Entropy-based Reinforcement Learning for Computation Offloading Service in Software-Defined Multi-access offloading execution

Kexin Li[a], Xingwei Wang[a*], Qiang Ni[b], Min Huang[c]

[a] College of Computer Science and Engineering, Northeastern University, Shenyang, Liaoning 110819, China

[b] School of Computing and Communications, Lancaster University, Lancaster, U.K

[c] College of Information Science and Engineering, Northeastern University, Shenyang, Liaoning 110819, China

Abstract

The rapid growth of Internet of Things (IoT) devices and the emergence of multiple edge applications have resulted in an explosive growth of data traffic at the edge of the networks. Computation offloading services in Multi-access offloading execution (MEC) enabled networks to offer potentials of a better Quality of Service (QoS) than traditional networks. They are expected to reduce the propagation delay and enhance the computational capability for delay-sensitive tasks especially. Nevertheless, the distributed computing resources of edge devices urgently need reasonable resource controllers to ensure such distributed computing resources to be effectively scheduled. The benefits of Software-Defined Networking (SDN) may be explored to demonstrate their full potential through MEC services to reduce the response time of programs. In this paper, a new SDN-based MEC computation offloading service architecture is proposed to increase the coordination and offloading capabilities at the control plane. Besides, to deal with dynamic network changes and increase the exploration degree, we propose a novel Entropy-based Reinforcement Learning algorithm for delay-sensitive tasks computation offloading at the edge of the networks. Finally, the evaluation findings indicate that our proposed model has the potential to improve the network resource allocation and balanced performance significantly.

Keywords: Multi-access offloading execution, Computation offloading, Reinforcement learning, Entropy, Software-defined networking, Markov decision process

1. Introduction

In recent years, the volume and complexity of data in physical space have grown significantly with the rapid development of the Internet of Things (IoT) devices [1], and other intelligent systems [2]. Meanwhile, various intensive and innovative computer applications, such as smart healthcare, Virtual Reality (VR), and Augmented Reality (AR), are installed on devices and require various real-time services [3]. The physical space is filled with complex data, large-scale information, and intelligent applications, which are hardly used in traditional networks [4].

Multi-access offloading execution (MEC) architectures are regarded as the most promising architecture to address the inefficiencies of processing latency, dynamic connectivity, and network management in highly task-intensive systems and have received extensive attention in recent years [5]. MEC ensures efficient coordination among distributed resources by deploying cloud-like resources and related internet services at the network edge. Network management inevitably brings new challenges. Therefore, to effectively integrate the distributed resources of the edge network and realize flexible networking, it is necessary to introduce a control mechanism to arrange the distributed environment into the network. The advantages of Software-Defined Networking (SDN) meet these requirements, and many investigations on SDN apply to complementary techniques for MEC operations [6, 7]. The SDN-based MEC architecture can significantly improve the computing power of IoT devices. When IoT devices request computing resources from edge servers, the SDN controller collects the complete network information, such as network states, task states, and IoT device states, then makes a computation offloading decision by coordinating the collected distributed resources from a global perspective, thereby reducing the execution delay and energy consumption of the tasks of these IoT devices [8].

However, SDN-based MEC computation offloading still faces challenges related to the limited computing resources and capabilities of IoT deceives. For example, many intelligent applications deployed on IoT devices, such as AR and VR, are response-sensitive and require complex computing and real-time analysis [9, 10]. While some research has been carried out on SDN-based MEC computation offloading, most of them optimized by minimizing latency or energy consumption, and few investigations have been conducted on both latency-constrained and energy-constrained tasks. On the one hand, some delay-sensitive tasks, such as smart health and Auto drive, are required to be completed within a limited time to guarantee the Quality of Service (QoS) [11]. On the other hand, for battery-

* Corresponding author. Tel.: +86 13022429092
E-mail address: wangxw@mail.neu.edu.cn (Xingwei Wang).

powered IoT devices such as wearable and mobile phones. Once these devices are in low power mode, they may automatically shut down or shut down the communication module [12]. In this case, the remaining applications will not be completed, thus requiring tasks to be completed within the energy budget. Moreover, to reduce the execution delay of tasks and prolong the battery life of edge devices, applications can be appropriately transferred to edge servers with sufficient computing resources and battery power resources for execution. Thus, we aim to minimize the sum of processing delay and focus on delay-sensitive computational tasks with processing delay toleration and energy consumption budget combining SDN and MEC system.

In order to solve those complex computational offloading problems, the task is constructed as Markov Decision Process (MDP) model. As the large offloading action dimensions make computing offloading more complicated, we propose an Entropy-based Deep Reinforcement Learning (E-DRL) method to improve the exploration ability of the algorithm and provide more execution schemes. The main contributions of this paper are as follows:

- We present an SDN-based MEC offloading architecture that facilitates the cooperation between the user equipment and edge node and further reduces the task processing latency in a centralized manner.
- We formulate the task computation offloading problem as a delay minimization problem by considering task delay toleration and energy consumption budget. We transform it into an optimization problem based on the Markov Decision Process (MDP) model to make it solvable.
- To solve the defined optimization problem, we develop an entropy-based reinforcement learning algorithm to learn the optimal offloading policy for all user equipment. It can explore more action possibilities and enhance its exploration ability by introducing a concept of entropy-based strategy in the reward.
- Case studies demonstrate that the proposed method exhibits a more favorable computational performance than benchmark RL methods in reducing both processing delay and energy consumption.

The remainder of this work is structured as follows. Section 2 focuses on related work. Next, Section 3 describes the SDN-based system model and expands on the problem statement. Sections 4 and 5 address the optimization problem and the proposed algorithm based on the MDP model. Section 6 assesses the proposed scheme's performance. Finally, Section 7 summarizes this paper.

2. Related Work

With the increase of IoT devices and applications, the amount of data and types of applications have also increased. Therefore, a control mechanism is necessary to arrange the distributed environment into the network. Inspired by SDN, the SDN-based MEC computation offloading architecture, which integrates SDN and offloading execution, has emerged to manage the data resource centrally through introducing SDN centralized control and orchestration features [13]. Mensah et al. [14] formulated the task offloading and associated resource allocation problem as a game with mixed strategies based on SDN controller. It optimized the number of devices in the vehicular network, supporting communication and computational limitation. Hu et al. [15] proposed an SDN-based method and principle from the perspective of physical-information mapping. Its ultimate goal was to achieve a highly automated and intelligent MEC system. Misra et al. [16] consider the problem of task offloading in a dynamic network, where IoT devices are connected to fog computing nodes by multi-hop IoT access points by SDN controller from a global view. However, the above work is only applicable to limited scenarios and cannot be transferred to other systems. Therefore, based on the above research, a two-layer distributed SDN-controlled MEC network architecture is proposed to manage MEC computation offloading.

In the study of MEC computation offloading, execution delay and energy consumption are considered as important indicators of the feasibility of a method [9, 17]. Satake et al. [18] proposed a dynamic task offloading method adapting to changes in the network resources framework and solved the optimization problem by minimizing task execution time. Consequently, some works consider energy consumption a significant constraint of the research. Yan et al. [19] expected Wireless Power Transfer (WPT) to apply to MEC to prolong the operation time of the battery in the wireless network system. Huang et al. [20] proposed a distributed computation offloading algorithm considering both the energy consumption and delays. Moreover, Zhao et al. [21] added the energy consumption constraints to the optimization objectives for relay-assisted Non-Orthogonal Multiple Access (NOMA)-MEC network with WPT. However, the preceding work frequently ignores the top limit of the energy consumption of the equipment in the actual scenario. Based on the above research, we aim to minimize the sum of processing delay and focus on delay-sensitive computational tasks with processing delay toleration and energy consumption budget in this research.

With the rapid development of deep reinforcement learning (DRL) methods in computational offloading and the introduction of the concept of resource allocation, we are inspired by the management work that handles large state and action spaces [22]. For instance, Huang et al. [19] proposed a Deep Reinforcement Online Learning-based (DROL) offloading framework that learns the binary offloading decisions from experience. It eliminates the need to solve combinatorial optimization problems and reduces computational complexity considerably, but it is weak in large-scale networks. Chen et al. [23] modeled the optimal computational offloading policy as an MDP and developed a Double Deep Reinforcement Learning (DDRL) method to derive the optimal offloading strategy by maximizing the long-term performance of the utility. Yan et al. [24] proposed a new DRL-based offloading framework to address the challenges of task dependency and adaptation to dynamic scenarios. Although the suggested DRL method allows for the

automated discovery of models shared by several applications and the derivation of the appropriate offloading strategy in various circumstances, the challenge of offloading optimization in SDN-based MEC remains an open problem.

