Online Deep Reinforcement Learning for Computation Offloading in Blockchain-Empowered Mobile Edge Computing

Xiaoyu Qiu, Luobin Liu, Wuhui Chen, *Member, IEEE,* Zicong Hong and Zibin Zheng, *Senior Member, IEEE*

***Abstract*—Offloading computation-intensive tasks (e.g., blockchain consensus processes and data processing tasks) to the edge/cloud is a promising solution for blockchain-empowered mobile edge computing. However, the traditional offloading approaches (e.g., auction-based and game-theory approaches) fail to adjust the policy according to the changing environment and cannot achieve long-term performance. Moreover, the existing deep reinforcement learning-based offloading approaches suffer from the slow convergence caused by high-dimensional action space. In this paper, we propose a new model-free deep reinforcement learning-based online computation offloading approach for blockchain-empowered mobile edge computing in which both mining tasks and data processing tasks are considered. First, we formulate the online offloading problem as a Markov decision process by considering both the blockchain mining tasks and data processing tasks. Then, to maximize long-term offloading performance, we leverage deep reinforcement learning to accommodate highly dynamic environments and address the computational complexity. Furthermore, we introduce an adaptive genetic algorithm into the exploration of deep reinforcement learning to effectively avoid useless exploration and speed up the convergence without reducing performance. Lastly, our experimental results demonstrate that our algorithm can converge quickly and outperform three benchmark policies.**

***Index Terms*—online computation offloading, blockchain, mobile edge computing, deep reinforcement learning**

I. Introduction

MOBILE edge computing (MEC) is a promising solution that allows mobile devices to run the highly demanding applications by providing computational resources. However, building trust among multiple parties (e. g. different mobile users and edge/cloud providers) in MEC is a challenge because these parties usually have conflicting interests. Fortunately, blockchain technologies taking advantage of decentralization, anonymity, and trust have begun to exert a significant influence on MEC [1] [2] . Nonetheless, running blockchain mining processes (e.g., performing Proof of Stake (PoS)) while supporting increasingly intelligent applications (processing/analyzing tasks) can require vast computing and storage resources [3]. Thus, the limited computing and storage capacities of mobile devices are restricting real-world blockchain-empowered mobile applications [4]. Therefore, it is vital to develop a computation offloading solution that can extend the capacities of mobile devices by offloading computation-intensive tasks (e.g., blockchain consensus processes) to the edge/cloud servers.

To develop the computation offloading solution for blockchain-empowered MEC, social welfare maximization auction-based approaches [5] (e.g., sealed-bid auction, combinatorial auction, forward, reverse, and double auction) and game-theory approaches [6] (e.g., noncooperative game, stackelberg game [7] [8], and bargaining game) have been proposed, in which the blockchain mining tasks can be offloaded to the edge/cloud servers. These methods often require many iterations for an algorithm to reach a satisfying optimum [9]. In addition, the above approaches are mostly built in consideration of one-shot optimization, they may not apply well to maximize the long-term performance of computation offloading. Further, these algorithms usually assume specific models for the computation offloading solutions, which may not accurately characterize the realistic environment.

Deep reinforcement learning (DRL) is emerging as an effective approach to obtain the optimal decision-making policy and maximize the long-term rewards [10]. First, the combination of reinforcement learning and deep learning [11] has been extensively researched to create strategies for computation offloading. Kawamoto et al. [12] utilized Q-learning to construct an unmanned aircrafts communication management system. However, the performance of the algorithm is greatly affected by the convergence and accuracy of the deep Q-network (DQN). Second, improving the performance of a DQN for computation offloading has attracted much attention recently. Zhang et al. [13] proposed a modified DQN by using the stacked autoencoder, which achieved an outstanding resource management in cloud servers. However, the dimension of action space would expand exponentially when considering multi-hop multi-user MEC systems. Third, some recent studies have attempted to address the high complexity problem. Mnih et al. [10] combined a value-based method with a policy-based method and proposed asynchronous methods for deep reinforcement learning. Nonetheless, to our knowledge, the slow convergence caused by high-dimensional action space remains an urgent and challenging problem.

In this paper, we study the computation offloading problem for both mining tasks and data processing tasks in multi-hop multi-user blockchain-empowered MEC. To solve the problem, we propose a new model-free DRL-based online computation offloading algorithm, namely deep reinforcement learn-

X. Qiu, L. Liu, W. Chen, Z. Hong and Z. Zheng are with School of Data and Computer Science, Sun Yat-sen University, China and National Engineering Research Center of Digital Life, Sun Yat-sen University, Guangzhou 510006, China. Email: qiuxy23@mail2.sysu.edu.cn, kelvin.liu1221@gmail.com, {chenwuh@mail.sysu.edu.cn}, hongzc@mail2.sysu.edu.cn, zhzibin@mail.sysu.edu.cn

ing combined with genetic algorithm (DRGO). The DRGO algorithm aims to maximize long-term rewards and accommodate highly dynamic environments via deep reinforcement learning. In particular, to speed up the convergence caused by high-dimensional action space, the DRGO algorithm adopts adaptive genetic algorithm (AGA) [14] during exploration in deep reinforcement learning, which effectively avoids useless exploration to a great extent and reduces the computational complexity of deep reinforcement learning. To our knowledge, this is the first time that AGA has been introduced to solve the large action space problem of DRL.

Our contributions can be mainly summarized as follows.

1) We are the first to study the online computation offloading need for both the blockchain mining tasks and data processing tasks in multi-hop multi-user blockchain-empowered MEC. We then formulate the online offloading problem as a Markov decision process (MDP) to consider the dynamic environments.
2) To adopt the dynamic environments and achieve long-term offloading performance, we leverage the deep reinforcement learning to obtain the optimal offloading policy based on the past offloading experience. In this context, a considerable number of iterations for reaching a satisfying optimum can be avoided.
3) To speed up the convergence, we introduce adaptive genetic algorithm into the exploration of deep reinforcement learning. In this way, we can make the most of the evaluating role of critic network in exploration, avoiding useless exploration to a great extent without reducing the performance.

The remainder of the paper is structured as follows. In Section II, we review the related work. In Section III, we present a system model and Section IV describes the problem formulation. In Section V, we introduce the proposed DRL-based method for computation offloading, and Section VI presents the numerical simulations. Section VII concludes.

II. RELATED WORKS

Many studies have examined the computation offloading need for edge computing or cloud computing [15]. Liu et al. [16] proposed a multiplier-based computation offloading algorithm, where the blockchain mining tasks can be offloaded to nearby edge computing nodes. Chen et al. [17] transformed the resources allocation problem of computation offloading into a mixed-integer linear programming problem to improve quality-of-service and reduce execution times. Chatzopoulos et al. [18] proposed a hidden market design-based offloading approach that allows users to specify the amount of resources they are willing to share. However, all these algorithms are constrained by the trade-off between efficiency and optimality. Moreover, they consider immediate rewards and assume specific models that may be incompatible.

Recently, artificial intelligence approaches such as deep learning [19] [20] and reinforcement learning [21] have emerged as effective measures to solve computation offloading problems. A resource allocation algorithm based on deep Q-learning is proposed in [22] to optimize the performance of computation offloading in MEC. Liu et al. [23] proposed a deep learning-based resource allocation algorithm to enable high energy efficiency and low power consumption. Qi et al. [24] explored DRL to obtain the optimal offloading decision for the Internet of Vehicles. However, to apply DRL to computation offloading, the major challenge lies in the convergence and accuracy of the deep neural network (DNN).

To accelerate the learning process, transfer learning integrating with DRL is proposed in [25] to reduce the random exploration at the initial learning process. Min et al. [26] compressed the state space by utilizing a deep convolutional neural network (CNN) and achieved a near-optimal offloading policy. In contrast, to avoid overestimation and reduce bias, Wang et al. [9] devised two double DQN-based algorithms to address the cost-minimization problem of D2D offloading. However, when considering the multi-hop multi-user environment, the exponential explosions of the state space and action space are inevitable and prevent the above approaches from working.

A few recent studies have attempted to solve the exponential explosion of the action space. Wei et al. [27] proposed a policy gradient-based actor-critic algorithm for the computation offloading and content caching problems with continuous-valued state and action variables. Zhu et al. [28] proposed a hybrid actor-critic method, which asynchronously trained the network parameters and decreased the long-term offloading cost. In addition, although methods such as asynchronous advantage actor-critic (A3C) [29] or deep deterministic policy gradient (DDPG) [30] can solve the problem of exponential explosion, they also lead to slow convergence. Because of the high-dimensional action space, traditional exploration methods such as the ϵ-greedy policy are obviously inefficient and inadequate. Therefore, we develop a DRL-based online offloading algorithm to address the challenge. In particular, to speed up the convergence, we adopt AGA during exploration, which can also avoid useless exploration to a great extent and reduce the computational complexity.

III. SYSTEM MODEL

In this work, we study the computation offloading need for blockchain-empowered MEC, where nearby IoT devices are grouped as communities such as smart home and smart industrial. Because these IoT devices belong to the same communities, it is reasonable to model the system as an optimization problem about long-term total reward. Fig. 1 shows the architecture of our proposed system, which consists of the following components.

