E_n^{local} + x_{n,m}E_n^{off} \quad (12)$$

To reduce system overhead and improve IoT UDs' QoS, we leverage the Edge control node to acquire the optimal CORA strategies between IoT UDs and ENs. Our objective is to minimize the average execution latency of the computational tasks in the entire MEC system, considering energy, computation resources, and communication resource constraints. To achieve this, the formulation of the optimization problem is represented as follows:

$$\min_{x_{n,m}, w_{n,m}, f_{n,m}^{MEC}} \frac{1}{NT}\sum_{t=0}^{T-1}\sum_{n=1}^{N} D_n \quad (13a)$$

$$\text{s.t. C1: } D_n \le \tau_n, \quad \forall n \in N \quad (13b)$$

$$\text{C2: } x_{n,m} \in \{0,1\}, \quad \forall n \in N \quad (13c)$$

$$\text{C3: } 0 \le \sum_{n=1}^{N} x_{n,m} f_{n,m}^{MEC} \le f_{MEC}, \quad \forall n \in N,\ m \in M \quad (13d)$$

$$\text{C4: } 0 \le \sum_{n=1}^{N} w_{n,m} \le w, \quad \forall n \in N, m \in M \quad (13e)$$

$$C5: 0 \le \sum_{m=1}^{M} x_{n,m} \le 1, \quad \forall n \in N, \; m \in M \tag{13f}$$

$$C6: 0 \le E_n \le e_n, \quad \forall n \in N \tag{13g}$$

The objective function (13a) minimizes the overall system execution delay during s. The first constraint (13b) ensures that the computation time does not exceed its tolerance delay. The second constraint (13c) guarantees the successful execution of each UD's computational task. Constraint (13d) ensures that the allocated computation resources for tasks executed on edge node m's server do not exceed the maximum value f_{MEC}. Constraint (13e) represents that the allocated bandwidth for tasks on EN should be less than or equal to the maximum value w. Constraint (13f) restricts each computation task to be computed by only one MEC server. Constraint (13g) guarantees that the energy consumption of tasks remains within the energy cost threshold.

Due to the interdependence and the need for simultaneous optimization of variables $x_{n,m}, w_{n,m}$, and $f_{n,m}^{MEC}$,the problem formulated is an NP-hard MINLP problem. For dynamic MEC systems with a large state-action space, it is challenging to obtain optimal CORA strategies using traditional methods such as convex optimization, heuristic and greedy algorithms, etc. To address this problem, we transform this problem into a MDP and employ an entropy-based deep reinforcement learning algorithm to deal with it.

3. Entropy-based Deep Reinforcement Learning Solution for CORA

3.1. Markov Decision Process Model

In this subsection, we transform the original problem into a MDP model that is represented by a quintuple$< \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma >$. $\mathcal{S}$ denotes the group of environmental states, $\mathcal{A}$ denotes the group of activities, $\mathcal{P}$ is the group of state transition probabilities, $\mathcal{R}$ is the reward function of state-actions. And $\gamma \in [0,1]$represents the discount factor related to the importance of future rewards.

In the MDP model, the agent acts as a decision-maker that interacts with the external environment. In this paper, we consider the edge control node as an intelligent agent. At the start of each time interval, the agent gathers whole state information. When a UD n has a new task to process, the agent determines a CORA action based on the collected state information. The agent is provided with a reward by the system environment, which is contingent on the current state and action. The agent continues this process and accumulates rewards until it obtains the optimal CORA decision that minimizes system costs. The edge control node sends this decision to the target UD n, which then decides whether to offload the computational task based on this decision. Next, we define the MDP elements related to the objective function problem.

State Space: The state space of the MDP reflects the present system environment within time period t. In our problem, the system state includes the task information of users, APs' resources in the EC, and ME's resource information. We define the state of the MDP using a vector $\mathcal{S} = \{s_t = \{\Omega_n(t), C(t), B(t), E_b(t)\}\}$, where $\mathcal{S}$ represents the local states of whole IoT UDs in time period t; $\Omega_n(t)$represents the execution task characteristics of IoT UD n in time period t; $C(t) = \{f_1^{MEC}, f_2^{MEC}, \dots, f_m^{MEC}\}$ and $B(t) = \{w_1, w_2, \dots, w_m\}$ represent the remaining computation capacity and remaining bandwidth resources of the EN set in time slot t, respectively. $E_b(t) = \{e_n\}$ denotes the energy expenditure of the system at time period t.