In this paper, a delay toleration and energy consumption budget MEC offloading model based on SDN is proposed combined with future network requirements. This model uses virtualization technology to abstract resources and migrate computing to the edge network. In addition, we proposed a novel entropy-based reinforcement learning algorithm for delay toleration computation offloading on this SDN-based MEC computation offloading architecture.

3. System Description and Problem Formulation

This section will introduce the system in detail, dealing with the resiliency and scalability problems. It consists of the architecture of MEC based on SDN, network model, communication model, and problem formulation.

3.1. The architecture of MEC based on SDN

The SDN-based framework of this computational task offloading for MEC is shown in Fig. 1. It includes two planes, i.e., the data plane and the control plane. The data plane consists of a set of User Equipments (UEs), which consists of devices with computational tasks (i.e., smartphones, wearable devices, and other IoT devices) and a set of Edge Nodes (ENs). The EN corresponds to small cell Base Stations (BS), and each BS deploys an edge server for computation tasks delivering from the UE. The control plane is realized by an SDN controller deployed at control BS, we form the learning model on SDN controller to better train and execute the designed algorithm and apply it to the centralized control model. Therefore, we may consider a control BS as an agent that seeks to take the optimal decision with regard to its requests. The UEs are connected to the control BS or small cell BS through a wireless link, while the small cell BSs are connected to the control BS via a high-speed front-haul network.

The main idea of SDN is to decouple the control plane from the data plane through visualization. Through the air interface separation technology [25], the controller can be separated from the data provided by the edge server. Specifically, the SDN controller can perform global control over the whole network. Combing the overall perspective of the network states (i.e., offloading task characteristics, remaining resource states of EN, and network states), the controller allocates computation and communication resources for the tasks. Based on these resources, the controller makes offloading decisions: offload to EN or not, and sends the offloading strategies to each UE for execution.

We proposed that the SDN central controller involves two functions: Task Monitoring Function and Edge Cloudlet Function. Task Monitoring Function means that the SDN central controller is responsible for collecting the global information (i.e., task sizes, energy consumption, computing speed). It is advised to formulate the optimal offloading decisions for further processing. Edge Cloudlet Function is responsible for collecting the information of edge servers (i.e., the compute and storage capacity of edge servers, how much the server is loaded).

Table 1: Table of Notations

Parameter	Description
M	Number of UE
N	Number of EN
δ_m	Delay toleration of task x_m
c_m	Computation amount of task x_m
s_m	Amount of data contents of the task x_m
χ_m	Indicator of the offloading scheme of task x_m
e_m	Energy consumption budget of task
f_m	Computing capacity of the UE m
f_n	Computing capacity of the EN n
ω	Channel bandwidth
$\varrho_{m,n}$	Transmitting power between UE m and EN n
$h_{m,n}$	Channel gain between UE m and EN n
L	Path loss at a unit distance
$d_{m,n}$	Distance between UE m and EN n
ϖ^2	Gaussian noise ratio
ϵ_m	The coefficient of energy consumption
$r_{m,n}$	Transmission rate between UE m and EN n

Within the framework we propose, the task offloading processing is as follows. The tasks that arrive at UE follow a Poisson distribution with arrival rate ς [26]. The SDN controller collects all computational information, i.e., offloading task characteristics, EN remaining resource status, and network status. Moreover, based on the received computational information, the SDN controller gives the task computation offloading strategy (i.e., execution location of the task) for UE executing the task according to the task's processing delay and energy consumption. The UE executes the corresponding action according to the computation offloading strategy. In the case that the task is offloaded to the EN, the computation result of the task is returned to the UE after the execution is completed.

3.2. Network Model

We consider a multi-user scenario MEC system with a set of UEs $M = \{1, 2, ...m\}$ who has computational tasks to be executed locally or offload to EN, denoted as a set $N = \{1, 2, ...n\}$. Each UE is assumed to have a finite computation capability and connects to EN for task offloading.

We assume that each EN deploys an edge server and has a higher computation capacity than UEs. In this MEC system, time is assumed as slotted, and we denote the time slot index set by $T \in \{1, 2, ...t\}$. Aiming to enhance the QoS of users in this system, we take the delay toleration δ_m and energy consumption budget e_m of the task into consideration. We adopt a widely used task model to describe task x_m for the task of UE m, i.e., $x_m = \{s_m, c_m, f_m, \delta_m, e_m\}$, where s_m stands for the amount of data sizes of task x_m (i.e., data input and associated code), c_m stands for the computation amount of the task x_m (i.e., the CPU cycles needed of the task x_m), f_m stands the CPU computing capability of the UE m, and δ_m is the toleration delay of the task

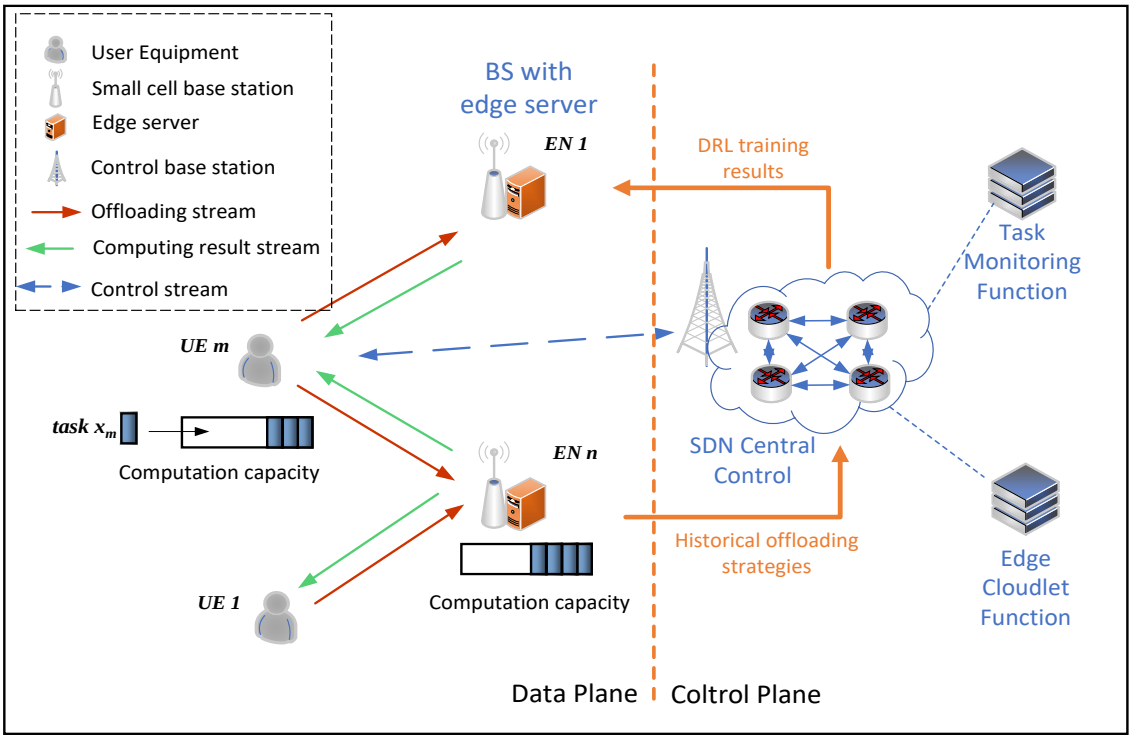

Figure 1: The System Framework of SDN-based MEC computation offloading.

x_m. The computational tasks x_m arrive dynamically to UEs with task arrival rate ς_m, and these tasks can be computed locally or be offloaded to EN to reduce the process delay and energy consumption. The SDN controller collects the information of the computational tasks deploying on the UEs (i.e., data sizes, delay toleration) and gives the offloading strategy based on this information. Then, these tasks can be offloaded to EN through a wireless channel or processed locally based on offloading decisions. This system focuses on Local computing and offloading the computational task to EN without further offloading to the central cloud or other UEs. The notations used in the definition and formulation are shown in Table 1.