1) Blockchain network: To provide security and privacy for IoT devices, the blockchain technology is introduced to MEC, which is known for its security and immutability. A blockchain works as a decentralized database and stores data as blocks after validation. Some nodes, known as miners, collect transactions from the whole blockchain network and attempt to perform consensus processes such as Proof-of-Work (PoW) or PoS. In this way, the first miner who reaches consensus is rewarded and a new block is connected to the whole blockchain after validation. Therefore, the data contained in the block cannot be tampered with or forged.

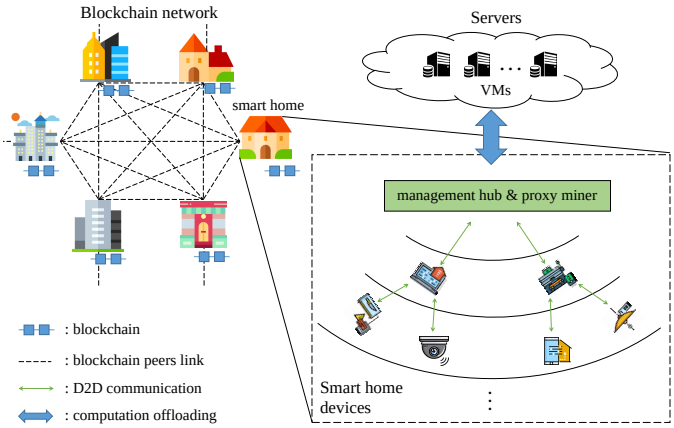


Fig. 1. Multi-hop blockchain-empowered mobile edge computing

2) Computation offloading for data processing tasks: In the system model, end devices are connected through a multi-hop ad-hoc network that supports data transmission and computation offloading. Assuming there are N end devices (EDs), we let $\mathcal{N} = \{1, 2, 3, ..., N\}$ denote the set of the EDs. Each ED $i (i \in \mathcal{N})$ is uniquely identified using an account in the blockchain network. We denote b_i (tokens) as the balance of the account associated with ED i, that can be used to purchase computational resources for data processing tasks, such as 3-D sensing and augmented reality. Because of the limited computational power of EDs, executing the data processing tasks locally could cause a long delay. Therefore, the tasks can be scheduled to be executed locally or offloaded to edge/cloud servers. In this work, we introduce the concept of task queue, which is a data structure that contains a list of tasks to be performed in turn. If the computational resources are occupied at the current time epoch, tasks can be queued at the task queue until the resources are available. The brief information of the generated tasks is transmitted to a management hub, which is responsible for making offloading decisions for EDs to achieve higher performance.

3) Computation offloading for blockchain mining tasks: As part of the blockchain network, EDs can participate in the mining process to obtain rewards. However, the mining process is not performed locally because of resource constraints. In general, EDs can entrust proxy miners to purchase computational resources from edge/cloud servers and perform the consensus processes in the edge/cloud servers [31].

4) Resource purchase scheme in edge/cloud servers: We assume that a virtual machine (VM) is the smallest unit to perform the tasks [32] and can only execute one task at the same time. In the servers, VMs are divided into different levels to meet the requirements of various types of offloading requests [32]. In this paper, we assume that VMs for data processing tasks in the servers are divided into k levels, which are denoted by $L^c = \{1, 2, ..., k\}$. Similarly, VMs for mining tasks are divided into j levels that are denoted by $L^m = \{1, 2, ..., j\}$. In general, the edge servers provide the resources for computation. In the case that they cannot serve the EDs, they offload the tasks to the cloud servers and charge the communication and computation cost to the EDs.

5) Trade-off between two tasks: As the offloading of both tasks needs to purchase resources from servers, it is important to consider the trade-off between both tasks. For example, if an ED spends most of its tokens in blockchain mining at the current time epoch, it may fail to execute the incoming data processing tasks because the mining cost is paid immediately, while the reward is obtained until reaching consensus. On the other hand, the reward obtained after reaching consensus can be used to improve the performance of data processing tasks. Therefore, it is critical to jointly consider both tasks. We integrate the management hub for data processing tasks with the proxy miner for mining tasks as a joint management module (JM).

6) Multi-hop ad-hoc network: In the ad-hoc network, some EDs are directly connected to the JM, but some may not connect because of distance or device limitations. However, some nodes can communicate with their neighboring peers and help their neighboring peers transfer data to the JM. Therefore, similar to [33], we divide EDs into different hierarchical levels based on their network topology. We assume EDs that directly connect to JM are in level 1. In the same manner, EDs that can only send data to the JM via neighboring EDs in level 1 are in level 2. Subsequent levels are formatted similarly. To avoid data transmission congestion and adapt to the dynamic environment, each ED can dynamically select nearby devices as its next hop to transmit data.

IV. PROBLEM FORMULATION

A. State and Action

At each decision epoch t, the state of the system can be characterized by $S_t = (S^m, M^c, \mathcal{G}, P^c, F^c, P^m, F^m, Q^s, H)$. The meaning of each variable is as follows.

- $S^m = \{S_i = (q_i, f_i, u_i, b_i) | i \in \{1, 2, .., N\}\}$ denotes the collected states of all EDs, including the tasks queue length q_i, the CPU frequency of local device f_i, the unique network ID u_i and the balance b_i of the associated account.
- $M^c = \{M_i^c = (l_i, d_i, v_i, s_i^u, s_i^d) | i \in \{1, 2, .., N\}\}$ denotes the brief information of all data processing tasks of EDs, including CPU cycles l_i, the deadline of the task d_i, completion reward v_i, uploading data size s_i^u, and downloading data size s_i^d.
- $\mathcal{G}$ is the current network state, including the topology and data transmission rates between each ED. If ED i cannot communicate with ED j, the data transmission rate between ED i and ED j is set as 0.
- $P^c = \{p_1^c, p_2^c, ..., p_k^c\}$ and $F^c = \{f_1^c, f_2^c, ..., f_k^c\}$ denote the prices and CPU frequencies of different VM levels for data processing tasks, respectively. Similarly, $P_m = \{p_1^m, p_2^m, ..., p_j^m\}$ and $F_m = \{f_1^m, f_2^m, ..., f_j^m\}$ are for mining VMs.
- Q^s denotes the length of the task queue in the servers.
- Lastly, H is the estimation of the hash power (also called hash rate) of the blockchain network, representing the computational power that the blockchain network is consuming. H determines how many hash operations can

TABLE I
NOTATION DEFINITIONS

Symbol	Definition
$\mathcal{N}$	Set of EDs
N	Number of EDs
k, j	Number of VM levels for data processing and mining tasks
L^c, L^m	Set of VMs levels for data processing and mining tasks
P^c, P^m	Set of VMs' prices for data processing and mining tasks
F^c, F^m	Set of VMs' computational capacity for data processing and mining tasks
S_t	State of system at decision epoch t
S^m	Set of collected states of EDs
M^c	Set of brief informations of data processing tasks
$\mathcal{G}$	Network state
Q^s	Task queue length in servers
H	Estimation of hash power of blockchain network
A_t	Set of offloading decision for EDs
q_i	Queue length of ED i
f_i	Computation ability of ED i
u_i	Network ID of ED i
b_i	Balance of ED i
M_i^c	State of data processing task of ED i
l_i	Required CPU cycles for task M_i^c
d_i	Deadline for task M_i^c
a_i	Completion reward for task M_i^c
s_i^u, s_i^d	Uploading and downloading data size for task M_i^c
σ	Unit energy consumption of CPU cycles (joules per cycle)
p^e	Unit price of energy (tokens per joule)
$K_{<i,j>}$	Maximal achievable data rate between ED i and ED j
N_i	Number of EDs that transmit data via ED i
W_i	Transmit power of ED i
γ_i	Relative hash power of ED i
$\mathcal{R}$	Reward of the first miner reaching consensus
Θ	Task failure penalty
p_c, p_m	Crossover probability and mutation probability
$Q(S, A)$	State-action value function
$K_{\max}$	Max number of generation in AGA
θ^μ, θ^Q	Parameters of actor network and critic network
φ	Preset value to evaluate critic network

be performed per second and can be estimated based on the compact status reports issued by the miners [34].

Based on the system state S_t at decision epoch t, JM makes a task offloading action $A_t = \{a_i | i \in \{1, 2, .., N\}\}$ and sends a_i to the corresponding ED i. Each action a_i can be represented as (c_i, m_i, n_i). $c_i \in \{0, 1, 2, ..., k\}$ denotes the offloading decision of data processing tasks. If $c_i = 0$, ED i executes the tasks locally. Otherwise, ED i purchases a corresponding level of VM and offloads tasks to servers. Similarly, $m_i \in \{0, 1, 2, ..., j\}$ denotes the offloading decision of data mining tasks. If $m_i = 0$, ED i does not decide to mine. In addition, to offload the tasks, the required data and program code need to be transmitted to the servers. Considering the multi-hop scenario that we discussed, n_i denotes the network ID of the next hop in the upper level. In this work, the action is coded in binary. For example, if n_i = 3, we convert it to '11'.

B. Cost of Data Processing Task

1) Local Execution Model: If a data processing task is executed locally at ED, that is, $c_i = 0$, ED i need to consume a large amount of energy to compute its tasks. In our proposed blockchain-empowered system, we convert the cost of energy consumption into tokens to unify units. Let σ denote the unit energy consumption of CPU cycles (joules per cycle), l_i denote the required CPU cycles and p^e denote the unit price of energy (tokens per joule). Thus, the local execution cost of ED is as follows.

$$C_i^{\text{local}} = \sigma l_i p^e \cdot 1_{\{c_i=0\}}, \tag{1}$$

where $1_{\{\Omega\}}$ is the indicator function, that is, $1_{\{\Omega\}}$ equals to 1 if the condition Ω is satisfied. Otherwise, $1_{\{\Omega\}}$ equals to 0.