Action Space: The agent's goal is to establish a mapping between the state space and the action space. In our system, the agent is responsible for deciding the CORA schemes for different IoT UDs within time period t. As a result, the action vector comprises three components: the task offloading decision variable, the bandwidth resource allocation variable, and the computation resource allocation variable. We represent the action space of the agent using $\mathcal{A} = \{a_t = \{x_{n,m}, w_{n,m}, f_{n,m}^{MEC}\}\}(\forall n \in N, m \in M)$. Where $x_{n,m} \in \{0,1\}$ represents the decision regarding task offloading for the given task Ω_n at time period t. $w_{n,m}$ and $f_{n,m}^{MEC}$ represent the computation resource and bandwidth resource obtained by UD n from EN m at time t, respectively. To reduce the action space, we preprocess the computational tasks. At time t, for a task Ω_n coming from UD n, if the execution time does not exceed a predefined threshold, the task will be processed locally. Otherwise, the MEC server will handle the processing of the task through offloading. This approach reduces the dimensionality of the action space in the MDP, thus accelerating the convergence of the algorithm.

Reward Function: In every time period t, the agent takes actions in a given state, and then the current environment provides an instantaneous reward to the agent. In other words, the reward can be viewed as a mapping between the state space and action space. In RL, the aim is to maximize the overall cumulative reward. However, according to equation (13a), our goal is to minimize the average task execution delay across the whole MEC system while considering energy and resource limitations. Therefore, we define the instantaneous reward $\mathcal{R}(r_t(s_t, a_t))$ for a computational task as follows:

$$r(s_t, a_t) = -\sum_{n=1}^{N} D_n^t \tag{14}$$

Policy: a policy π can be described as $\pi(a_t|s_t) \in [0,1]$, which corresponds to the probability of selecting an action a_t in the current state s_t.In order to minimize the system's long-term delay, we need to find an optimal policy for computational tasks in different time slots, which can be represented as follows:

$$\pi^* = \arg\max_{\pi} \mathbb{E}[\sum_{t=1}^{\infty} \gamma^{t-1} \mathcal{R}(s_t, a_t)] \tag{15}$$

where $\mathbb{E}$ represents the expectation. Under the given policy, the average system delay can be represented by a Q-function, which can be understood as the expected discounted reward associated with choosing action a_t in the initial state s_t. It can be expressed as:

$$Q_\pi(s_t, a_t) = \mathbb{E}[r(s_t, a_t) + \gamma \mathcal{R}(s_{t+1}, a_{t+1}) + \dots | s_t, a_t \tag{16}$$

3.2. Entropy-based Deep Reinforcement Learning Algorithm for CORA

Policy that minimizes system delay can be achieved through value and policy iterations. In [25], an efficient and stable DRL framework based on entropy is put forward to handle high-dimensional continuous state-action space problems. Inspired by this approach, we adopt an entropy-

based DRL(EDRL) scheme to obtain the optimal CORA policies in the MEC system. Specifically, by incorporating the concept of entropy, our approach achieves higher exploration capability and stability.

Entropy serves as a metric for quantifying the degree of randomness and uncertainty in a stochastic phenomenon. Its applications span across diverse fields including information theory, physics, statistics, and artificial intelligence, etc. [29]. The magnitude of information entropy can be used to reflect the level of exploration, where a higher entropy indicates stronger exploration capability. Therefore, we incorporate an entropy term into the original reward, resulting in an entropy reward, which can be expressed as:

$$R_{t+1}^{\alpha} = R_{t+1} + \alpha H(\pi(\cdot|s_t)), t = 0,1,\ldots \quad (17)$$

where the parameter $\alpha > 0$ regulates the extent of policy randomness. $H(\pi(\cdot|s_t)) = -\log \pi(\cdot|s_t)$ denotes the entropy of the policy in a given state. By introducing entropy into the reward, it enhances exploration capability as entropy leads to more uniform policy outputs and learning closer to optimal actions. More specifically, certain states may be associated with multiple optimal actions due to having similar Q-values, so these actions may be assigned equal probabilities. Additionally, to prevent repeated selection of the same action and avoid suboptimal results, none of these optimal actions are assigned excessively high probabilities. This strategy promotes faster learning. Furthermore, we can obtain the long-term reward with an entropy term as follows:

$$U_t^{\alpha} = R_1^{\alpha} + \gamma R_2^{\alpha} + \ldots = \sum_{t=0}^{T-1} \gamma^t R_{t+1}^{\alpha} \quad (18)$$

The Q-function and state value function are denoted as follows:

$$Q_{\pi}^{\alpha}(s_t, a_t) = \mathbb{E}_{\pi}[U_t^{\alpha}|s_t, a_t], s_t \in \mathcal{S}, a_t \in \mathcal{A} \quad (19)$$

and

$$V_{\pi}^{\alpha}(s_t) = \mathbb{E}_{\pi}[U_t^{\alpha}|s_t], s_t \in \mathcal{S} \quad (20)$$

Moreover, in order to approximate the state value function, Q-function, and policy function, we utilize three distinct Deep Neural Networks (DNNs) denoted as θ, ω, and ϕ. Therefore, for a given pair of state and action, the state value function can be represented as:

$$V_{\theta}(s_t) = \mathbb{E}_{\mathcal{A}\sim\pi}[Q_{\pi}^{\alpha}(s_t, a_t)] + \alpha H(\pi(\cdot|\mathcal{S})) \quad (21)$$