3.3. Communication Model

Next, we introduce the communication model of this system. We consider the transmission delay and energy consumption when UE m offloads their tasks x_m on to EN n. In terms of the Shannon Theorem [27], the transmission rate $r_{m,n}$ from UE m to EN n is equal to the data offloading capacity:

$$r_{m,n} = \omega \log_2(1 + \frac{\varrho_{m,n} G_r}{L d_{m,n} + \varpi^2}) \tag{1}$$

where ω is channel bandwidth and $\varrho_{m,n}$ is the transmitting power of UE m, G_r is the antenna gain at EN, L is the path loss at a unit distance and $d_{m,n}$ is the distance between UE m and EN n. ϖ^2 is the power of additive white Gaussian noise. Consequently, the transmission delay $D^{tr}_{m,n}$ of the computation task x_m from UE m to EN n can be expressed as:

$$D^{tr}_{m,n} = \frac{s_m}{r_{m,n}} \tag{2}$$

Further, we can obtain the transmission energy consumption for UE m offloads its task x_m to EN n is as follows:

$$E^{tr}_{m,n} = \varrho_{m,n} \cdot D^{tr}_{m,n} = \varrho_{m,n} \frac{s_m}{r_{m,n}} \tag{3}$$

3.4. Task Execution Model

Considering the local execution process, we set f_m as the CPU computing capability of the UE m. Thus, the computation delay for local execution D^l_m of task x_m can be expressed as:

$$D^l_m = \frac{c_m}{f_m} \tag{4}$$

From another perspective, the energy consumption for local execution E^l_m is given by:

$$E^l_m = \epsilon_m c_m \tag{5}$$

where ϵ_m is the coefficient of energy consumption per CPU cycle of UE m.

Moreover, we consider the offloading execution process of this system. If the computational task x_m is offloaded to EN n, the execution delay $D^{ex}_{m,n}$ for offloading execution of task x_m on EN n can be given as:

$$D^{ex}_{m,n} = \frac{c_m}{f_n} \tag{6}$$

where f_n represents the CPU clock speed of the server at EN n, thus the offloading execution delay D_m^e equals the execution delay adding the transmission delay which can be expressed as:

$$D_m^e = D_{m,n}^{tr} + D_{m,n}^{ex} \tag{7}$$

In this system, the data size of the task results is generally smaller than before (i.e., the data size of the task to upload is 1000kb, while the data size of the task offloading strategy to download is 10kb at most). Therefore, we neglect the time spent on sending the task results back to the UE, similar to many studies such as [16, 28].

3.5. Problem Formulation

To improve system performance and better support for the advanced services of smart UEs, we coordinate the UE, EN and SDN controller to find the proper centralize offloading policy, which allows SDN controller to make offloading strategies by minimizing the average delay of each task.

We defined a integer decision variable $\chi_m \in \{0, 1\}$ for each UE's task, which indicated the task x_m is processed at local (i.e., $\chi_m = 1$) or offloaded to EN n for executing (i.e., $\chi_m = 0$). Formally, the computational task offloading problem can be formulated as follows:

$$\mathbf{P1}: \text{minimize} \sum_{m=1}^{M} \chi_m D_m^l + (1-\chi_m) D_m^e \tag{8a}$$

$$\text{s.t. } C1: \chi_m \in \{0,1\}, \forall m \in M \tag{8b}$$

$$C2: (1-\chi_m) E_{m,n}^{tr} + \chi_m E_m^l \le e_m, \forall m \in M, \forall n \in N \tag{8c}$$

$$C3: D_m^l, D_m^e \le \delta_m, \forall m \in M \tag{8d}$$

The objective function (8a) minimizes the total system delay in this computation offloading process. The first constraint (8b) guarantees the task is completed successfully, (8c) guarantees that the sum of energy consumption requirements of tasks x_m cannot exceed it's energy consumption budget e_m. The last condition (8d) indicated the total delay of task x_m should not exceed its tolerance delay δ_m. It is rather challenging to achieve the above objective since the optimization problem in (8) is NP-Hard. Next, proof of NP-Hardness for this problem is presented.

Theorem 1. *The optimization problem is NP-Hard.*

Proof. Consider there are M tasks $x_1, x_2, ..., x_m$ with the offloading energy budget E_{budget}, which means that with different offloading schemes for each time slot t, the energy consumed by the UE cannot exceed the energy budget for each task. We use D_m to denote total system latency of task x_m, and E_m to denote energy consumed for task x_m by UE m. Then, our optimization problem can be formulated as

$$minimize \frac{\sum_{m=1}^{m=M} D_m}{M} \tag{9a}$$

$$s.t. \sum_{m \in M} E_m \le E_{budget}. \tag{9b}$$

Then, this instance of our problem corresponds to the Knapsack problem [29, 30] when we define

$$\frac{D_m}{M} = -p_m \tag{10}$$

and the optimization problem can be formulated as:

$$maximize \sum_{m \in M} p_m \tag{11a}$$

$$s.t. \sum_{m \in M} E_m \le E_{budget}. \tag{11b}$$

Thus, this special case of workflow scheduling with energy budget constraint is NP-hard, and the optimization problem of this computation offloading scheme is also NP-Hard. This completes the proof.

As **Theorem 1.** proved, the problem in (8) is NP-Hard, and the computation offloading decision is memoryless with a sequential decision-making process. Therefore, we formulate the task computation offloading as an MDP minimization problem.

4. MDP Optimization Problem Formulation

In this section, we model the defined system delay minimization problem as an MDP problem, which can be represented by a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$, where $\mathcal{S}$ is the global state space, representing the object of next state. $\mathcal{A}$ is the action space, representing the object of the next action. $\mathcal{P}$ is the set of the state-to-state transition probabilities, and $\mathcal{R}$ is the reward function with determined state-action pair.

At the beginning of each time slot t, the SDN controller observes the global states $\mathcal{S}$ (e.g., task information, energy consumption budget of all task). If a new task arrives and needs to be processed, the SDN controller gives an offloading action $\mathcal{A}$ based on the global states $\mathcal{S}$. Then, the state and action will result in a reward $\mathcal{R}$. To balance the offloading strategies among UEs, we consider that the SDN controller broadcasts the processing strategies for all UE in each time slot. Based on all received strategies, UE can always make proper offloading action and choose optimal EN, depending on their own energy consumption and delay which are related to the reward $\mathcal{R}$. The control messages are exchanged periodically. In the following, we provide a detailed introduction to the formation of elements in the considered MDP.

State: The state of an MDP can be represented by $\mathcal{S} = \{s_t = \{(x_m(t), F(t), E_{budget}(t))\}\}$, where $s_t = \{s_1, \cdots s_m\}$ represents the local state for all UE at time slot t, $x_m(t)$ represents the feature of the computation task of UE m at time slot t, $F(t) = \{f_1, ...f_n\}$ represents the CPU computing capability of the set of EN at time t, and $E_{budget}(t) = \{e_m\}$ denotes the energy consumption budget of this computation offloading process at time t.

Action: The action of the agent can be defined as $\mathcal{A} = \{a_t = \{\chi_m(t), \xi_m(t)\}\}$, where the $\chi_m(t) \in \{0, 1\}$ indicates the task x_m

be executed locally or not at time t, $\chi_m(t) = 0$ represents that the task x_m will be allocated to EN; otherwise, the task will be computed locally and $x_m(t) = 1$. $\xi_m(t) \in \{1, 2...n\}$ indicates the EN $\xi_m(t)$ is the place that for task x_m executing at time t, i.e., while task x_m will be offloaded to EN n, then $n = \xi_m(t)$ at this time.

State-to-state transition: The state-to-state transition probability distribution is denoted as $P = \{P_{a_t}(s, s_{t+1})|s, s_{t+1} \in \mathcal{S}, a_t \in \mathcal{A}, P \in \mathcal{P}\}$. State s_t can be transferred into s_{t+1} by taking action a_t based on probability $P_{a_t}(s_t, s_{t+1})$.