2) Offloading Computation Model: Considering the limited computational power of EDs, they can offload their tasks to the servers to achieve better performance. If ED i decides to offload its tasks to servers, the corresponding price for VM of level c_i must be paid, which is as follows.

$$X_i^C = l_i p_{c_i}^c \cdot 1_{\{c_i \in [1,k]\}}, \tag{2}$$

where $p_{c_i}^c$ is the price of the purchased VM (token per Gcycle).

In addition, to offload tasks, EDs need to upload the required data and program code to the servers and download the return data back with the assistance of JM or their neighboring peers. We denote the maximal achievable transmission rate between ED i and ED j as $K_{<i,j>}$. It is worth noting that ED i may receive transmission requests from other EDs and we assume the bandwidth allocated to each ED is equal. Therefore, the expected rate of transmitting data from ED i to ED j can be written as follows:

$$r_i = \frac{K_{<i,j>}}{N_i}, \tag{3}$$

where N_i is the number of EDs that transmit data via ED i.

To offload tasks to servers, EDs should pay for the energy consumption during data transmission. We denote $\mathcal{U}_i$ as the set of EDs that transmit data via ED i (including itself), thus the cost of data transmission in our offloading computation model is as follows:

$$C_i^{\text{trans}} = \frac{\sum_{j \in \mathcal{U}_i} (s_j^u + s_j^d)}{r_i} \cdot \mathcal{W}_i p^e \cdot 1_{\{c_i \in [1,k]\}}, \tag{4}$$

where $\mathcal{W}_i$ is the transmit power of ED i. s_j^u and s_j^d are the uploading and downloading data sizes of ED j. And p^e denotes the unit price of energy (token per joule).

To sum up, the total cost for offloading task is as follows:

$$C_i^{\text{offload}} = \left(X_i^C + \frac{\sum_{j \in \mathcal{U}_i} (s_j^u + s_j^d)}{r_i} \cdot \mathcal{W}_i p^e \right) \cdot 1_{\{c_i \in [1,k]\}}. \tag{5}$$

C. Reward of Mining Task

1) Reward of Blockchain Mining: If $m_i \neq 0$, that is, ED i decides to perform the mining task in servers, it needs to compete for accounting rights with other miners by calculating the numerical solutions of a random hash. In a blockchain network, the hash power determines how many hash operations can be performed per second. According to [35], the relative hash power of ED i to the blockchain network is as follows:

$$\delta_i = \frac{f_{m_i}^m}{H}, \tag{6}$$

where $f_{m_i}^m$ is the hash power of the VMs that ED i purchase, and H is the estimation of the hash power of the blockchain network, which is based on the compact status reports issued by miners [34]. In general, as δ_i increases, the possibility to reach a consensus increases, and the expected reward increases.

In the blockchain network, if a well-formed block is no longer part of the longest and well-formed blockchain, called an orphan block, the miner who mined that block does not actually get the reward (or the transaction fees). According to [4], the possibility of a block becoming an orphan block is approximately $1 - e^{-\lambda v(s^m)}$, where $-\lambda$ denotes a constant rate, s^m is the size of the block to be mined and $v(s^m)$ denotes a function of block size. We set $\mathcal{R}$ as the rewards of the first miner that reaches consensus. Therefore, the expected rewards of ED i are:

$$R_i = e^{-\lambda v(s^m)} \delta_i \cdot \mathcal{R} \cdot 1_{\{m_i \in [1,j]\}}. \tag{7}$$

2) Cost of Blockchain Mining: Based on our proposed blockchain-empowered MEC, a payment is required for the allocated computational resources when ED i decides to perform the mining tasks in the servers, which is as follows:

$$X_i^M = p_{m_i}^m \cdot 1_{\{m_i \in [1,j]\}}. \tag{8}$$

Overall, the total reward of mining tasks can be represented as follows:

$$R_i^{\text{mine}} = \left(e^{-\lambda v(s^m)} \delta_i \cdot \mathcal{R} - X_i^M \right) \cdot 1_{\{m_i \in [1,j]\}}. \tag{9}$$

D. Task Failure Cost and Completion Reward

1) Local Execution Failure: In this part, we consider the failure cost of the data processing tasks. In our model, the computation resources of local devices are not infinite. We assume tasks performed locally are queued at the task queue based on the first-in-first-out principle. If the task queue is full, the generated task will be rejected and EDs will receive a penalty Θ. We denote the task queue of ED i at the current time epoch as q_i and the maximum task queue length as $q_i^{\max}$. In addition, we assume that the queuing delay is the estimated execution time of all the prior tasks in the queue and is denoted as d_i^{queue}. Thus, the task delay for local execution is determined by the queuing delay and execution delay, which can be represented as:

$$d_i^{\text{local}} = \left(d_i^{\text{queue}} + \frac{l_i}{f_i} \right) \cdot 1_{\{c_i = 0\}}, \tag{10}$$

where l_i is required CPU cycles and f_i is the CPU frequency of ED i. If $d_i^{\text{local}} > d_i$, that is, the task delay is larger than the deadline, the execution of the task fails and ED i should pay a penalty. Therefore, the failure penalty for local execution is defined as:

$$\Theta_i^{\text{local}} = \Theta_{\{\{q_i = q_i^{\max}\} \vee \{d_i^{\text{local}} > d_i\}\}}, \tag{11}$$

where $\vee$ means "logic OR".

2) Offload Execution Failure: Similarly, the task delay for offloading is determined by the transmission delay, queue delay and execution delay. First, we denote L_i as the complete routing path from ED i to servers, which contains the network IDs of all passing EDs and JM. Therefore, the transmission delay of ED i is as follows:

$$d_i^{\text{trans}} = \sum_{j \in L_i} \frac{(s_i^u + s_i^d) \cdot N_j}{r_j}, \tag{12}$$

where s_i^u and s_i^d are the uploading data size and downloading data size respectively. N_j is the number of EDs that transmit data via ED j ($j \neq 0$) or JM ($j = 0$) and r_j is the data transmission rate of ED j ($j \neq 0$) or JM ($j = 0$).

Second, we denote the task queue at the servers as q_s, the maximum task queue length as $q_s^{\max}$, and the estimated queuing time as d_s^{queue}. Therefore, the sum of the queue delay and the execution delay of the offloaded tasks is as follows:

$$d_i^{\text{execute}} = \frac{l_i}{p_{c_i}^c} + d_s^{\text{queue}}, \tag{13}$$

where l_i is the required CPU cycles and $p_{c_i}^c$ is the CPU frequency of the purchased VM.

By using Equation (12) and (13), the task delay can be represented as:

$$d_i^{\text{offload}} = \sum_{j \in L_i} \frac{(s_i^u + s_i^d) \cdot N_j}{r_j} + \frac{l_i}{p_{c_i}^c} + d_s^{\text{queue}}. \tag{14}$$

In addition, similar to local execution, if the task delay exceeds the deadline d_i, ED i receives a penalty that can be represented as:

$$\Theta_i^{\text{offload}} = \Theta_{\{\{d_i^{\text{offload}} > d_i\} \vee \{q_s = q_s^{\max}\}\}}. \tag{15}$$

3) Completion Reward: If a data processing task is successfully executed, ED i receives a reward v_i. Thus, the completion reward of ED i is as follows:

$$V_i = v_i \cdot 1_{\{\Theta_i^{\text{local}} = 0\} \wedge \{\Theta_i^{\text{offload}} = 0\}}, \tag{16}$$

where $\wedge$ means "logic AND".

E. Cost of System

To sum up, using Equations (1), (5), (9), (11), (15) and (16), the cost of ED i is:

$$C_i = C_i^{\text{local}} + C_i^{\text{offload}} - R_i^{\text{mine}} + \Theta_i^{\text{local}} + \Theta_i^{\text{offload}} - V_i. \tag{17}$$

From the perspective of system, the instantaneous system cost can be represented as:

$$C_{\text{total}} = \sum_{i \in \mathcal{N}} C_i, \tag{18}$$

where $\mathcal{N} = \{1, 2, 3, ..., N\}$ is the set of EDs.

F. MDP-Based Optimization Problem

In our performance optimization problem, it is important to note that costs such as energy consumption and purchasing VMs should be paid immediately. However, the task completion rewards are not given until the tasks are successfully executed. Similarly, the mining rewards are not given until reaching consensuses. If an ED spends most of its tokens on blockchain mining or the current data processing tasks, it may obtain many rewards after achieving consensuses or finishing the tasks. However, it may not have enough tokens for the incoming data processing tasks and thus suffers a penalty. Therefore, it is important to consider the computation offloading problem in the long run.