Similarly, the equation for Bellman backup based on entropy is computed by incorporating entropy into the reward. Then, the Q-function is expressed as:

$$Q_{\omega}(s_t, a_t) = Q_{\pi}^{\alpha}(s_t, a_t) + \alpha H(\pi(\cdot|\mathcal{S})) \quad (22)$$

In particular, according to equation (16), the optimal deep policy can be represented as:

$$\pi_{\phi}(a_t|s_t) = \arg\max_{\pi} \mathbb{E}_{(s_t,a_t)\sim\rho(\pi)}[\sum_{t=0}^{T-1} \gamma^t (r(s_t, a_t) - \alpha \log(\pi(\cdot|s_t)))] \quad (23)$$

where $(s_t, a_t) \sim \rho(\pi)$ denotes the distribution of trajectories following a policy $\pi(a_t|s_t)$. As a result, given the optimal policy $\pi_{\phi}(a_t|s_t)$, the Bellman optimality equation for a Q-function $Q_{\omega}(s_t, a_t)$ related to the value function $V_{\theta}(s_t)$ can be written as:

$$Q_{\omega}(s_t, a_t) = \mathbb{E}_{a_{t+1}\sim\pi}\left[r(s_t, a_t) + \gamma V_{\theta}(s_{t+1})\right] \quad (24)$$

By incorporating a broader entropy objective into the design of random offloading strategies, the agent can randomly select a set of offloading actions to ensure that no useful set of offloading operations is overlooked. After constructing the relevant functions, we proceed to describe the training process of the DRL algorithm. The network framework of the EDRL algorithm is depicted in Figure 2. It mainly comprises an actor network and a critic network. Where the actor network, also known as the policy network, plays the role of selecting action a_t given the state s_t and policy π_{ϕ}. In addition, the critic, referred to as the value network, is tasked with assessing the value of actions selected by the actor network and generates evaluation value $Q_{\omega}(s_t, a_t)$ used to adjust the actor network. Detailed explanations of the training procedures for both the actor network and critic network will be provided below.

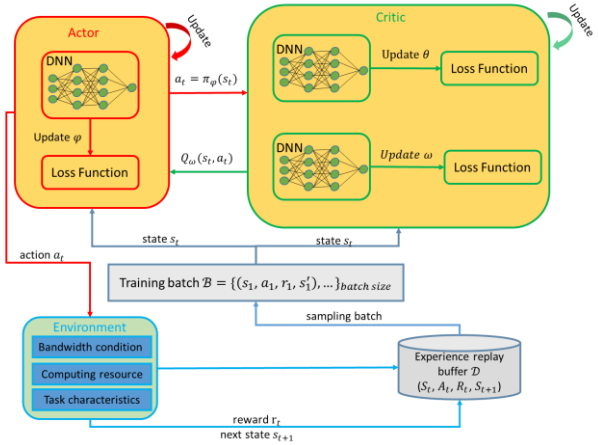

Figure 2. EDRL algorithm architecture

Before each training iteration, we initiate the process by randomly selecting a batch from the experience replay buffer. The experience replays buffer stores past experiences, which include records of the present state, action, immediate reward, and the next state of the environment, thereby improving the utilization of state-action pairs [5]. To update the Q-critic network with the assistance of the optimal Bellman equation, mean squared error (MSE) is used as the loss function to fit a Q-function $Q_{\omega}(s_t, a_t)$ to the Q-target function $Q_{\omega^-}(s_t, a_t)$.Hence, the objective function of the Q-function is expressed as:

$$L_Q(\omega) = \frac{1}{2|\mathcal{B}|}\sum_{k=1}^{\mathcal{B}}[Q_{\omega}(s_k, a_k) - Q_{\omega^-}(s_k, a_k)]^2 \quad (25)$$

The objective function $Q_{\omega^-}(s_k, a_k)$ satisfies equation:

$$Q_{\omega^-}(s_k, a_k) = r(s_k, a_k) + \gamma\left(Q_{\omega^-}(s_{k+1}, a_{k+1}) - \alpha \log\left(\pi_{\phi}(a_{k+1}|s_{k+1})\right)\right) \quad (26)$$

where ω^- represents the parameters of the Q-critic target network. Furthermore, to obtain values close to the optimum for $Q_{\omega}(s_t, a_t)$,we employ the most common gradient descent method to update the parameters ω by minimizing $L_Q(\omega)$. The gradient $L_Q(\omega)$ can be written as:

$$\nabla_{\omega} L_Q(\omega) = \frac{1}{|\mathcal{B}|}\sum_{k=1}^{\mathcal{B}} -\nabla_{\omega} Q_{\omega}(s_k, a_k) r(s_k, a_k) + \gamma (Q_{\omega}^{-}(s_{k+1}, a_{k+1}) - \alpha \log(\pi_{\phi}(a_{k+1}|s_{k+1}))) \quad (27)$$

Finally, the parameters of the Q-critic network can be updated in the following manner:

$$\omega \leftarrow \omega - \lambda_{\omega} \frac{1}{|\mathcal{B}|}\sum_{\mathcal{B}} L_Q(\omega), \quad \omega^{-} \leftarrow \epsilon\omega + (1-\epsilon)\omega^{-}, \quad \epsilon \in [0,1] \quad (28)$$

where λ_{ω} denotes the learning rate of the Q-critic network.

