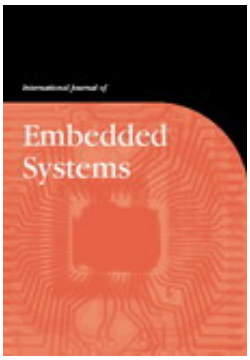


International Journal of Embedded Systems

ISSN online: 1741-1076 - ISSN print: 1741-1068
https://www.inderscience.com/ijes

Deep reinforcement learning-based collaborative computation offloading and caching decision for internet of things

Jianxin Li, Ke Yuan, Qian Wang, Siguang Chen

DOI: 10.1504/IJES.2024.10066673

Article History:

Received:	23 October 2022
Last revised:	04 November 2023
Accepted:	07 January 2024
Published online:	10 February 2025

Deep reinforcement learning-based collaborative computation offloading and caching decision for internet of things

Jianxin Li, Ke Yuan*, Qian Wang and Siguang Chen

School of Internet of Things,
Nanjing University of Posts and Telecommunications,
Nanjing 210000, China
Email: ljx19825089686@163.com
Email: keyuan98@126.com
Email: qian_wang96@163.com
Email: sgchen@njupt.edu.cn
*Corresponding author

Abstract: Currently, as an extension of cloud computing, edge computing has attracted much attention for its ability to reduce delay and energy and bandwidth consumption. For satisfying the demands of resource-intensive and delay-sensitive applications and solving the problems of existing computation offloading algorithms, such as inability to handle massive amounts of data in time and the need for strong computing capability, an intelligent computation offloading, resource allocation and collaborative caching scheme with joint optimisation of delay and energy consumption is proposed. Specifically, we formulate an optimisation problem of minimising the weighted sum of all tasks' completion time and energy consumption under the constraints of delay, bandwidth, computing capability, and energy. To solve the above mixed integer nonlinear programming problem (MINLP), we develop a deep reinforcement learning (DRL)-based collaborative computation offloading and caching decision (DRL-CCOC) algorithm. The algorithm jointly optimises offloading decisions, caching decisions, the occupation ratio of the wireless channel bandwidth and the edge server's computing capability which is allocated to the task. It can generate the optimal policy and adapt to dynamic network environment with the ability of autodidacticism. Finally, the simulation results demonstrate that the DRL-CCOC can converge at a faster rate and reduce the total cost significantly compared with other methods, they also confirm the strong dynamic adaptability of our algorithm.

Keywords: computing; computation offloading; caching; deep reinforcement learning; DRL; resource allocation.

Reference to this paper should be made as follows: Li, J., Yuan, K., Wang, Q. and Chen, S. (2024) 'Deep reinforcement learning-based collaborative computation offloading and caching decision for internet of things', *Int. J. Embedded Systems*, Vol. 17, Nos. 3/4, pp.161–170.

Biographical notes: Jianxin Li received his BE in Internet of Things Engineering from Nanjing University of Posts and Telecommunications in 2022. He is currently pursuing his Master's degree at Nanjing University of Posts and Telecommunications. His main research interests include computation offloading and deep learning.

Ke Yuan received her Master's degree at Nanjing University of Posts and Telecommunications. Her main research includes computation offloading and internet of things.

Qian Wang received her Master's degree at Nanjing University of Posts and Telecommunications. She is currently pursuing her Doctoral degree at Nanjing University of Posts and Telecommunications. Her main research interests are in the area of computation offloading and federated learning.

Siguang Chen received his PhD in Information Security from Nanjing University of Posts and Telecommunications. He is currently a Full Professor at Nanjing University of Posts and Telecommunications. His main research interests include computation offloading and edge intelligence.

1 Introduction

With the advancement of internet of things (IoT) and continuously increasing requirements of resource-intensive and delay-sensitive applications to the public, such as objective recognition and augmented reality (Ma et al., 2020). IoT devices play important roles in our daily life. However, faced with application requirements for data collection, processing, the local IoT devices cannot handle them on time because of resource constrained. To deal with these puzzles, cloud computing attracts attention. In cloud computing, the local devices transmit their tasks to the cloud server, where the cloud server uses its vast computing resource for computation, thus greatly making up the shortcoming of local computing. Nevertheless, the cloud server is usually far away from the devices, which resides in the core network (Ale et al., 2021), transmitting and downloading massive data to/from the remote cloud server cause heavy traffic jams in wireless channel and unpredicted service delay.