In this paper, we formulate the computation offloading problem as a discrete time MDP, which is widely used to address sequential stochastic decision-making problems. In this context, the considerations we discussed above are automatically considered in an MDP model, which can be defined as a tuple $< \mathbf{S}, \mathbf{A}, P, R(\cdot), \mathbf{T} >$:

- $\mathbf{S}$: the state space
- $\mathbf{A}$: the action space
- P: the transition probability from taking action A_t under state S_t to next state S_{t+1}
- $R(\cdot)$: the immediate reward function that takes the state–action pair as input
- $\mathbf{T}$: a sequence of time

In our scenarios, the immediate reward function $R(\cdot)$ of the MDP equals to $-C_{\text{total}}(\cdot)$, which is the system cost function in Equation (18). Therefore, the objective of this research is to find an optimal policy π^* that makes decisions to maximize the long-term reward, which can be defined as:

$$\max \mathbb{E}_{\pi,S}\left[\sum_{t=1}^{\mathbf{T}} R(S_{t+1}|S_t, \pi(A_t))\right]. \tag{19}$$

V. ALGORITHM DESIGN

Deep reinforcement learning (DRL) is emerging as an effective approach to obtain the optimal decision-making policy and maximize the long-term rewards [10]. In traditional deep reinforcement learning, a DNN is used to approximate the mapping from the current system state S_t to the expected rewards or selected probabilities of all possible actions. Because it requires exhaustive search through all actions, it is unsuitable for problems with high-dimensional action spaces, which leads to slow convergence [10]. To support problems with high-dimensional action spaces, approaches such as A3C [29] or DDPG [30] are proposed, which combine policy gradient with value function. These algorithms use two kind of neural networks: actor network and critic network. The actor network makes action based on the state, while the critic network evaluates the action taken. However, they often require specific design for the exploration mechanism to explore the complex action space. Otherwise, it inevitably leads to a largely unexplored action space.

To address the challenge, we propose an online computation offloading framework combined with DRL and AGA. The AGA is introduced in the exploration phase to avoid largely unexplored action space. The next section focuses on the details of our proposed algorithm and its update strategy.

A. Overview of the DRGO Algorithm

Fig. 2 shows that the DRGO algorithm consists of two important components: an actor network and a critic network. In the actor network, rather than outputting the expected rewards or possibilities of all possible actions, it works as a policy π that takes system states S_t as inputs and outputs an action A_t at decision epoch t. For any given system state S_t, the offloading policy π can be defined as a mapping:

$$\pi : S_t \mapsto A. \tag{20}$$

And for any action A, the critic network is used to evaluate the expected long-term reward, which can be defined as a mapping.

$$Q : (S_t, A) \mapsto R. \tag{21}$$

In the system, JM receives requests from EDs periodically and combines them to system state S_t. The DRGO algorithm is triggered regularly or when JM receives a certain amount of requests from EDs. In this context, the decision epoch is changeable in practice, which is more realistic for actual situations. At decision epoch t, the actor network takes system state S_t as input and produces an action used in exploration or exploitation. We denote the action after exploration or exploitation as A_t. Next, JM transmits A_t to EDs to obtain the next state S_{t+1} and reward R_t. Then, we store (S_t, A_t, R_t, S_{t+1}) in replay memory. At each training epoch, we select samples from the replay memory to update the parameters in the actor network and the critic network toward maximizing the long-term reward.

B. Using AGA to Generate the Candidate Solution Set

The actor network works as a policy π that takes system state S_t as the inputs and produces a action A (Algorithm 1: line 6). To maximize the expected long-term reward of the system, we need to train the actor network toward outputting the optimal action A^*. In traditional reinforcement learning method such as DDPG, an entropy bonus is added to ensure sufficient exploration. However, to fully explore all actions, it consumes a great amount of time and leads to low data utilization.

Therefore, in this work, two methods are used to guarantee both efficiency and sufficiency of the exploration process. As mentioned, the critic network can be used to evaluate the expected long-term reward for each state-action pair on condition that it is well-trained. And the loss value is used to measure the inconsistency between the predicted value and the actual value. Therefore, if the loss value of the critic network is larger than the preset value φ, that is, the critic network cannot evaluate candidate actions very well, we use the random mechanism to generate a new action after obtaining the output of the actor network for sufficiency. Otherwise, we use the AGA for efficiency (Algorithm 1: line 7). The preset value φ

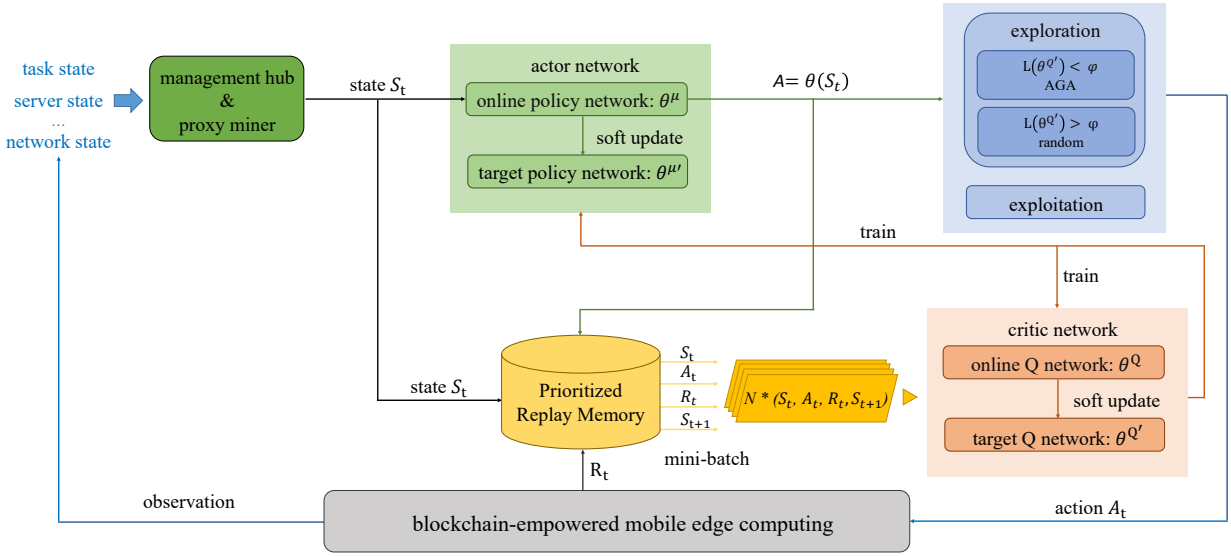


Fig. 2. DRL combined with genetic algorithm for computation offloading in blockchain-empowered MEC.

can be set as the loss value when the critic network converges, which can be obtained by pretraining the critic network.

The usage of AGA in the exploration process is illustrated in Fig 3. First, it combines the output of the actor network A with $m-1$ randomly generated solutions, where m is the size of the population. Then it uses genetic operators such as crossover and mutation to generate the candidate solution set $\mathbb{A} = \{A_t^1, A_t^2, ..., A_t^m\}$ as the next and subsequent generations. For each generation of AGA, we use the critic network to measure the performance of each action and use selection operators to select candidates that contribute to the population of the next generation. AGA terminates when the number of generations has reached a preset maximum number $K_{\max}$.

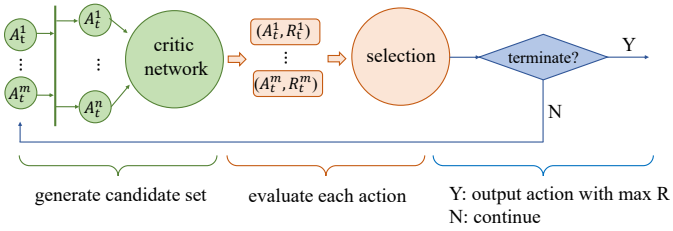


Fig. 3. High-efficiency DRL exploration using AGA.

Here, we introduce two important terms, the probability of crossover (p_c), which indicates the converge characteristic, and the probability of mutation (p_m), which indicates the characteristic of breaking through the limitation of the current search space. If the population tends to stay at the local optimum, we increase the values of p_c and p_m. Otherwise, we decrease the values of p_c and p_m. One way to measure the convergence ability is $\overline{R} - R_{\min}$, where $\overline{R}$ is the average reward and $R_{\min}$ is the minimum reward. It is obvious that $\overline{R} - R_{\min}$ is likely to be lower when a population converges to a local optimum (or global optimal) solution. In this way, p_c and p_m can be expressed as:

$$p_c = \frac{k_1}{\overline{R} - R_{\min}}, \tag{22}$$

$$p_m = \frac{k_2}{\overline{R} - R_{\min}}. \tag{23}$$

Increasing p_c and p_m may cause disruptions of the near-optimal solutions when a population reaches a global optimal solution. To solve this problem, p_c should depend on the reward of its parent R', and p_m should depend on the reward of itself R. As $R_{\min} - R'$ increases, p_c should also increase. Similarly, as $R - R_{\min}$ increases, p_m should also increase. In addition, we preset k_3 and k_4 as the recommended values when p_c and p_m increase beyond 1. Therefore, the expressions for p_c and p_m are

$$p_c = \begin{cases} \dfrac{k_1(R_{\min} - R')}{\overline{R} - R_{\min}} & R' \le R_{\min} \\ k_3 & R' > R_{\min} \end{cases}, \tag{24}$$

$$p_m = \begin{cases} \dfrac{k_2(R - R_{\min})}{\overline{R} - R_{\min}} & R \ge R_{\min} \\ k_4 & R < R_{\min} \end{cases}, \tag{25}$$

where k_1, k_2, k_3, and $k_4 < 1.0$.