Similar to a Q-function $Q_{\omega}(s_t, a_t)$, the V-critic network is employed to approximate the value function $V_{\theta}(s_t)$ and its parameters are updated also by MSE and stochastic gradient descent. The objective function of the V critic network can be formulated as:

$$L_V(\theta) = \frac{1}{2\mathcal{B}}\sum_{k=1}^{\mathcal{B}} [V_{\theta}(s_k) - (Q_{\omega}(s_k, a_k) - \alpha \log(\pi_{\phi}(a_k|s_k))]^2 \quad (29)$$

Furthermore, the gradient descent for $L_V(\theta)$ can be expressed as:

$$\nabla_{\theta} L_V(\theta) = \frac{1}{\mathcal{B}}\sum_{k=1}^{\mathcal{B}} \nabla_{\theta} V_{\theta}(s_k)[V_{\theta}(s_k) - Q_{\omega}(s_k, a_k) + \alpha \log(\pi_{\phi}(a_k|s_k)] \quad (30)$$

Finally, the V-critic network can be updated as follows:

$$\theta \leftarrow \theta - \lambda_{\theta} \nabla_{\theta} \frac{1}{|\mathcal{B}|}\sum_{\mathcal{B}} L_V(\theta) \quad (31)$$

where λ_{θ} denotes the learning rate of a Q-critic network.

Combining equations (24) and (25), we update the policy network using $Q_{\omega}(s_t, a_t)$. In order to prevent being trapped in local optima and promote exploration in state s_t, we reset the parameters of the action $a_t = f_{\phi}(\iota_t; s_t)$. Additionally, we set $f_{\phi}(\iota_t; s_t) = f_{\phi}^{\mu}(s_t) + \iota_t \odot f_{\phi}^{\sigma}(s_t)$, where $f_{\phi}^{\mu}(s_t)$ and $f_{\phi}^{\sigma}(s_t)$ respectively represent the mean and variance of a Gaussian distribution, with ι_t representing a Gaussian sampling noise vector. Therefore, an optimal policy is acquired by updating the policy parameters with the Kullback-Leibler (KL) divergence, given by the following equation:

$$\arg\min D_{KL}[\pi_{\phi}(\cdot|s_k)||\frac{\exp(\frac{1}{\alpha}Q_{\omega}(s_t,\cdot))}{Z_{\omega}(s_t)}] \quad (32)$$

where $D_{KL}(c||d)$ represents the KL divergence that is used to evaluate the difference between distributions c and d. $Z_{\omega}(s_t)$ represents the sum of $Q_{\omega}(s_t, a_t)$, which normalizes the state s_t. Therefore, combining equation (32), the policy parameters that incorporate the KL divergence can be expressed as:

Algorithm 1: EDRL algorithm for the CORA.

Initializing parameters:
$\varphi, \omega, \omega^{-}$ and θ,experience replay buffer $\mathcal{D}$, batch size for sampling $\mathcal{B}$, initial state s, and maximum iteration *T*.
for each episode in episodes **do**
  Initializing the simulation environment;
  Randomly generate an start-state s.
  for t from 0 to *T*-1 **do**
    Based on the current state of the environment, the actor network selects an action $a_k \sim \pi_{\varphi}(\cdot|s_t)$;
    Selecting the next state s_{t+1} and reward r(s_t,a_t);
    Storing s_t, a_t, r_t, s_{t+1}) in the experience replay buffer $\mathcal{D}$.
    Updating state s_t=s_{t+1};
    Sampling a batch of samples $\mathcal{B}$ from the experience replay buffer by random selection $\mathcal{B}$ $\{(s_t, a_t, r_t, s_{t+1})\}$;
    Updating the Q-function parameters of the Q-critic network ω:
    $\omega \leftarrow \omega - \lambda_{\omega} \frac{1}{|\mathcal{B}|}\sum_{\mathcal{B}} L_Q(\omega)$;
    Updating the parameters of the V-critic network θ:
    $\theta \leftarrow \theta - \lambda_{\theta} \nabla_{\theta} \frac{1}{|\mathcal{B}|}\sum_{\mathcal{B}} L_V(\theta)$;
    Updating the parameters of the actor network φ:
    $\varphi \leftarrow \varphi - \lambda_{\varphi} \nabla_{\varphi} \frac{1}{|\mathcal{B}|}\sum_{\mathcal{B}} L(\varphi)$;
    Updating the parameters of the target network ω^{-}:
    $\omega^{-} \leftarrow \epsilon\omega + (1-\epsilon)\omega^{-}$;
    if s_{t+1} is the terminal state **then**
      break;
    else
      continue;
    end if
  end for
end for

$$L(\phi) = \alpha \log \pi_{\phi}(f_{\phi}(\iota_k; s_k)|s_k) - Q_{\omega}(s_k, f_{\phi}(\iota_k; s_k)) \quad (33)$$