Reward function: In Reinforcement learning algorithm, the environment gives the agent feedback in a particular state-action pair at discrete time t, which can be expressed as $\mathcal{S} \times \mathcal{A} \rightarrow \mathcal{R}$. In order to derive the reward function of this model, we find an immediate utility at time t to quantify the task computation reward for the UEs. Therefore, in this system, the immediate reward of service is defined as $\mathcal{R} = \{r_t(s_t, a_t)\}$ which can be expressed as follows:

$$r(s_t, a_t) = \text{minimize} \sum_{m=1}^{M} \chi_m D_m^l + (1 - \chi_m) D_m^e \tag{12}$$

We need to find a policy π that can minimize the system delay from a long-term perspective. Every state-action pair has a value function $Q(s_t, a_t)$ by introducing the action in MDP, i.e., the expected discounted reward when starting in state s_t and selecting an action a_t. The state-action value is defined as:

$$Q(s_t, a_t) = \mathbb{E}_\pi[r_t|\mathcal{S} = s_t, \mathcal{A} = a_t] \tag{13}$$

We now consider off-policy learning of action-values $Q(s_t, a_t)$. The objective is to search for a policy strategy $\pi(a_{t+1}|s_{t+1})$ to maximize the reward of the agent. The update function of $Q_\pi(s_t, a_t)$ is formulated as:

$$\begin{aligned} Q_\pi(s_t, a_t)^* =& r(s_t, a_t) + \sum_{s_{t+1} \in S} P_{a_t}(s_t, s_{t+1}) \\ & \sum \pi(a_{t+1}|s_{t+1})(Q_\pi(s_{t+1}, a_{t+1})) \end{aligned} \tag{14}$$

where $P_{a_t}(s_t, s_{t+1})$ is defined as state-to-state transition probability. Based on the above, we can get the task offloading strategy π_m^* for each UEs such that its long-term discount reward is maximize, i.e.,

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

where γ is the discount parameter associated with the importance level of the reward, in this system, we pay more attention to the current reward. The expectation $\mathbb{E}[\cdot]$ is with respect to the system parameters of computational tasks of all UEs, i.e., task data sizes, computational requirements.

5. Problem solution with Maximum Entropy Reinforcement Learning

In this section, we present an SDN-based MEC computation offloading algorithm named Entropy-based Deep Reinforcement Learning (E-DRL) to solve the above optimization problem. Specifically, the proposed algorithm has a higher exploration ability by introducing the concept of entropy so that the agent can explore as many actions as possible and prevent falling into the situation of the local optimum. Particularly, the E-DRL algorithm is introduced in Section 5.1, and the details of the training process are introduced in Section 5.2.

Algorithm 1: An Maximum Entropy Reinforcement Learning to solve the offloading decision problem

Initialize θ, $\bar{\theta}$, φ and ϕ, the state s_t, experience replay memory D, batch size I of sample batch
for *each iteration* **do**
  for *each environment step* **do**
    Choose action $a_t \sim \pi_\phi(a_t|s_t)$
    Choose next state s_{t+1} and reward $r(s_t, a_t)$
    Store (s_t, a_t, r_t, s_{t+1}) in the experience replay memory D
    Update the sample batch D'
  end
  for *i=1 to I* **do**
    Calculate $Q_\theta(s_t, a_t)$ based on (24)
    Calculate $L_Q(\theta) = \mathbb{E}[\frac{1}{2}(Q_\theta(s_t, a_t) - Q_{\bar{\theta}}(s_t, a_t))^2]$ based on (25)
    Update $\pi_\phi(s_t, a_t)^*$ based on (29) and (30)
    for *each gradient step* **do**
      $\theta \leftarrow \theta - \lambda_\theta \hat{\nabla}_\theta L(\theta)$
      $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi L(\phi)$
      $\varphi \leftarrow \lambda_\varphi \hat{\nabla}_\varphi L(\varphi)$
      $\bar{\theta} \leftarrow \iota\theta + (1 - \iota)\bar{\theta}, \iota \in [0, 1]$
    end
  end
end

5.1. E-DRL Algorithm

In this paper, we adopt a maximum entropy-based DRL algorithm to make offloading strategies in this SDN-based MEC computation offloading system. The details of the algorithm are shown in Algorithm 1.

Entropy measures the randomness or uncertainty of a random phenomenon, and the magnitude of entropy reflects the degree of exploration which applies to various areas such as information theory, finance, statistics, cryptography, physics, and artificial intelligence [31]. To encourage the exploration of computation offloading policy, we set R^e as the relevant quantity of the task corresponding to the entropy reward, that is, to increase the entropy determined by the action distribution based on the original reward:

$$R_{t+1}^e = R_{t+1} + \mathfrak{e} H(\pi(\cdot|s_t)), t = 0, 1, ... \tag{16}$$

where $\mathfrak{e} > 0$ is the temperature parameter that controls the stochastic level of the optimal policy, $H(\pi(\cdot|S_t) = -log\pi(\cdot|s_t)$ represents the entropy of the strategy π in the state s_t. Introducing $-log\pi(\cdot|s_t)$ as entropy in reward can encourage exploration

since maximizing entropy will make the output of the policy more uniform in action and learn more near-optimal behavior. Specifically, some states can map to more than one optimal action, assigning equal probabilities to actions with similar Q values. Moreover, any action will not be given a very high probability for these optimal actions, avoiding the repeated selection of the same action and falling into sub-optimal can improve learning speed.

Then, the entropy long-term reward G_t^e, value function $V_\pi^e(s_t)$ and Q function $Q_\pi^e(s_t, a_t)$ are recorded as:

$$G_t^e = \sum_{t=0}^{+\infty} \gamma^t R_{t+1}^e, t = 0, 1, 2, ... \tag{17}$$

$$V_\pi^e(s_t) = \mathbb{E}_\pi[G_t^e|s_t], s_t \in \mathcal{S} \tag{18}$$

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

Moreover, in order to solve the high-dimensional decision problem, we approximately calculate the state value function $V_\varphi(s_t)$, Q function $Q_\theta(s_t, a_t)$, and strategy function $\pi_\phi(s_t|a_t)$ same as other DRL algorithm [32]. When we give a certain state-action pair (s_t, a_t), the state s_t to the action a_t is completely determined, based on (17)-(19), the state value function $V_\varphi(s_t)$, can be rewritten as follows:

$$\begin{aligned} V_\varphi(s_t) &= \mathbb{E}_{\mathcal{A}\sim\pi}[Q_\pi^e(s_t, \mathcal{A})] + \mathfrak{e}H(\pi(\cdot|\mathcal{S})), \\ &= \mathbb{E}_{\mathcal{A}\sim\pi}[Q_\pi^e(s_t, \mathcal{A})] - \mathfrak{e}log\pi(\mathcal{A}|s_t), s_t \in \mathcal{S} \end{aligned} \tag{20}$$

Furthermore, according to the Bellman backup equation, the entropy-based Bellman backup equation can be calculated via adding the entropy into the reward, which can be expressed as:

$$\begin{aligned} Q_\theta(s_t, a_t) &= Q_\pi^e(s_t, a_t) + \mathfrak{e}H(\pi(\cdot|\mathcal{S})) \\ &= r(s_t, a_t) + \gamma\mathfrak{e}\mathbb{E}_{\mathcal{A}\sim\pi}H(\pi(\cdot|s_{t+1})) + \gamma\mathbb{E}_{\mathcal{A}\sim\pi}[Q_\theta(s_{t+1}, a_{t+1})] \\ &= r(s_t, a_t) + \mathbb{E}_{s_{t+1},a_{t+1}}[Q_\theta(s_{t+1}, a_{t+1}) - \mathfrak{e}log(\pi(a_{t+1}|s_{t+1}))] \end{aligned} \tag{21}$$

Above all, the policy $\pi_\phi(s_t|a_t)$ based on the equation (15) can be rewritten as:

$$\pi_\phi(s_t|a_t) = \arg\max_\pi \mathop{\mathbb{E}}_{(s_t,a_t)\sim\rho(\pi)}[\sum_{t=0}^{\infty} \gamma^t(r(s_t, a_t) - \mathfrak{e}log\pi(\cdot|s_t))] \tag{22}$$

where $(s_t, a_t) \sim \rho(\pi)$ represents the trajectory distribution probability under the policy π. Therefore, under the optimal policy $\pi_\phi(s_t|a_t)$, the Bellman optimally equation for the Q function $Q_\theta(s_t, a_t)$ related to the value function $V_\varphi(s_t)$ can be written as:

$$\begin{aligned} Q_\theta(s_t, a_t)) &= \mathop{\mathbb{E}}_{a_{t+1}\sim\pi}[r(s_t, a_t) + \gamma(Q_\pi(s_{t+1}, a_{t+1}) + \mathfrak{e}H(\pi(\cdot|s_{t+1})))] \\ &= \mathop{\mathbb{E}}_{a_{t+1}\sim\pi}[r(s_t, a_t) + \gamma(V_\varphi(s_{t+1}))] \end{aligned} \tag{23}$$

By utilizing a more general maximum entropy objective to make stochastic offloading policies, the agent will stochastically choose a set of offloading actions which guarantees every useful set of offloading actions will not be ignored.