For coping with the issues in cloud computing, edge computing, as an extension of cloud computing, receives great interest (Lin et al., 2020; Jiang et al., 2019; Lin et al., 2019). Compared with cloud computing, edge computing is closer to the devices. Therefore, the devices can offload tasks to nearby the edge server (ES) rather than sending them to remote cloud server (Zhang et al., 2015). Accordingly, the delay and energy consumption caused by transmission can be reduced. Chen et al. (2019) proposed a dynamic computation offloading algorithm, based on stochastic optimisation, which decomposes the optimisation problem into a series of sub-problems and solves these sub-problems in an online and distributed way to minimise cost and queue length. Their experiments' result verifies the effectiveness of computation offloading in edge computing. In order to fully utilise computing resources, Zhao et al. in [8] studied the design of computation offloading in fog radio access network to minimise the total cost in respect to the energy consumption and service latency. Zhao et al. (2019) proposed a multi-objective computation offloading algorithm combining multi-objective evolutionary algorithm based on decomposition with invasive weed optimisation and differential evolution. In Chowdhury (2021) first investigated different challenges, requirements, and latency-sensitive applications of emerging IoT and presented a flexible heuristic-based prioritised resource assignment strategy for IoT application execution in the 5G era to ensure low-latency for heavy weight IoT application processing. Dinh et al. (2018) introduced a model-free reinforcement learning offloading mechanism which helps users learn their long-term offloading strategies to maximise their long-term utilities. There is also combined cloud and edge computing to enable applications to be executed with low latency (Ning et al., 2019). However, in these studies, they do not consider caching in ES, so that the same tasks are computed repeatedly leading to redundant delay and energy consumption.

To further improve the users' quality of experience, caching mechanisms are studied in edge computing in works (Simon et al., 2020; Hao et al., 2018; Yu et al., 2018; Ndikumana et al., 2017), such as caching the content or the result. Xu et al. (2018) investigated the extremely compelling but much less studied problem of dynamic service caching in mobile edge computing-enabled dense cellular networks and optimised dynamic service caching and task offloading. Usually, the limited caching space at resource-constrained ES needs caching placement to determine which tasks to cache. Dave and Kotak (2023) proposed to analyse cache memory parameters before choosing the best cache to lower power and cost. Bi et al. (2020) introduced a single ES that mobile users can upload and run their programs at the ES, while the server can selectively cache the previously generated programs for future reuse. Liu et al. (2018) proposed a novel mobile edge computing enabled wireless blockchain framework where the cryptographic hashes of blocks can be cached. As to ignore the impact of tasks caching on computation offloading at ES, which leads to a longer competition time, Zhao et al. (2020) formally defined the problem of offloading dependent tasks with service caching, and designed an efficient convex programming-based algorithm to solve this problem. Yang et al. (2020) formulated a long-term reward maximisation problem optimising the task offloading and caching decisions and computation resource allocation. To tackle this optimisation problem, they propose a single-agent Q-learning algorithm which is invoked to learn a long-term resource allocation strategy. Chen et al. (2018) introduced a new concept of computing task caching and gave the optimal computing task caching policy to optimise computing, caching, and communication in edge cloud. The simulation results show that their algorithm had shorter delay than other schemes. To further reduce the data transmission cost and job delay, a shareable cache to the ES is added in Wei et al. (2020), namely, the authors proposed a cache-aware computation offloading strategy for edge cloud computing and minimise the equivalent weighted response. Accordingly, they design an online computation offloading algorithm to optimise the problem. Due to the allocation of resource in fog computing, Lan et al. (2019) introduced two computation offloading scenarios, in the first scenario, a task caching algorithm based on genetic algorithm (GA) is used to optimise the problem. In the second scenario, the optimisation problem of offloading and resource allocation is expressed as a mixed integer nonlinear programming problem (MINLP) which is solved by a distributed algorithm with Lagrange multiplier.

Traditional algorithms cannot handle massive amounts of data in time and be applied in practical dynamic systems. As a result, deep learning (DL) is introduced into computation offloading collaborative caching. For example, Wang and Friderikos (2020a) proposed a framework to reduce the computational complexity of the underlying integer mathematical program by first predicting decision variables related to optimal locations using a deep convolutional neural network (CNN) and then CNN is trained offline to solve the optimisation problems. The work

of Lei et al. (2019) combines the predictions from CNN with the optimal branch and bound algorithm to reduce the feasible region of time slot allocation and improve the caching content delivery problem. Besides, there are many works studying caching and computation offloading with DL, such as Wang and Friderikos (2020b), that summarise the utilisation of DL for caching in edge network. By their analysis, caching and machine learning are effectively used in edge computing. However, the complex network structure of DL algorithms brings lots of hardware cost. What's more, DL algorithms need plenty of labelled data and extremely strong computing capability.