C. Selecting from the Candidate Set in AGA

In the next step, we use the critic network to select from the candidate set by estimating $Q(S_t, A)$ for each state–action pair in the candidate solution set $\mathbb{A}$. Considering a stationary task offloading policy π, the system follows the state transition probability, which can be expressed as:

$$\Pr\{S_{t+1}|S_t, \pi(S_t)\} = \prod_{i=0}^{N} \Pr\{S_{t+1}^i|S_t^i, \pi(S_t^i)\}, \tag{26}$$

where S_t^i is the state of ED i in decision epoch t.

The task offloading problem can be regarded as an MDP. Considering the offloading policy π, $V(S,\pi)$ denotes the expected reward for following policy π in state S, and $Q(S,A)$ is the expected reward for selecting action A in state S and then following policy π. $V(S,\pi)$ and $Q(S,A)$ are considered as the state value and state–action value functions, respectively. Taking expectation with respect to system reward R at each epoch over a time sequence $\{t|(t \in \mathbf{N}^+)\}$, the expected long-term cumulative reward can be characterized as:

$$\mathrm{V}(S,\pi) = E_\pi \left[(1-\gamma) \cdot \sum_{t=1}^{\infty} (\gamma)^{t-1} \cdot R(S_t, \pi(S_t)) \right], \quad (27)$$

where S_t $(t \in \mathbf{N}^+)$ is the system state at epoch t, $(\gamma)^{t-1}$ is the discount factor, and $R(S_t, \pi(S_t))$ denotes the total system reward for taking action $\pi(S_t)$ under state S_t. Decomposed into the Bellman equation, the maximum state value achievable by policy π for state S is:

$$\mathrm{V}^*(S,\pi) = \max_A \sum_{S'} \Pr\{S'|S,A\} \cdot (R(S,A) + \gamma \cdot \mathrm{V}^*(S',\pi)), \quad (28)$$

where S' is the next state of S after choosing action A. In contrast, similar to Equation (28), the optimal state–action value function can be defined as:

$$Q^*(S,A) = \sum_{S'} \Pr\{S'|S,A\} \cdot (R(S,A) + \gamma \cdot \max_{A'} Q^*(S',A')), \quad (29)$$

where S' is the next state of S after choosing action A, and A' is the offloading action performed under state S'. Equations (28) and (29) are the Bellman optimality equations. In Q-learning, the policy π is updated using value iteration in a recursive way based on the observation of system state S.

The prerequisite for using the critic network in the selection of AGA is the convergence of the Q-function.

Theorem 1. *The critic network of the DRGO algorithm converges to the optimal Q-function.*

Proof. Because the state space $\mathbf{S}$ and action space $\mathbf{A}$ are finite, the state transition probabilities $\Pr\{S_{t+1}|S_t, \pi(S_t)\}$ in Equation (26) are stationary, and all state–action pairs $\{(S_t, A_t)|t \in N^+\}$ can be visited infinitely often. Given (S_t, A_t, S_{t+1}, R_t), the update rule of the critic network in the DRGO algorithm therefore is:

$$Q(S_t,A_t) = Q(S_t,A_t) + \alpha(S_t,A_t)[R_t + \gamma \max_{b\in\mathbf{A}} Q(S_{t+1},b) - Q(S_t,A_t)]. \quad (30)$$

We subtract $Q^*(S_t,A_t)$ from both sides, thus obtaining

$$\Delta(S_t,A_t) = Q(S_t,A_t) - Q^*(S_t,A_t), \quad (31)$$

which yields

$$\Delta(S_t,A_t) = (1-\alpha)\Delta(S_t,A_t) + \alpha(S_t,A_t)F(S_t,A_t), \quad (32)$$

$$F(S_t,A_t) = [R_t + \gamma \max_{b\in\mathbf{A}} Q(S_{t+1},b) - Q^*(S_t,A_t)]. \quad (33)$$

Because $\sum_{t=1}^{\infty} \alpha$ is infinite and $\sum_{t=1}^{\infty} \alpha^2$ is finite, according to [36], $\Delta(S_t,A_t)$ converges to zero w.p.1 if:

1) $\|\mathbb{E}[F(S_t,A_t)|F]\|_\infty \leq \gamma \|\Delta(S_t,A_t)\|_\infty$, with $\gamma < 1$.
2) $\mathrm{var}[F(S_t,A_t)|F] \leq C(1 + \|\Delta(S_t,A_t)\|_\infty^2)$, with $C > 0$

First, the following equation is derived.

$$\begin{aligned} \|\mathbb{E}[F(S_t,A_t)|F]\|_\infty &= \Pr(S_{t+1}|S_t,A_t)F(S_t,A_t) \\ &\leq \gamma \|Q(S_t,A_t) - Q^*(S_t,A_t)\|_\infty \\ &= \|\Delta(S_t,A_t)\|_\infty. \end{aligned} \quad (34)$$

Then, the following equation is obtained.

$$\mathrm{var}[F(S_t,A_t)|F] = \mathrm{var}[R_t + \gamma \max_{b\in\mathbf{A}} Q(S_{t+1},b)|F]. \quad (35)$$

Because R_t is bounded, the following is true.

$$\mathrm{var}[F(S_t,A_t)|F] \leq C(1 + \|\Delta(S_t,A_t)\|_\infty^2). \quad (36)$$

Here, C is a constant. Hence, $\Delta(S_t,A_t)$ converges to zero w.p.1, which means the critic network of the DRGO algorithm converges to the optimal Q-function $Q^*(S,A)$. □

Therefore, we can use the critic network to estimate the performance of each action in the candidate solution set. We denote the optimal action A^* as:

$$A^* = arg \max_{A_i \in \mathbf{A}} Q^*(S_t, A_i). \quad (37)$$

D. Self-Adjusting K in AGA

In AGA, with a larger K, we can reach a better offloading decision and achieve higher performance. However, a larger K also leads to computational complexity and large time consumption. Therefore, to balance the performance and complexity, we propose an algorithm with self-adjusting K.

The rule of adjusting K is simple. After obtaining the optimal action A^* from the candidate solution set, we compare it with the original output of the actor network A. If A is equal to A^*, then subtract one from K; otherwise, we increase the value of K based on the difference between A and A^*. The difference can be measured by the L2-norm of $A - A^*$, which is denoted as $\|A - A^*\|_2$.

In addition, to avoid K becoming too large, making AGA consume too much time, we add a constraint $K \leq K_{\max}$ to limit the maximum value of K. Given a strictly monotonically increasing functions ϕ, the updating rule of K is:

$$K = \begin{cases} \max(0, K-1) & A^* = A \\ \min\left(K + \phi(\|A - A^*\|_2), K_{\max}\right) & A^* \neq A. \end{cases} \quad (38)$$

E. Updating Policy with Prioritized Experience Replay

The DRGO algorithm regularly updates the parameters in its deep neural networks based on the experience. It maintains a replay memory with the finite size of x to store historical experiences, which can be represented as $\mathcal{M} = \{M_1, M_2, ..., M_x\}$, where $M_i = (S_t, A_t, R_t, S_{t+1})$ (Algorithm 1: line 9). Then,

Algorithm 1 Online Deep Reinforcement Learning Framework with AGA in the Task Offloading System

1: **Initialize** parameters of the online actor network and critic network with weights θ^{μ} and θ^{Q} randomly.
2: **Initialize** parameters of the target actor network and critic network with weights $\theta^{\mu'} \leftarrow \theta^{\mu}$ and $\theta^{Q'} \leftarrow \theta^{Q}$.
3: **Initialize** the prioritized replay memory $\mathcal{M}$ with the size of x.
4: **Set** the max number of generations $K_{\max}$, the training interval δ, the step count t, and φ used to measure the loss value of the critic network.
5: **Repeat**
6: At the beginning of decision epoch t, JM takes system state S_t as an input to the actor network and obtains action A_t.
7: With probability $1-\epsilon$ output action A_t. Otherwise, if $L(\theta^{Q'}) \leq \varphi$, use AGA to replace A_t with the optimal action A^* of a generated solution set $\mathbb{A}$; or replace A_t with a random action if $L(\theta^{Q'}) > \varphi$.
8: Execute action A_t, observe reward R_t, and observe new state S_{t+1}
9: Store transition (S_t, A_t, R_t, S_{t+1}) in replay memory.
10: **if** t mod δ = 0 **then**
11: Select samples from replay memory and update θ^{μ} and θ^{Q} using the Adam algorithm by minimizing the loss function in Equation (40) and (41).
12: **end if**
13: Regularly update the target networks:
14:
$$\theta^{Q'} \leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q'},$$
$$\theta^{\mu'} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'}.$$
15: **Until** A predefined stopping condition is satisfied.

in the training phase, we sample a mini-batch of transitions from the replay memory to update parameters in the actor network and the critic network toward maximizing the long-term reward.

Traditional deep reinforcement learning randomly selects a batch of training samples from the replay memory, making it difficult to learn valuable experience. Therefore, we organize the replay memory with the SumTree structure. It arranges samples in the replay memory according to their priorities, which is called prioritized experience replay [37]. As the loss value of a sample increases, it becomes more likely to be used to update the network. By selecting samples from replay memory with higher priority, our model can learn more efficiently, and the probability of the selecting sample i is:

$$P(i) = \frac{p_i^{\beta}}{\sum_k p_k^{\beta}}, \tag{39}$$

where $p_i > 0$ is the priority of sample i, and the exponent β denotes how much prioritization is used.