5.2. Training Process

Fig. 2 shows the structure of the Entropy-based deep reinforcement learning Algorithm. On the one hand, the Entropy-based DRL Algorithm introduces a parameter-containing function to approximate the value function and then uses the approximate value of this value function to estimate the return value. This process is called **Learning Q Function**. On the other hand, it uses the estimated return value to estimate the policy gradient and then updates the policy parameters. This process is called **Learning the Policy**.

1) Learning Q Function Similar to the DQN algorithm [32], the update method of the Q function $Q_\theta(s_t, a_t)$ is fitted by Cross Entropy loss between Q function $Q_\theta(s_t, a_t)$ and Q target function $Q_{\bar{\theta}}(s_t, a_t)$ and it updates Bellman residuals. Therefore, the objective function of can be written as follows:

$$L_Q(\theta) = \mathbb{E}[\frac{1}{2}(Q_\theta(s_t, a_t) - Q_{\bar{\theta}}(s_t, a_t))^2] \tag{24}$$

where $Q_{\bar{\theta}}(s_t, a_t)$ satisfies $Q_{\bar{\theta}}(s_t, a_t) = r(s_t, a_t) + \gamma(Q_{\bar{\theta}}(s_{t+1}, a_{t+1}) - \mathfrak{e}log(\pi_\phi(a_{t+1}|s_{t+1})))$, and $\bar{\theta}$ represents the parameter of the target Q function. The parameter θ be updated by calculating the loss function by the different between Q function $Q_\theta(s_t, a_t)$ and the target Q function $Q_{\bar{\theta}}(s_t, a_t)$, which improves the convergence of training and the stability of the algorithm. Furthermore, in order to get the closest optimization of $Q_\theta(s_t, a_t)$, the most common method is to update θ through gradient descent to minimize $L(\theta)$:

$$\begin{aligned} \hat{\nabla}_\theta L(\theta) = &- \nabla_\theta Q_\theta(s_t, a_t)(r(s_t, a_t) + \\ &\gamma(Q_{\bar{\theta}}(s_{t+1}, a_{t+1}) - \mathfrak{e}\log(\pi_\phi(a_{t+1}|s_{t+1})))) \end{aligned} \tag{25}$$

Finally, the Q function and the target network parameters can be updated as

$$\begin{aligned} &\theta \leftarrow \theta - \lambda_\theta \hat{\nabla}_\theta L(\theta) \\ &\bar{\theta} \leftarrow \iota\theta + (1 - \iota)\bar{\theta}, \iota \in [0, 1] \end{aligned} \tag{26}$$

Similar to the strategy of Q function $Q_\theta(s_t, a_t)$, value function $V_\varphi(s_t)$ is also approximated by a neural network, and the parameter φ is updated alternately by stochastic gradient descent, where the strategy function follows a Gaussian distribution. The objective function of value function $V_\varphi(s_t)$ is as follow:

$$L(\varphi) = \mathbb{E}_{s_t\sim D}[(\frac{1}{2}(V_\varphi(s_t) - \mathbb{E}_{a_t\sim\pi}[Q_\theta(s_t, a_t) - log\pi_\phi(a_t|s_t)])^2] \tag{27}$$

where D is the experience replay memory of the past experience, which records the states, actions and rewards at the current moment and states at the next moment of the environment and improves the utilization of the state–action transitions [33]. Therefore, the gradient descent to minimize φ is written as:

$$\hat{\nabla}_\varphi L(\varphi) = \nabla_\varphi V_\varphi(s_t)(V_\varphi(s_t) - Q_\theta(s_t, a_t) + \log\pi_\phi(a_t|s_t)) \tag{28}$$

Finally, the value function network parameters can be updated as

$$\varphi \leftarrow \lambda_\varphi \hat{\nabla}_\varphi L(\varphi) \tag{29}$$

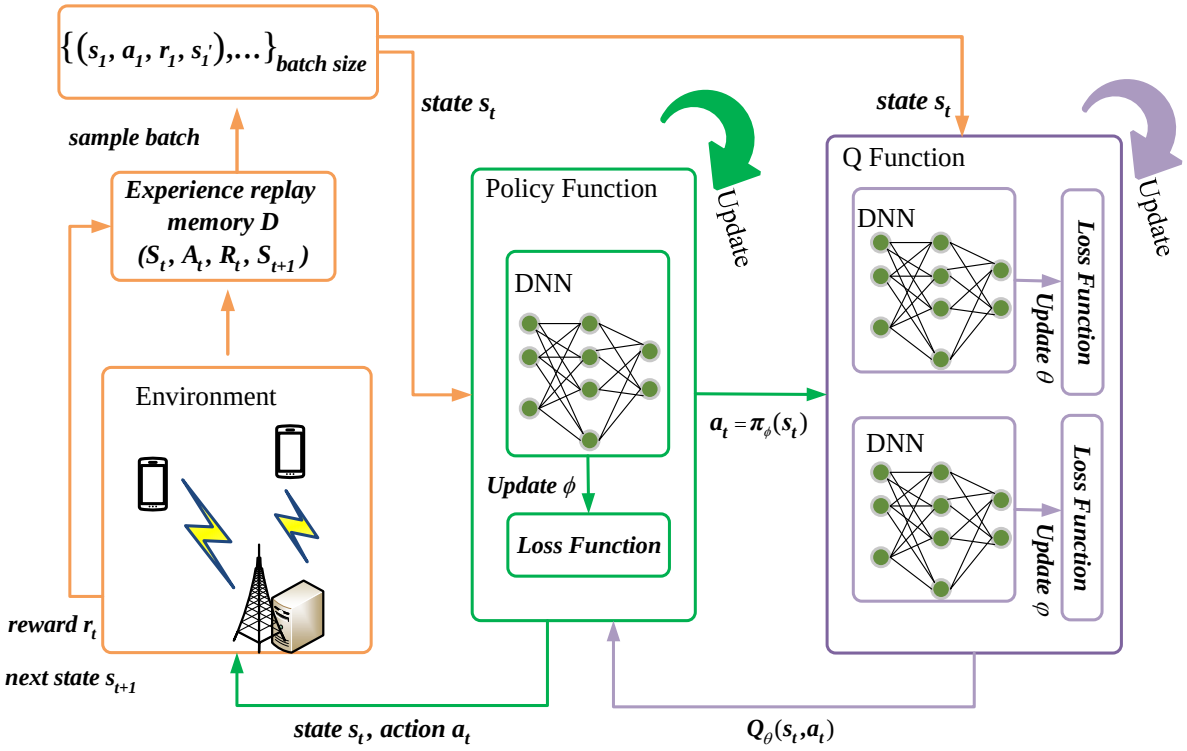

Figure 2: The structure of E-DRL algorithm

2) Learning the Policy The policy strategy can be update though Q function $Q_\theta(s_t, a_t)$ based on (22), (23). In order to avoid getting caught in local optimization and keep the action probability distribution of the strategy can confirm to the distribution of Q function $Q_\theta(s_t, a_t)$ in the state s_t at time t, we set the action $a_t = f_\phi(\tau_t; s_t)$ and update the policy strategy parameters with Kullback-Leibler (KL) divergence:

$$\pi_\phi(s_t, a_t)^* = \arg\min D_{KL}(\pi_\phi(s_t|a_t))||\frac{exp(\frac{1}{\varsigma}Q_\theta(s_t, \cdot))}{Z^{\pi_\phi(s_t|a_t)}(s_t)}) \quad (30)$$

where $Z^{\pi_\phi(s_t|a_t)}(s_t)$ is the sum of Q function $Q_\theta(s_t, a_t)$ which is used for normalization in state s_t. The action $a_t = f_\phi(\tau_t; s_t) = f_\phi^\mu(s_t) + \tau_t \odot f_\phi^\sigma(s_t)$ in this step, the function $f(\cdot)$ is used to calculate the average and variance of the Gaussian distribution, τ_t is the noise of Gaussian sampling.