Furthermore, the gradient based on action sampling about $L(\phi)$can be expressed as follows:

$$\nabla_{\phi} L(\phi) = \nabla_{\phi} \alpha \log \pi_{\phi}(a_k|s_k) + \nabla_{\phi} f_{\phi}(\iota_k; s_k)[\nabla_a \alpha \log \pi_{\phi}(a_k|s_k) - \nabla_a Q(s_k, a_k)] \quad (34)$$

Finally, the parameters of the actor network can be updated as follows:

$$\phi \leftarrow \phi - \lambda_{\phi} \nabla_{\phi} \frac{1}{|\mathcal{B}|}\sum_{\mathcal{B}} L(\phi) \quad (35)$$

where λ_{ϕ} denotes the learning rate of the Q-critic network. The specific description of the EDRL algorithm is shown in Algorithm 1.

Table 2. Simulation Parameters

Parameter Description	Value
Amount of IoT user device n	[5,30]
Amount of edge node m	5
Transmission bandwidth w	8 MHz
Wireless transmission power p_n	25 dBm
Noise power spectral density σ	-160 dBm/Hz
Channel gain g_n	126+35log(L)
Computing capability of user device n f_n^{local}	[0.5,1.5] GHz
Computing Capability of Each MEC Server f_{MEC}	9GHz
Standby Power of User Device n p_n^w	0.02W

4. Simulation Results and Discussion

4.1. Simulation Setup

We have set our simulation parameters in the following manner. In each time period, the data volume of computing task Ω_n for UD n is denoted as $\delta_n \in [15,95]$Mbit, and the respective CPU cycle requirements for $\zeta_n \in [1000,4000]$ (Megacycles). In this paper, we conssider that the computing capacity of different UDs follows a uniform distribution [0.5, 1.5]. Just like [26], the channel gain can be represented as 126 + 35log(L), where L represents the distance between UD and EN. In addition, we set the empirical replacement buffer capacity to $\mathcal{D}$=5000, the batch size for sampling the training network to 256, the discount factor to 0.9, and the learning rate to 0.001. The main evaluation parameter settings are listed in Table 2.

To evaluate the performance of the EDRL algorithm, we introduce four baselines as follows:

Local Execution: IoT UDs perform whole tasks on their local devices.

Edge Execution: IoT UDs offload all computational tasks to MEC servers for execution.

DDPG: Deep Deterministic Policy Gradient (DDPG) is a classical RL algorithm used to handle large continuous action spaces. In this paper, we use it as a baseline for comparison with our EDRL method.

Exhaustion: The exhaustive method, also known as a brute force search approach, offers an approximate optimal solution and serves as an upper limit for other methods. However, implementing this method in large-scale systems is challenging.

4.2. Results and Discussion

As shown in Figure 3, we compare the convergence performance of the EDRL and DDPG, with 10 User Devices in the system. From Figure 3, it is easy to observe that both DDPG and EDRL converge at around 500 episodes. However, the convergence reward of EDRL is higher than that of DDPG, indicating that EDRL has stronger learning capability compared to DDPG. Additionally, EDRL achieves a relatively stable reward value at approximately 1000 episodes, while DDPG exhibits larger fluctuations in reward value after convergence. This is because, compared to the deterministic policy-based DDPG method, EDRL introduces entropy during the learning process of the CORA in the dynamic MEC system. This introduction of entropy brings more exploration and stability factors to the algorithm.

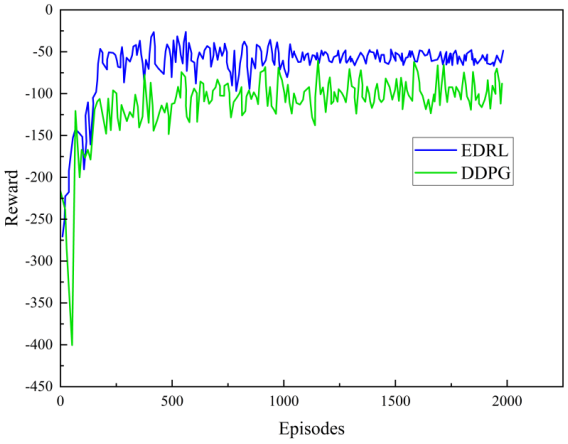

Figure 3. Comparison of convergence performance between DDPG and EDRL

With 10 IoT UDs and a task deadline of 3s, Figure 4 depicts the performance comparison of different methods in terms of latency and energy consumption as the size of the tasks increases. We increase the size of the tasks from 15 Mbit to 95 Mbit. From the figure, it can be noticeable that the average latency and average energy consumption of the six methods increase with the workload. It is evident that the our EDRL method is very close to the approximate optimal exhaustive method. Specifically, when the workload is 95 Mbit, compared to edge execution, local execution, and DDPG, the EDRL method reduces the average latency by 64%, 61%, and 7.3% respectively, and correspondingly reduces the average energy consumption by 69.2%, 75.7%, and 23.9%.