For the actor network, we denote samples selected from prioritized replay memory as $\mathcal{M}_{\mathcal{T}} = \{(S_t, A_t) | t \in \mathcal{T}\}$, where $\mathcal{T}$ represents the set of time indices. Using Adam optimizer, the parameters θ^{μ} of the actor network are updated in the direction of minimizing the cost function $L(\theta^{\mu})$ (Algorithm 1: lines 13-14). As mentioned, we represent actions in binary form. Therefore, the loss function $L(\theta^{\mu})$ is defined by:

$$L(\theta^{\mu}) = E_{\mathcal{M}_{\mathcal{T}}}\left[{A_t}^T log\mu(S_t) + (1-A_t)^T log(1-\mu(S_t))\right], \tag{40}$$

where $\mu(S_t)$ denotes the output of the current actor network with input S_t. The loss function $L(\theta^{\mu})$ is the averaged cross-entropy loss of the selected memory set.

For the critic network, we denote samples selected from prioritized replay memory as $\mathcal{M}'_{\mathcal{T}} = \{(S_t, A_t, R_t, S'_t) | t \in \mathcal{T}\}$. Similarly, we denote the loss function $L(\theta^{Q})$ as follows.

$$L(\theta^{Q}) = E_{\mathcal{M}'_{\mathcal{T}}}\left(R_t + \gamma \max_{A'_t} Q(S'_t, A'_t) - Q(S_t, A_t)\right)^2, \tag{41}$$

where γ is the discount value.

VI. SIMULATION

In this section, we evaluate the performance of our proposed algorithm through numerical simulations.

A. Simulation Setup

We consider a scenario where EDs are randomly distributed within an area of 500 m $\times$ 500 m. In addition, EDs are connected through a multi-hop ad-hoc network based on the WirelessHART protocol over IEEE 802.15.4 [38]. EDs can communicate with each other within the range of 100 m, so they are divided into different levels according to their distances from the JM. The channel bandwidths between EDs are 2.4 GHz. The transmit powers of EDs are 0.5 W [39], while the additive white Gaussian noise power is –120 dBm. The path loss exponent is -2. The maximum achievable rate (in bps) of r_i for downloading/uploading is set as 250 kbps [40]. For the interference between EDs, we use the sum of the transmission powers from all the interfering EDs as the interference strength [41]. In addition, we study the surveillance and security systems for a smart home in this simulation. Consequently, according to [38], the task's uploading data sizes follow the uniform distribution on [100, 1000] KB, while the transmission of the downloading data is ignored because the amount of returning data is usually small.

For the task execution, the required CPU cycles of data processing tasks l_i follow the uniform distribution on [20, 40] Gcycles [3]. And the CPU frequencies of ED f_i range from 10 GHz to 20 GHz. For simplicity, there are five different VM level for data processing tasks in the servers, whose CPU frequencies are 50 GHz, 100 GHz, 150 GHz, 200 GHz, and 250 GHz, respectively. Then, the prices for these five different VM levels are 1×10^{-6} tokens/Gcycle, 2×10^{-6} tokens/Gcycle, 3×10^{-6} tokens/Gcycle, 4×10^{-6} tokens/Gcycle, and 5×10^{-6} tokens/Gcycle, respectively. The arrival rate of data processing tasks is 0.6. The task queue lengths of EDs and the servers are 2 tasks and 10 tasks, respectively. The deadline of each data

processing task d_i follows the uniform distribution on [T, 5T], where T is the decision period. To compute the transmit cost, we set each energy unit corresponds to 2×10^{-3} joules [42]. To unitize the cost, we set that each joule costs 1×10^{-4} tokens [43]. The rewards for successfully finishing tasks follow the uniform distribution on [1×10^{-6}, 1×10^{-5}] tokens.

For the mining tasks, we set the parameters in the simulations according to [31]. The data size of the block to be mined is $[5, 10]$ Kb [16]. Similar to data processing tasks, there are five different VM levels for mining tasks, whose hash powers are 20 MHash/s, 40 MHash/s, 60 MHash/s, 80 MHash/s, and 100 MHash/s, respectively [34]. In addition, the prices are 2×10^{-5} tokens, 4×10^{-5} tokens, 6×10^{-5} tokens, 8×10^{-5} tokens, 1×10^{-4} tokens respectively. The hash power of the blockchain network follows a uniform distribution on [1×10^{3}, 1×10^{5}] GHash/s. In addition, the miner that first solves the complex mathematical problem and achieves consensus is rewarded with $R = 30$ tokens.

For the design of the DNN, we use two hidden layers consisting of 200 neurons and 100 neurons. The activation functions of hidden layers are Rectified Linear Unit (ReLU). In addition, as the action uses binary encoding, the activation functions of output layers in actor networks are Sigmoid functions. We set the size of the replay memory as 10240, the batch size as 128, and the training interval as 10.

For the parameters of AGA, the initial probability of crossover p_c is 0.8, and the initial mutation rate p_m is 0.01. We assign k_2 and k_4 a value of 0.5 and assign k_1 and k_3 a value of 0.8 in Equation (24) and (25). We set the initial value of K as 10 and $K_{\max}$ as 100. During each successive generation, AGA uses the roulette-wheel selection as the fitness selection because of its similarity [44]. However, it is worth noting that there is no universal selection method for all problems. Thus, numerical experiments are required to obtain the best selection methods.

B. Simulation Design

1) Evaluation Metrics:

- Average cost: This can be calculated by the ratio of system cost to the number of EDs in the system.
- Task drop rate: This is the percentage of the failed data processing tasks, which is used to evaluate the decision of resource allocation.
- Average transmit time: This is the average time that spent on uploading the required data and program code of the processing tasks. We can use it to evaluate the degree of the congestion in transmission links.

2) Baseline Algorithm:

- Greedy Algorithm: This algorithm randomly generates 10^7 actions and selects the best one.
- Genetic Algorithm: Genetic algorithms are commonly used to search high-quality solutions for problems with large search space. We set the size of the population as 200, the crossover probability as 0.7, the mutation probability as 0.05, and the maximum generation as 10000.
- DDPG: DDPG is a state-of-art DRL method that has successfully solved many challenging problems across a variety of domains with large action spaces [30].

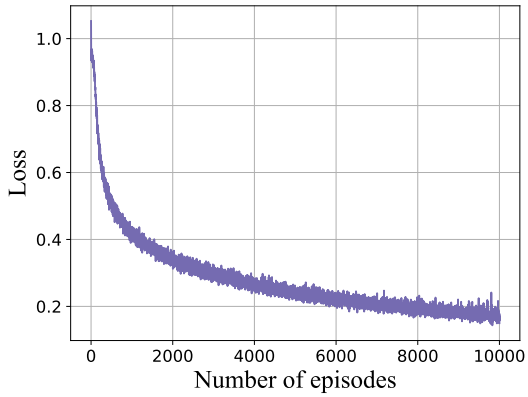


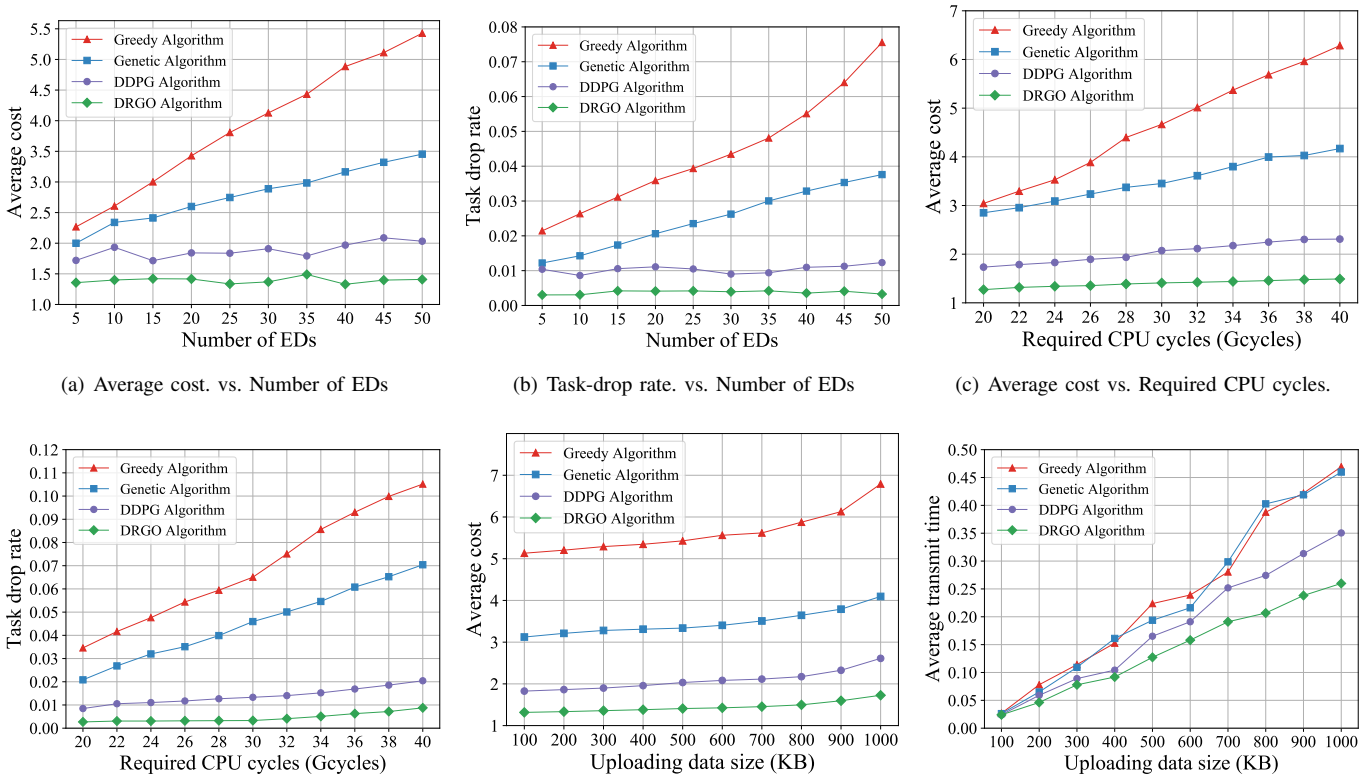
Fig. 4. The convergence property of the DRGO algorithm.