Therefore, combining with equation (29), the policy strategy with KL divergence as the expectation terms can be written as:

$$L(\phi) = \mathop{\mathbb{E}}_{s_t, \tau_t \sim \pi}[\log \pi_\phi(f_\phi(\tau_t; s_t|s_t) - Q_\theta(s_t, f_\phi(\tau_t; s_t))] \quad (31)$$

Consequence, the approximate gradient formula based on action sampling can be written as:

$$\begin{aligned}\hat{\nabla}_\phi L(\phi) =& \nabla_\phi \log(\pi_\phi(a_t|s_t)) + (\nabla_{a_t} \log(\pi_\phi(a_t|s_t)) \\ &- \nabla_{a_t} Q(s_t, a_t) \nabla_\phi f_\phi(\tau_t; s_t)\end{aligned} \quad (32)$$

Finally, the policy network parameters can be updated as

$$\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi L(\phi) \quad (33)$$

6. SIMULATION RESULTS AND DISCUSSION

In order to ensure that the strategies generated by DRL are efficient and usable, we set the parameters same as [34] as the initial starting point of each, and compare the performance of the system in this SDN-based MEC computation offloading environment. The main default simulation settings are listed in Table 2. Furthermore, we evaluate the presented intelligent offloading strategy by comparing it with Local computing, and offloading execution to demonstrate the necessity of computational offloading. Moreover, we compare the work with a similar problem statement, such as [34], based on Deep Deterministic Policy Gradient (DDPG) algorithm. The DDPG algorithm is a state-of-the-art off-policy algorithm that is widely used in MDP problems, i.e., Unmanned Aerial Vehicle [35], Robot grasping [36]. Finally, we evaluate the comparison of classic DRL and E-DRL in terms of the normalized reward with episodes to demonstrate the effectiveness of our method.

In the offloading execution experiment, we offload all tasks to EN randomly, and for Local computing, all tasks are executed locally. Both offloading execution and Local computing are widely used in academia, representing the general method

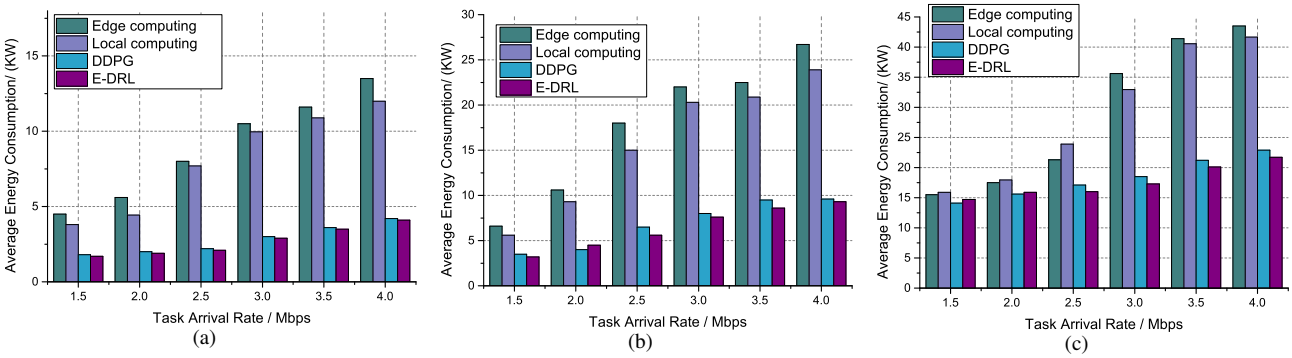

Figure 3: Performance under different task arrival rate vs average energy consumption/ (KW): (a)number of UE is 5; (b) number of UE is 50; (c)number of UE is 100.

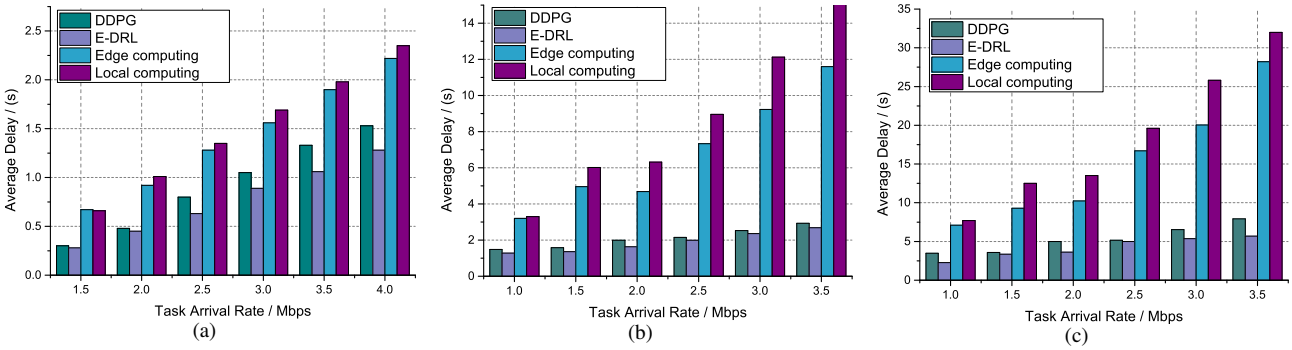

Figure 4: Performance under different task arrival rate vs average delay: (a)number of UE is 5; (b) number of UE is 50; (c)number of UE is 100.

Table 2: Default simulation parameters

Parameter	Description	value
m	Number of user equipment (UE)	[5,100]
n	Number of Edge node (EN)	10
ω	Bandwidth	6MHz
$\varrho_{m,n}$	Transmitting power	25dBm
G_r	antenna gain	-5dB
L	path loss	4
d_{mn}	Distance between UE m and EN n	35m
f_m	Computing capacity of UE m	10GHz
f_n	Computing capacity of EN n	50GHz
γ	discount parameter	0.5

of offloading. First, we evaluated the impact of task arrival rate on the average energy consumption under three different numbers of UE. Then the effect of the task arrival rate on the average delay under four different numbers of UE was observed. Finally, the task's average delay and energy consumption under three different algorithms was considered comprehensively, and the conclusion was finally reached.

In Fig. 3, we evaluate the performance under different tasks and algorithm settings in this system. Fig 3(a) shows that with the increment of task arrival rate from 1.5 Mbps to 4 Mbps when the number of UE is 5, the average energy consumption increases obviously with four methods. A high task arrival rate means a more significant computation requirement. In the experiment, when the task arrival rate is maintained at 1.5 Mbps, the average energy consumption of the four methods is mainly maintained at the same level. However, as the task arrival rate increases, the average energy consumption of DDPG and E-DRL display a definite growing trend because DDPG and E-DRL make offloading decisions are generated by learning the training data. Furthermore, DDPG uses a deterministic strategy, while E-DRL uses a random training strategy and more exploratory by introducing entropy at training process. Therefore, the E-DRL approach is better than DDPG in a dynamic MEC environment. For the same reason, when the number of tasks is increasing, the proposed algorithm also has excellent performance shown in Fig. 3(b) (the number of UE is 50) and Fig.3 (c) (the number of UE is 100). In addition, we observe that the average energy consumptions of offloading execution and Local computing are much more than the average energy consumptions of E-DRL and DDPG. This is mainly because DDPG and E-DRL introduce an experience replay process for obtaining an intelligent computation offloading decision.