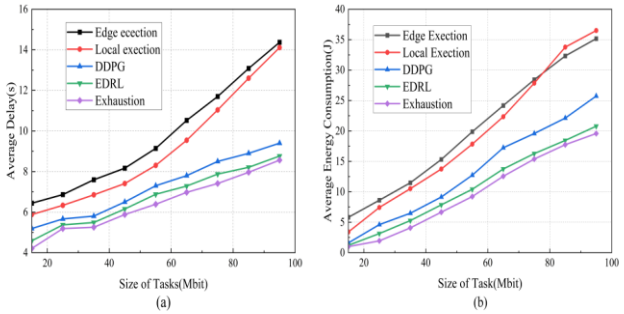

Figure 4. Performance comparison for different size of tasks:(a) Average Delay;(b) Average Energy Consumption

This indicates that the proposed EDRL method is effective in large-scale computational tasks within the dynamic MEC system. Furthermore, due to making appropriate CORA decisions based on system changes, DDPG outperforms edge execution and local execution. However, due to inferior exploration capability and stability compared to EDRL, the transfer of EDRL performs better than DDPG. However, due to inferior exploration capability and stability compared to EDRL, the performance of EDRL is better than that of DDPG. Additionally, edge execution and local execution experience the fastest growth, primarily due to the increasing strain on computing and communication resources as the workload increases, resulting in significant latency and energy consumption. It is worth noting that local execution performs better when local resources are sufficient, while edge execution is more favorable when resources at the edge are abundant.

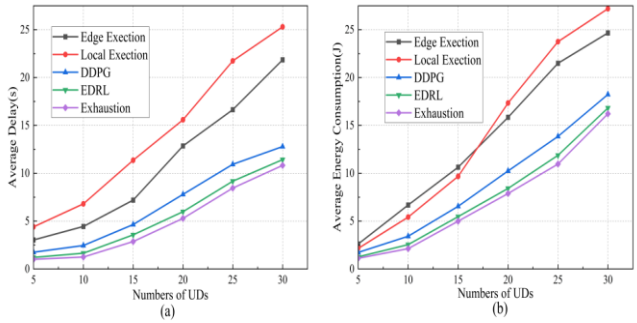

Figure 5. Performance comparison for different numbers of IoT UDs:(a) Average Delay, (b) Average Energy Consumption

Next, we vary the amount of IoT UDs to compare the performance of different methods in terms of latency and energy consumption. We change the number of IoT UDs from 5 to 30. As shown in Figure 5, it can be observed that as the amount of IoT UDs increases, the average latency and average energy consumption of each method also increase. From the figure, it is notable that except for the exhaustive

method, the EDRL outperforms the other three methods. The reason for this is that the EDRL method can actively engage with the dynamic MEC environment and acquire nearly optimal CORA strategies by leveraging entropy. Specifically, when the amount of IoT UDs is 30, compared to edge execution, local execution, and DDPG, the EDRL method reduces the average latency by 92.1%, 121.5%, and 11.9% respectively. Furthermore, from Figure 5, we can deduce easily that offloading computation tasks to the edge for execution is more advantageous when the amount of IoT UDs is higher. This is due to the constrained computing resources of IoT UDs, which leads to a decrease in allocated resources for computational tasks. Consequently, it results in a notable rise in both computation delay and energy consumption.

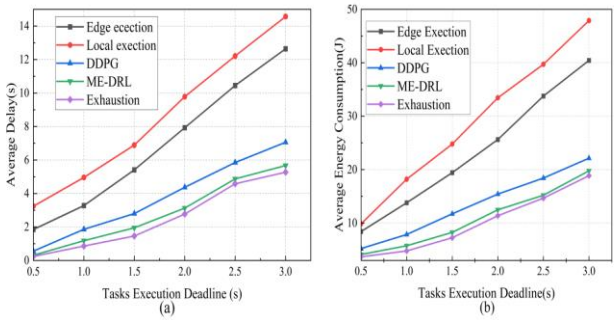

Figure 6. Performance comparison for different task execution deadline:(a) Average Delay, (b) Average Energy Consumption

Furthermore, in Figure 6, a comparison of the average delay and average energy consumption for different methods is presented at various task execution deadline. As expected, we can observe that all six methods show an increasing trend as the task execution deadline increases. EDRL performs the best among all methods, except for the exhaustive method, while DDPG performs second-best. It is worth noting that a larger task execution deadline indicates a higher tolerance for task execution latency. In such cases, tasks tend to be processed locally, leading to resource constraints on UDs and resulting in significant latency and energy consumption. This is why local execution exhibits the lowest performance in Figure 6.