C. Convergence Performance

In this simulation, we evaluate the convergence property of the DRGO algorithm using the above parameter settings. The convergence of the DRGO algorithm is the crucial property to obtain a policy which maps states to the optimal actions. Generally, a neural network is considered to converge when the learning curve becomes flat. Therefore, we plot the training loss $L(\theta^{\mu})$ in each training episode under the scenario where the number of EDs is 50. The results in Fig. 4 reveal the convergence behaviors of our algorithm, which show that the DRGO algorithm converges after 10^5 episodes. Therefore, the DRGO algorithm converges at an acceptable speed.

D. Performance Analysis

1) Performance under different numbers of EDs: First, we study the performance of our algorithm under different numbers of EDs. Fig. 5 (a) plots the average long-term cost under scenarios where the numbers of EDs ranging from 5 to 50. The figure shows that our algorithm outperforms the other three benchmark policies. In particular, the average costs of the greedy algorithm and the genetic algorithm increase relatively fast as the number of EDs increases, while the average cost of the DRGO algorithm shows no clear sign of increasing. This is because the increasing number of EDs leads to a larger search space. And in Fig. 5 (b), we plot the task drop rate over different number of EDs, which shows the same pattern as Fig. 5 (a). This is because the increase of EDs inevitably increases the data transmitted in the transmission links. In this context, it takes more time on data transmission when ED chooses to offload tasks to the server, which may cause execution failure. Even so, the DRGO algorithm has the lowest task drop rate and shows no sign of increasing because it can schedule the routing path to reduce (or even avoid) data congestion.



(a) Average cost. vs. Number of EDs

(b) Task-drop rate. vs. Number of EDs

(c) Average cost vs. Required CPU cycles.

(d) Task-drop rate vs. Required CPU cycles.

(e) Average cost vs. Uploading data size.

(f) Average transmit time vs. Upload data size.

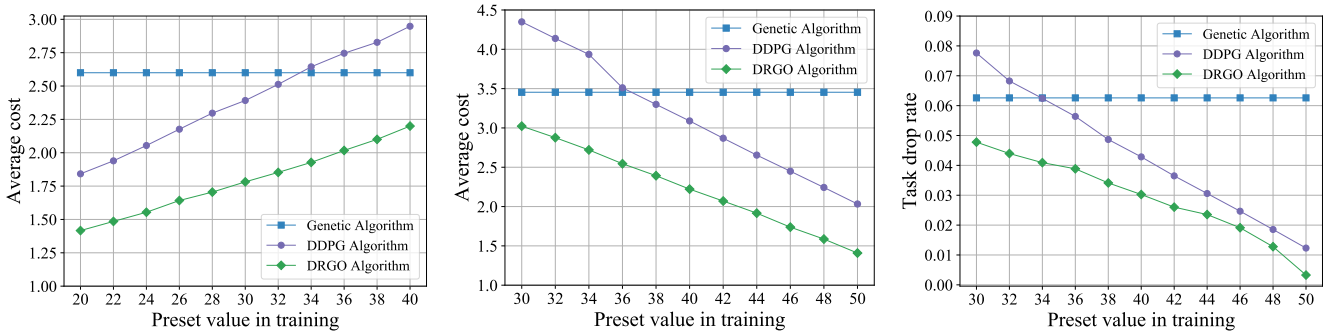
Fig. 5. Simulation results under: (a)-(b) different numbers of EDs; (c)-(d) different CPU cycles; (e)-(f): different uploading data sizes.

2) Performance under different CPU cycles: Next, we consider the impact of different CPU cycles required to execute the tasks. In this simulation, the required CPU cycles l_i of the data processing tasks range from 20 Gcycles to 40 Gcycles. In Fig. 5 (c), we plot the average cost of all four algorithms. Because larger CPU cycles often require more energy to finish tasks, the average long-term cost of all four algorithms increases. However, the average cost of the DRGO algorithm increases relatively slowly compared with three baseline algorithms, indicating that the DRGO algorithm can make intelligent offloading decisions to reduce the execution cost. In addition, Fig. 5 (d) plots the task drop rate, which shows that the DRGO algorithm achieves the lowest task drop rate. This is because the intelligent decisions about tasks executions decrease the execution delay.

3) Performance under different data sizes: In this simulation, we consider how the size of the required data during execution impacts the performance. We record the average cost of the system for various data sizes ranging from 100 KB to 1000 KB. In Fig. 5 (e), we plot the average long-term cost under different data sizes. The figure shows that the average long-term cost of the DRGO algorithm increases much more slowly compared with the greedy algorithm and the genetic algorithms. There are two reasons for this difference. On the one hand, a larger data size requires more energy on transmission. The DRGO algorithm can arrange a data transmission path that avoids data congestion and unnecessary cost. To prove it, we plot the average transmit time under different data size in Fig. 5 (f). We can see that the DRGO algorithm has the shortest transmit time. In addition, it is worth noting that the gap between the DRGO algorithm and the other three algorithms grows steadily, indicating that the DRGO algorithm brings greater performance gains under scenarios with a larger size of transmission data. On the other hand, a larger data size requires more space on memory during queuing. However, the memory space is not infinite in our considered scenario. Therefore, a poor offloading policy faces a task-failure penalty when if is not enough space for new tasks.

E. Scalability Analysis

In general, the dimension of a DNN input is an invariant value, while the adding and exiting of EDs are common for MEC. To support that, we regard the remaining ED states in the DNN as empty states if the number of EDs is smaller than the preset training value. For example, given the preset training value V_t and the actual number of EDs V_a ($V_a < V_t$), we set the remaining $V_t - V_a$ states to be 0 and drop the remaining $V_t - V_a$ actions in the output of the DRGO algorithm. On the other hand, if $V_a > V_t$, we divide the collected EDs' states into several parts and input these parts to the actor network in turn. Therefore, in this simulation, our goal is to validate the scalability of the DRGO algorithm, which is the performance of the DRGO algorithm under scenarios where the number of EDs is larger or smaller than the preset value in training.



(a) Average cost vs. Preset value in training. (b) Average reward vs. Preset value in training. (c) Task-drop rate vs. Preset value in training.

Fig. 6. Scalability Analysis.

First, we apply the DRGO algorithm to environments with fewer EDs than the preset value used in the training phase. And we set the number of EDs in the environment as 20. As shown in Fig. 6 (a), the preset values of EDs in the DRGO algorithm and DDPG algorithm range from 20 to 40 and the baseline algorithm is the genetic algorithm, which achieves higher performance than the greedy algorithm in the previous simulations. We can see that although larger preset training values increase the cost, the average costs of the DRGO algorithm are still lower than the greedy algorithm.

Next, we apply the DRGO algorithm to environments with more EDs than the preset value. In this simulation, we set the number of EDs as 50 and plot the average cost in Fig. 6 (b). The figure shows that although the DDPG algorithm suffers from higher cost when the preset value in training is lower than 36, the DRGO algorithm still performs well and reaches a lower cost than the genetic algorithm. In Fig. 6 (c), we go one step further and plot the task-drop rate of all three algorithms under the same environment setting. The figure shows that the task-drop rate of the DRGO algorithm remains the smallest. These simulation results indicate that our proposed DRGO algorithm can quickly adapt to environments where the number of EDs is larger or smaller than the preset values in the training phase.

VII. CONCLUSION

In this paper, we propose a deep reinforcement learning-based online computation offloading approach for blockchain-empowered MEC, in which both mining tasks and data processing tasks are considered. To achieve long-term offloading performance, our DRGO algorithm uses model-free deep reinforcement learning to adapt to highly dynamic environments and maximize the long-term reward. In particular, to overcome the slow convergence caused by the high-dimensional action space, the DRGO algorithm takes advantage of the genetic algorithms into deep reinforcement learning during the exploration process, survives the curse of high-dimensional action space, and converges at a acceptable speed. Simulation results have shown that the DRGO algorithm achieves better performance compared with three representative benchmark policies and shows strong robustness under various environments. In future work, we plan to study the exploration mechanism to further improve the efficiency of the exploration, which is the key to solve the convergence problem of deep reinforcement learning.

REFERENCES

[1] Z. Hong, H. Huang, S. Guo, W. Chen, and Z. Zheng. Qos-aware cooperative computation offloading for robot swarms in cloud robotics. *IEEE Transactions on Vehicular Technology*, 68(4):4027–4041, April 2019.