In Fig. 4, we evaluate the performance under different methods and task arrival rate settings in this system. As shown in Fig. 4(a), the average delay of all the methods increases with the growing task arrival rate from 1.5 Mbps to 4.0 Mbps. When the task arrival rate is 1.5 Mbps, there is little computation requirement in the system, so the average delay is mostly main-

Table 3: The processing delay (s) with different data size

Method	2MB	3MB	4MB	5MB	6MB	7MB	8MB	9MB	10MB
E-DRL	**4.272**	**5.305**	**6.207**	**7.171**	**8.038**	**8.648**	**9.235**	**9.833**	**10.264**
DDPG	5.123	6.583	7.479	8.195	9.074	10.027	11.001	11.711	12.221

tained at the same level. As the task arrival rate increases from 1.5 Mbps to 4.0 Mbps, the average delay of our approach increases by 38.12%, while those of other benchmark methods increase by at least 52.20%. It implies that as the load of the system increase, the average delay of the proposed method increases less dramatically than those of the benchmark methods. Furthermore, as the number of UEs increases, the average delay of each benchmark increases due to the potentially increasing load at the edge server. Moreover, it shows in Fig. 4(b) and Fig. 4(c), as the number of UE, the average delay of four methods increases. It is because when the number of the UE increase, more computation tasks increase, and more computation requirements. Our method consistently maintains the lowest latency compared to other methods, with a slight increase in computational latency as the task arrival rate increases. It shows that our algorithm is more robust and can maintain excellent performance at peak traffic times.

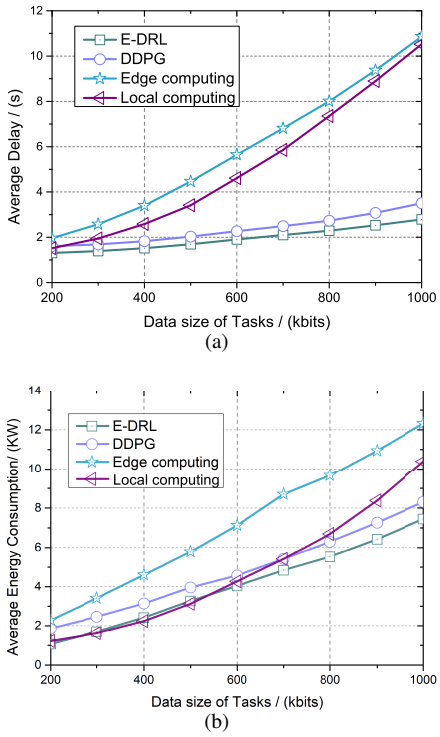

Figure 5: Performance under different data size of tasks: (a)average delay; (b) average energy consumption.

We can observe that the average delay of all the methods increases with the growing data size of tasks from 200 kbits to 1000 kbits in Fig. 5 (a). Our approach always achieves a lower average delay than benchmark methods, especially when the data size is large. offloading execution consumes time on transmission data to edge servers. Therefore, it is probably the most inefficient way to offload to the EN. In Fig.5 (b), as the data size of tasks increases, the average energy consumption of each method increases. Moreover, the rise of E-DRL is significantly lower than DDPG, Local computing, and offloading execution. The main reason is that E-DRL can effectively address the unknown load dynamics at the edge servers. When the data size of tasks increases to 1000 kbits, it achieves an average energy consumption of 4.2%-62.5% lower than those of DDPG, offloading execution, and Local computing.

Further, we study the impact of large-scale data size on the processing delay for the proposed algorithm in Table 3. The large-scale data size simulation assessment can ensure the robustness and strength of the proposed algorithm under heavy traffic load. We vary the data size from 2 MB to 10 MB. It is evident that the large-scale task data size prolongs the task processing delay in task offloading. In particular, our proposed method outperforms DDPG in terms of the processing delay of the generated task. Intuitively, the processing delay of our proposed method at 10MB is about 19.6%, which is a decrease compared to the DDPG-based method.

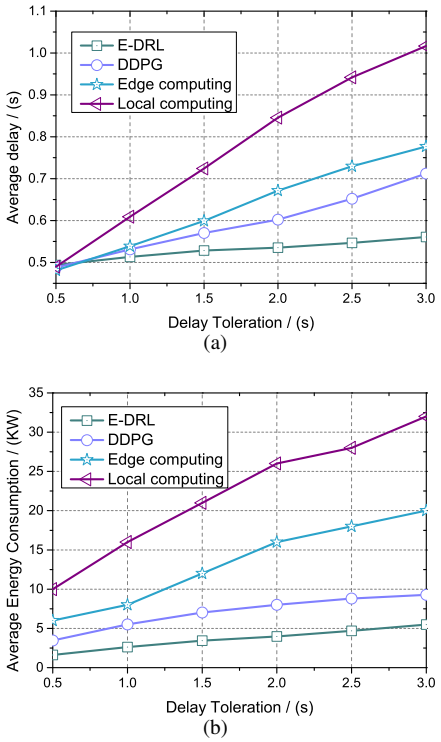

Figure 6: Performance under different delay toleration: (a)average delay; (b) average energy consumption.

At last, to explore the effect of delay toleration on task execu-

tion delay and energy consumption, we changed the task delay toleration from 0.5s to 3.0s. We conducted experiments on the performance of the four algorithms in terms of execution and energy consumption. As shown in Fig. 6(a), the proposed algorithm always achieves a low average delay than the benchmark methods, especially when the delay toleration is large. This is because when the delay toleration is large, the task has enough time to be executed and completed. As shown in Fig. 6(a), our approach has only a slight effect on the change of delay toleration. By contrast, Local computing and offloading execution have very obvious variations to the shift of tolerance. It is mainly because our approach could complete the task in a short time. In comparison, the average delay of the other methods is large than 0.9 seconds. In Fig. 6(b), as the delay toleration increases, the average energy consumption of each method increases. As the delay toleration increase from 0.5 seconds to 3.0 second, our average energy consumption increase by 14.28%, while those of the benchmark methods increase by at least 28.57%-62.82%. It implies that as the delay toleration of this method increases, the average energy consumption of the proposed approach increases less dramatically than those of the benchmark methods.

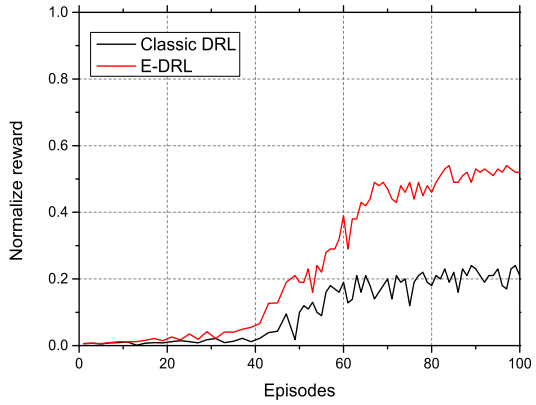

Figure 7: Comparison of classic DRL and E-DRL in terms of the normalize reward with episodes

As we mentioned, traditional methods cannot handle systems with large-scale action spaces. Furthermore, we evaluate the agent on each episode with 2000 training batches. From Figure 7, we can find that the classic DRL (i.e. DQN) learns a little after 100 episodes. It shows that an excessively large output layer may impair the performance of classical DRL. By contrast, our method achieves better performance, thus demonstrating the effectiveness of our method in handling large-scale action spaces.

7. CONCLUSION

This paper proposes a novel SDN-based MEC computation offloading architecture, comprehensively analyzes the basic requirements of the architecture, and uses the advantages of SDN centralized control to achieve the full potential of MEC. Moreover, we have studied the delay toleration and energy consumption budget computational tasks offloading process based on DRL in multi-user MEC. Furthermore, we proposed an entropy-based reinforcement learning approach to solve the processing delay minimization problem with delay and energy constraints. Numerical results verify that the presented E-DRL approach is effective and achieves better performance than baseline algorithms in SDN-based MEC computation offloading optimization problems. In the future, we will consider the effects of user devices mobility and investigate a joint dynamic MEC computation offloading and resource allocation problem.

References

[1] A. Ghasempour, "Internet of things in smart grid: Architecture, applications, services, key technologies, and challenges," *Inventions*, vol. 4, pp. 1–12, 2019.

[2] W. Shi and X. Zhang, "Edge computing: State-of-the-art and future directions," *Journal of Computer Research and Development*, vol. 56, pp. 69–89, 01 2019.