5. Conclusion

In this article, we study the problems of computation offloading and resource allocation (CORA) in a time-varying MEC-enabled IoT network. Firstly, we propose a multiuser and multiserver MEC-enabled IoT network architecture with a centralized edge management node for the decision process offloading computation and resource allocation. Then, a CORA problem is designed to minimize the average task execution delay while considering energy consumption and available resource. Considering the dynamic characteristics of the system and the coupling between variables, it is challenging to handle this problem by traditional methods such as heuristic algorithms and convex optimization approaches. To overcome this issue, we convert the original problem into a MDP. Further, an entropy-based DRL (EDRL) approach is introduced to handle the constructed MDP model. Extensive simulation experiments have been conducted to validate the performance of proposed method. The simulation results showed that proposed EDRL method outperforms other baselines in terms of computation delay and energy consumption, except for the exhaustive method. Furthermore, our proposed method achieves also better convergence characteristics compared to DDPG. In the future, we will continue to explore more realistic scenarios for computational offloading in MEC-enabled IoT system, such as considering IoT UDs mobility, task partitioning, and task dependencies, etc.

References

[1] P. Bellavista, J. Berrocal, A. Corradi, S.K. Das, L. Foschini, A. Zanni, A survey on fog computing for the Internet of Things, Pervasive and Mobile Computing. 52 (2019) 71–99. https://doi.org/10.1016/j.pmcj.2018.12.007.

[2] H. Tan, An efficient IoT group association and data sharing mechanism in edge computing paradigm, Cyber Security and Applications.1(2023)100003.https://doi.org/10.1016/j.csa.2022.100003.

[3] E. Liu, E. Effiok, J. Hitchcock, Survey on health care applications in 5G networks, IET Communications. 14 (2020) 1073–1080. https://doi.org/10.1049/iet-com.2019.0813.

[4] F. Mogyorosi, P. Babarczi, J. Zerwas, A. Blenk, A. Pasic, Resilient Control Plane Design for Virtualized 6G Core Networks, IEEE Trans. Netw. Serv. Manage. 19 (2022) 2453–2467. https://doi.org/10.1109/TNSM.2022.3193241.

[5] T. Alfakih, M.M. Hassan, A. Gumaei, C. Savaglio, G. Fortino, Task Offloading and Resource Allocation for Mobile Edge Computing by Deep Reinforcement Learning Based on SARSA,IEEEAccess.8(2020)54074–54084. https:// doi.org/10. 1109/ACCESS.2020.2981434.

[6] H. Wu, K. Wolter, P. Jiao, Y. Deng, Y. Zhao, M. Xu, EEDTO: An Energy-Efficient Dynamic Task Offloading Algorithm for Blockchain-Enabled IoT-Edge-Cloud Orchestrated Computing, IEEE Internet Things J. 8 (2021) 2163–2176. https://doi.org/10.1109/JIOT.2020.3033521.

[7] D.B. Son, T.H. Binh, H.K. Vo, B.M. Nguyen, H.T.T. Binh, S. Yu, Value-based reinforcement learning approaches for task offloading in Delay Constrained Vehicular Edge Computing, Engineering Applications of Artificial Intelligence. 113 (2022) 104898. https://doi.org/10.1016/j.engappai.2022.104898.

[8] L.-A. Phan, D.-T. Nguyen, M. Lee, D.-H. Park, T. Kim, Dynamic fog-to-fog offloading in SDN-based fog computing systems, Future Generation Computer Systems. 117 (2021) 486–497. https://doi.org/10.1016/j.future.2020.12.021.

[9] A. Mukherjee, D. De, D.G. Roy, A Power and Latency Aware Cloudlet Selection Strategy for Multi-Cloudlet Environment, IEEETrans.CloudComput.7(2019)141154.https://doi.org/10.1109/TCC.2016.2586061.

[10] D. Niu, Y. Li, Z. Zhang, B. Song, A service collaboration method based on mobile edge computing in internet of things, Multimed Tools Appl. 82 (2023) 6505–6529. https://doi. org/10.1007/s11042-022-13394-x.

[11] Z. Zhang, W. Zhang, F.-H. Tseng, Satellite Mobile Edge Computing: Improving QoS of High-Speed Satellite-Terrestrial Networks Using Edge Computing Techniques, IEEE Network. 33 (2019) 70–76. https://doi.org/ 10.1109/MNET. 2018.1800172.

[12] Q. Li, S. Wang, A. Zhou, X. Ma, F. Yang, A.X. Liu, QoS Driven Task Offloading with Statistical Guarantee in Mobile Edge Computing, IEEE Trans. on Mobile Comput. (2020) 1–1. https://doi.org/10.1109/TMC.2020.3004225.