[2] J. Kang, R. Yu, X. Huang, M. Wu, S. Maharjan, S. Xie, and Y. Zhang. Blockchain for secure and efficient data sharing in vehicular edge computing and networks. *IEEE Internet of Things Journal*, pages 1–1, 2019.

[3] H. Guo, J. Liu, J. Zhang, W. Sun, and N. Kato. Mobile-edge computation offloading for ultradense iot networks. *IEEE Internet of Things Journal*, 5(6):4977–4988, Dec 2018.

[4] Z. Li, Z. Yang, S. Xie, W. Chen, and K. Liu. Credit-based payments for fast computing resource trading in edge-assisted internet of things. *IEEE Internet of Things Journal*, pages 1–1, 2019.

[5] Z. Li, Z. Yang, and S. Xie. Computing resource trading for edge-cloud-assisted internet of things. *IEEE Transactions on Industrial Informatics*, pages 1–1, 2019.

[6] W. Chen, Z. Zhang, Z. Hong, C. Chen, J. Wu, S. Maharjan, Z. Zheng, and Y. Zhang. Cooperative and distributed computation offloading for blockchain empowered industrial internet of things. *IEEE Internet of Things Journal*, pages 1–1, 2019.

[7] J. Kang, Z. Xiong, D. Niyato, P. Wang, D. Ye, and D. I. Kim. Incentivizing consensus propagation in proof-of-stake based consortium blockchain networks. *IEEE Wireless Communications Letters*, 8(1):157–160, Feb 2019.

[8] Z. Xiong, S. Feng, W. Wang, D. Niyato, P. Wang, and Z. Han. Cloud/fog computing resource management and pricing for blockchain networks. *IEEE Internet of Things Journal*, pages 1–1, 2019.

[9] Y. Wang, K. Wang, H. Huang, T. Miyazaki, and S. Guo. Traffic and computation co-offloading with reinforcement learning in fog computing for industrial applications. *IEEE Transactions on Industrial Informatics*, 15(2):976–986, Feb 2019.

[10] V. Mnih, K. Kavukcuoglu, D. Silver, A. Rusu, J. Veness, M. Bellemare, A. Graves, M. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015.

[11] H. Huang, J. Yang, H. Huang, Y. Song, and G. Gui. Deep learning for super-resolution channel estimation and doa estimation based massive mimo system. *IEEE Transactions on Vehicular Technology*, 67(9):8549–8560, Sep. 2018.

[12] Y. Kawamoto, H. Takagi, H. Nishiyama, and N. Kato. Efficient resource allocation utilizing q-learning in multiple ua communications. *IEEE Transactions on Network Science and Engineering*, pages 1–1, 2018.

[13] Y. Zhang, J. Yao, and H. Guan. Intelligent cloud resource management with deep reinforcement learning. *IEEE Cloud Computing*, 4(6):60–69, November 2017.

[14] M. Srinivas and L. M. Patnaik. Adaptive probabilities of crossover and mutation in genetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, 24(4):656–667, April 1994.

[15] W. Chen, B. Liu, H. Huang, S. Guo, and Z. Zheng. When uav swarm meets edge-cloud computing: The qos perspective. *IEEE Network*, 33(2):36–43, March 2019.

[16] M. Liu, F. R. Yu, Y. Teng, V. C. M. Leung, and M. Song. Computation offloading and content caching in wireless blockchain networks with mobile edge computing. *IEEE Transactions on Vehicular Technology*, 67(11):11008–11021, Nov 2018.

[17] W. Chen, Y. Yaguchi, K. Naruse, Y. Watanobe, and K. Nakamura. Qos-aware robotic streaming workflow allocation in cloud robotics systems. *IEEE Transactions on Services Computing*, pages 1–1, 2018.

[18] D. Chatzopoulos, M. Ahmadi, S. Kosta, and P. Hui. Flopcoin: A cryptocurrency for computation offloading. *IEEE Transactions on Mobile Computing*, 17(5):1062–1075, May 2018.

[19] Y. Wang, M. Liu, J. Yang, and G. Gui. Data-driven deep learning for automatic modulation recognition in cognitive radios. *IEEE Transactions on Vehicular Technology*, 68(4):4074–4077, April 2019.

[20] G. Gui, H. Huang, Y. Song, and H. Sari. Deep learning for an effective nonorthogonal multiple access scheme. *IEEE Transactions on Vehicular Technology*, 67(9):8440–8450, Sep. 2018.

[21] M. Liu, R. Yu, Y. Teng, V. Leung, and M. Song. Performance optimization for blockchain-enabled industrial internet of things (iiot) systems: A deep reinforcement learning approach. *IEEE Transactions on Industrial Informatics*, pages 1–1, 2019.

[22] X. He, K. Wang, H. Huang, T. Miyazaki, Y. Wang, and S. Guo. Green resource allocation based on deep reinforcement learning in content-centric iot. *IEEE Transactions on Emerging Topics in Computing*, pages 1–1, 2018.

[23] M. Liu, T. Song, and G. Gui. Deep cognitive perspective: Resource allocation for noma-based heterogeneous iot with imperfect sic. *IEEE Internet of Things Journal*, 6(2):2885–2894, April 2019.

[24] Q. Qi, J. Wang, Z. Ma, H. Sun, Y. Cao, L. Zhang, and J. Liao. Knowledge-driven service offloading decision for vehicular edge computing: A deep reinforcement learning approach. *IEEE Transactions on Vehicular Technology*, pages 1–1, 2019.

[25] M. Min, X. Wan, L. Xiao, Y. Chen, M. Xia, D. Wu, and H. Dai. Learning-based privacy-aware offloading for healthcare iot with energy harvesting. *IEEE Internet of Things Journal*, pages 1–1, 2019.

[26] M. Min, L. Xiao, Y. Chen, P. Cheng, D. Wu, and W. Zhuang. Learning-based computation offloading for iot devices with energy harvesting. *IEEE Transactions on Vehicular Technology*, 68(2):1930–1941, Feb 2019.

[27] Y. Wei, F. R. Yu, M. Song, and Z. Han. User scheduling and resource allocation in hetnets with hybrid energy supply: An actor-critic reinforcement learning approach. *IEEE Transactions on Wireless Communications*, 17(1):680–692, Jan 2018.

[28] H. Zhu, Y. Cao, X. Wei, W. Wang, T. Jiang, and S. Jin. Caching transient data for internet of things: A deep reinforcement learning approach. *IEEE Internet of Things Journal*, pages 1–1, 2019.

[29] V. Mnih, A.P. Badia, M. Mirza, A. Graves, T.P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *CoRR*, abs/1602.01783, 2016.

[30] T.P. Lillicrap, J.J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *CoRR*, abs/1509.02971, 2015.

[31] S. Biswas, K. Sharif, F. Li, B. Nour, and Y. Wang. A scalable blockchain framework for secure transactions in iot. *IEEE Internet of Things Journal*, pages 1–1, 2019.

[32] H. Liu, S. Liu, and K. Zheng. A reinforcement learning-based resource allocation scheme for cloud robotics. *IEEE Access*, 6:17215–17222, 2018.

[33] H. Nishiyama, M. Ito, and N. Kato. Relay-by-smartphone: realizing multihop device-to-device communications. *IEEE Communications Magazine*, 52(4):56–65, April 2014.

[34] A.P. Ozisik, G. Bissias, and B.N. Levine. Estimation of miner hash rates and consensus on blockchains (draft). *CoRR*, abs/1707.00082, 2017.

[35] M.A. Ferrag, M. Derdour, M. Mukherjee, A. Derhab, L.A. Maglaras, and H. Janicke. Blockchain technologies for the internet of things: Research issues and challenges. *CoRR*, abs/1806.09099, 2018.

[36] C.J.C.H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8(3):279–292, May 1992.

[37] T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized experience replay. *CoRR*, abs/1511.05952, 2016.

[38] T. Mendes, R. Godina, E. Rodrigues, J. Matias, and J. Catalão. Smart home communication technologies and applications: Wireless protocol assessment for home area network resources. 2015.

[39] J. Liu, N. Kato, J. Ma, and N. Kadowaki. Device-to-device communication in lte-advanced networks: A survey. *IEEE Communications Surveys Tutorials*, 17(4):1923–1940, Fourthquarter 2015.

[40] J. Benoit, A. Yao, L. Saladis, and Y. Zheng. Performance evaluations of multi-hop wirelesshart network and 6lowpan using different topologies. In *2018 Global Smart Industry Conference (GloSIC)*, pages 1–5, Nov 2018.

[41] C. Wang, F. R. Yu, C. Liang, Q. Chen, and L. Tang. Joint computation offloading and interference management in wireless cellular networks with mobile edge computing. *IEEE Transactions on Vehicular Technology*, 66(8):7432–7445, Aug 2017.

[42] F. Tang, Z. M. Fadlullah, N. Kato, F. Ono, and R. Miura. Ac-poca: Anticoordination game based partially overlapping channels assignment in combined uav and d2d-based networks. *IEEE Transactions on Vehicular Technology*, 67(2):1672–1683, Feb 2018.

[43] J. Liu, N. Kato, H. Ujikawa, and K. Suzuki. Device-to-device communication for mobile multimedia in emerging 5g networks. *ACM Trans. Multimedia Comput. Commun. Appl.*, 12(5s):75:1–75:20, September 2016.

[44] M. Madiafi K. Jebari. Selection methods for genetic algorithms. *International Journal of Emerging Sciences*, 3:333–344, 12 2013.