[3] J. Ren, Y. He, G. Huang, G. Yu, Y. Cai, and Z. Zhang, "An edge-computing based architecture for mobile augmented reality," *IEEE Network*, vol. 33, no. 4, pp. 162–169, 2019.

[4] J. Yan, S. Bi, and Y. J. A. Zhang, "Offloading and resource allocation with general task graph in mobile edge computing: A deep reinforcement learning approach," *IEEE Transactions on Wireless Communications*, vol. 19, no. 8, pp. 5404–5419, 2020.

[5] S. Bi and Y.-J. A. Zhang, "An admm based method for computation rate maximization in wireless powered mobile-edge computing networks," in *2018 IEEE International Conference on Communications (ICC)*, 2018, pp. 1–7.

[6] T. Z. He, A. N. Toosi, and R. Buyya, "Performance evaluation of live virtual machine migration in sdn-enabled cloud data centers," *Journal of Parallel and Distributed Computing*, vol. 131, no. SEP., pp. 55–68, 2019.

[7] Y. He, F. R. Yu, N. Zhao, V. C. M. Leung, and H. Yin, "Software-defined networks with mobile edge computing and caching for smart cities: A big data deep reinforcement learning approach," *IEEE Communications Magazine*, vol. 55, no. 12, pp. 31–37, 2017.

[8] L. Tello-Oquendo, I. F. Akyildiz, S.-C. Lin, and V. Pla, "Sdn-based architecture for providing reliable internet of things connectivity in 5g systems," in *2018 17th Annual Mediterranean Ad Hoc Networking Workshop (Med-Hoc-Net)*, 2018, pp. 1–8.

[9] S. Misra and N. Saha, "Detour: Dynamic task offloading in software-defined fog for iot applications," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 5, pp. 1159–1166, 2019.

[10] J. Luo, F. R. Yu, Q. Chen, and L. Tang, "Adaptive video streaming with edge caching and video transcoding over software-defined mobile networks: A deep reinforcement learning approach," *IEEE Transactions on Wireless Communications*, vol. 19, no. 3, pp. 1577–1592, 2020.

[11] X. Wang, Y. Han, V. Leung, D. Niyato, and X. Chen, "Convergence of edge computing and deep learning: A comprehensive survey," *IEEE Communications Surveys Tutorials*, vol. 22, no. 99, pp. 869–904, 2020.

[12] J. Wang, J. Pan, F. Esposito, P. Calyam, Z. Yang, and P. Mohapatra, "Edge cloud offloading algorithms: Issues, methods, and perspectives," *ACM Comput. Surv.*, vol. 52, no. 1, feb 2019.

[13] T. Subramanya, L. Goratti, S. N. Khan, E. Kafetzakis, I. Giannoulakis, and R. Riggio, "Sdec: A platform for software defined mobile edge computing research and experimentation," in *2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, 2017, pp. 1–2.

[14] R. N. Mensah, L. Zhiyuan, A. A. Okine, and J. M. Adeke, "A game-theoretic approach to computation offloading in software-defined d2d-enabled vehicular networks," in *2021 2nd Information Communication Technologies Conference (ICTC)*, 2021, pp. 34–38.

[15] P. Hu and W. Chen, "Software-defined edge computing (sdec): Principles, open system architecture and challenges," in *2019 IEEE SmartWorld, Ubiquitous Intelligence Computing, Advanced Trusted Computing, Scalable Computing Communications, Cloud Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*, 2019, pp. 8–16.

[16] S. Misra and S. Bera, "Soft-van: Mobility-aware task offloading in software-defined vehicular network," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 2, pp. 2071–2078, 2020.
[17] L. Ji and S. Guo, "Energy-efficient cooperative resource allocation in wireless powered mobile edge computing," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4744–4754, 2019.
[18] H. Satake, Y. Kobayashi, R. Tani, and H. Shigeno, "Dynamic task offload system adapting to the state of network resources in mobile edge computing," in *2020 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, 2020.
[19] L. Huang, S. Bi, and Y.-J. A. Zhang, "Deep reinforcement learning for online computation offloading in wireless powered mobile-edge computing networks," *IEEE Transactions on Mobile Computing*, vol. 19, no. 11, pp. 2581–2593, 2020.
[20] X. Ma, C. Lin, Z. Han, and J. Liu, "Energy-aware computation offloading of iot sensors in cloudlet-based mobile edge computing," *Sensors*, vol. 18, no. 6, p. 1945, 2018.
[21] P. Zhao, W. Zhao, H. Bao, and B. Li, "Security energy efficiency maximization for untrusted relay assisted noma-mec network with wpt," *IEEE Access*, vol. 8, pp. 147 387–147 398, 2020.
[22] J. Wang, J. Hu, G. Min, W. Zhan, Q. Ni, and N. Georgalas, "Computation offloading in multi-access edge computing using a deep sequential model based on reinforcement learning," *IEEE Communications Magazine*, vol. 57, no. 5, pp. 64–69, 2019.
[23] X. Chen, H. Zhang, C. Wu, S. Mao, Y. Ji, and M. Bennis, "Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4005–4018, 2019.
[24] Q. Tang, R. Xie, F. R. Yu, T. Huang, and Y. Liu, "Decentralized computation offloading in iot fog computing system with energy harvesting: A dec-pomdp approach," *IEEE Internet of Things Journal*, vol. 7, no. 6, pp. 4898–4911, 2020.
[25] S. Retal, M. Bagaa, T. Taleb, and H. Flinck, "Content delivery network slicing: Qoe and cost awareness," in *2017 IEEE International Conference on Communications (ICC)*, 2017, pp. 1–6.
[26] C. Liang, Y. He, F. R. Yu, and N. Zhao, "Enhancing video rate adaptation with mobile edge computing and caching in software-defined mobile networks," *IEEE Transactions on Wireless Communications*, vol. 17, no. 10, pp. 7013–7026, 2018.
[27] Y. Hou, "Wireless communication using electromagnetic wave with orbital angular momentum," in *2020 International Conference on Computing and Data Science (CDS)*, 2020, pp. 99–103.
[28] G. Faraci and A. Lombardo, "An nfv approach to share home multimedia devices," in *2017 IEEE Conference on Network Softwarization (NetSoft)*, 2017.
[29] X. Tang, W. Cao, H. Tang, T. Deng, J. Mei, Y. Liu, C. Shi, M. Xia, and Z. Zeng, "Cost-efficient workflow scheduling algorithm for applications with deadline constraint on heterogeneous clouds," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 9, pp. 2079–2092, 2022.
[30] T. Liu, Y. Xu, Z. Zhang, and J. Wu, "Long-term auction for inner-dependent task offloading in blockchain-enabled edge computing," in *2021 40th Chinese Control Conference (CCC)*, 2021, pp. 1881–1886.
[31] G. Ciuperca, V. Girardin, and L. Lhote, "Computation and estimation of generalized entropy rates for denumerable markov chains," *IEEE Transactions on Information Theory*, vol. 57, no. 7, pp. 4026–4034, 2011.
[32] H. Ge, Y. Song, C. Wu, J. Ren, and G. Tan, "Cooperative deep q-learning with q-value transfer for multi-intersection signal control," *IEEE Access*, vol. 7, pp. 40 797–40 809, 2019.
[33] Q. Wei, H. Ma, C. Chen, and D. Dong, "Deep reinforcement learning with quantum-inspired experience replay," *IEEE Transactions on Cybernetics*, pp. 1–13, 2021.
[34] Y. Ren, X. Yu, X. Chen, S. Guo, and Q. Xue-Song, "Vehicular network edge intelligent management : A deep deterministic policy gradient approach for service offloading decision," in *2020 International Wireless Communications and Mobile Computing (IWCMC)*, 2020, pp. 905–910.
[35] Y. Wang, W. Fang, Y. Ding, and N. Xiong, "Computation offloading optimization for uav-assisted mobile edge computing: a deep deterministic policy gradient approach," *Wireless Networks*, pp. 1–16, 2021.
[36] H. Zhang, F. Wang, J. Wang, and B. Cui, "Robot grasping method optimization using improved deep deterministic policy gradient algorithm of deep reinforcement learning," *Review of Scientific Instruments*, vol. 92, no. 2, p. 025114, 2021.