[13] Y.-H. Hung, C.-Y. Wang, R.-H. Hwang, Optimizing Social Welfare of Live Video Streaming Services in Mobile Edge Computing, IEEE Trans. on Mobile Comput. 19 (2020) 922–934. https://doi.org/10.1109/TMC.2019.2901786.

[14] Y. Zhan, S. Guo, P. Li, J. Zhang, A Deep Reinforcement Learning Based Offloading Game in Edge Computing, IEEE Trans. Comput. 69 (2020) 883–893. https://doi. org/10. 1109/ TC.2020.2969148.

[15] X. Chen, L. Jiao, W. Li, X. Fu, Efficient Multi-User Computation Offloading for Mobile-Edge Cloud Computing, IEEE/ACM Trans. Networking. 24 (2016) 2795–2808. https:// doi.org/ 10.1109/TNET.2015.2487344.

[16] C. Li, M. Song, H. Tang, Y. Luo, Offloading and system resource allocation optimization in TDMA based wireless powered mobile edge computing, Journal of Systems Architecture. 98 (2019) 221–230. https:// doi.org/ 10.1016/j. sysarc. 2019.07.009.

[17] K. Guo, M. Yang, Y. Zhang, X. Jia, Efficient resource assignment in mobile edge computing: A dynamic congestion-aware offloading approach, Journal of Network and Computer Applications. 134 (2019) 40–51. https://doi.org/ 10.1016/j. jnca.2019.02.017.

[18] F. Wang, J. Xu, S. Cui, Optimal Energy Allocation and Task Offloading Policy for Wireless Powered Mobile Edge Computing Systems, IEEE Trans. Wireless Commun. 19 (2020) 24432459.https://doi.org/10.1109/TWC.2020.2964765.

[19] P.-Q. Huang, Y. Wang, K. Wang, Z.-Z. Liu, A Bilevel Optimization Approach for Joint Offloading Decision and Resource Allocation in Cooperative Mobile Edge Computing, IEEE Trans. Cybern. 50 (2020) 4228–4241. https://doi.org/ 10. 1109/TCYB.2019.2916728.

[20] C.S. Chidume, C.O. Nnamani, Intelligent user-collaborative edge device APC-based MEC 5G IoT for computational offloading and resource allocation, Journal of Parallel and Distributed Computing. 169 (2022) 286–300. https://doi.org/ 10. 1016/j.jpdc.2022.07.007.

[21] S. Yang, G. Lee, L. Huang, Deep Learning-Based Dynamic Computation Task Offloading for Mobile Edge Computing Networks, Sensors.22(2022) 4088.https:// doi.org/ 10.3390/ s22114088.

[22] R. Du, C. Liu, Y. Gao, P. Hao, Z. Wang, Collaborative Cloud-Edge-End Task Offloading in NOMA-Enabled Mobile Edge Computing Using Deep Learning, J Grid Computing. 20 (2022) 14. https://doi.org/10.1007/s10723-022-09605-2.

[23] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, M. Riedmiller, Playing Atari with Deep Reinforcement Learning, (2013). http://arxiv.org/ abs/ 1312. 5602 (accessed July 30, 2023).

[24] T.P. Lillicrap, J.J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, D. Wierstra, Continuous control with deep reinforcementlearning, (2019). http://arxiv. org/abs/ 1509. 02971 (accessed July 30, 2023).

[25] T. Haarnoja, A. Zhou, P. Abbeel, S. Levine, Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor, in: J. Dy, A. Krause (Eds.), Proceedings of the 35th International Conference on Machine Learning, PMLR, 2018: pp. 1861–1870. https:// proceedings. mlr. press/v80/haarnoja18b.html.

[26] H. Zhou, K. Jiang, X. Liu, X. Li, V.C.M. Leung, Deep Reinforcement Learning for Energy-Efficient Computation Offloading in Mobile-Edge Computing, IEEE Internet Things J.9(2022)1517-1530.https:// doi.org/ 10.1109/ JIOT. 2021. 3091142.

[27] I.A. Elgendy, W.-Z. Zhang, H. He, B.B. Gupta, A.A. Abd El-Latif, Joint computation offloading and task caching for multi-user and multi-task MEC systems: reinforcement learning-based algorithms, Wireless Netw. 27 (2021) 2023–2038. https:// doi.org/10.1007/s11276-021-02554-w.

[28] J. Yan, S. Bi, Y.J.A. Zhang, Offloading and Resource Allocation With General Task Graph in Mobile Edge Computing: A Deep Reinforcement Learning Approach, IEEE Trans. WirelessCommun.19(2020)5404-5419. https:// doi.org/ 10. 1109/TWC.2020.2993071.

[29] Y. Pourasad, R. Ranjbarzadeh, A. Mardani, A New Algorithm for Digital Image Encryption Based on Chaos Theory, Entropy. 23 (2021) 341. https://doi.org/10.3390/e23030341.