Inspired by the above challenges, we design an intelligent computation offloading, resource allocation and collaborative caching scheme. The key contributions are three-fold as follows:

- In order to alleviate the pressure of current huge user numbers and energy consumption, we formulate a mixed integer nonlinear programming optimisation problem to minimise the weighted sum of the total completion time and energy consumption of tasks under the constraints of delay, bandwidth, computing capability, and energy with relatively small hardware and computing cost.
- To solve the above optimisation problem, a deep reinforcement learning (DRL)-based collaborative computation offloading and caching decision (DRL-CCOC) algorithm is proposed. This algorithm jointly optimises offloading decisions, caching decisions, the occupation ratio of the wireless channel bandwidth and the ES's computing capability which is allocated to the task. It can obtain the optimal policy and adapt to dynamic network environment with the ability of autodidacticism.
- Compared with other benchmark schemes, the simulation results show that the proposed algorithm has the lowest cost under different system parameters, and it can converge at a faster rate.

The rest of this article is organised as follows: Section 2 describes the system model; in Section 3, an optimisation problem is formulated; Section 4 introduces the proposed DRL-CCOC algorithm in detail; Section 5 is the simulation results and discussions. Finally, Section 6 is conclusions.

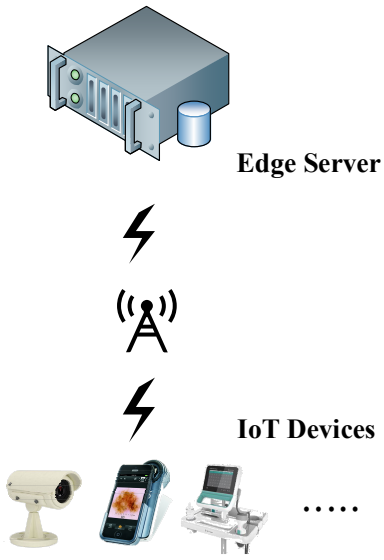
2 System model

In this section, we construct a computation offloading model with content caching in which ES and N IoT devices are considered in our system model. We define $i \in \{1, 2, 3, \ldots, N\}$ as the task of the device i generated. The size of task i and the number of CPU cycles required to compute for task i are w_i and c_i, respectively.

2.1 Network framework

We consider a two-layer architecture. At the bottom of this architecture is IoT devices where tasks will be generated at the beginning of a time slot with a certain probability. We design the specific processing of computing tasks as shown in Figure 1.

Figure 1 System mode (see online version for colours)



- IoT devices: IoT devices include cameras, dermoscopy, ultrasound machine, and so on. The users generate the task through IoT devices and tasks can be computed locally or offloaded to the ES for calculation through the adjacent base station (BS). Denote the task set as $\kappa = \{1, 2, \ldots, N\}$.
- ES: The ES plays a role with two main functions:
  1 the ES can receive the tasks offloaded from IoT devices through the BS and process them
  2 it can also cache the tasks to obtain less delay and energy consumption.

The ES identifies whether tasks generated by IoT devices are cached in the ES, computes them directly if they are; if there is no cache, caching decisions will be made. Meanwhile, the communication and computing resources of the ES will be allocated to tasks which are offloaded to the ES.

2.2 Caching and computing model

When the IoT device generates task i, it first makes the decision that the task is computed locally or offloaded to ES. The symbol $\phi = [\phi_1, \phi_2, \ldots, \phi_N]$ denotes the offloading decisions; the computation task will be executed at local if $\phi_i = 0$. When $\phi_i = 1$, the device needs check ES whether has cached task i. If it has cached, the device can hit cached task i and compute it at ES directly. Otherwise, we will offload task i to ES.

In system model, we assume that the caching decision for one computation task is defined as x_i. $x_i \in X$ and $X = [x_1, x_2, \ldots, x_N]$. If $x_i = 1$, it implies that ES has cached task i, i.e., the device hits the cached task at ES, the computation task will be computed directly. In our scenario, we also consider the allocation of communication and computing resources. Let σ_i as the wireless channel bandwidth occupation ratio of task i between devices and ES. ψ_i represents the computing resource occupation ratio which is allocated to task i. The ranges of σ_i and ψ_i are both [0, 1].

The aim of optimisation is to joint optimisation of offloading decisions, caching decisions, communication and computing resources to reduce the total cost of the tasks. The total cost includes delay and energy consumption.

- Local computing: When task i was computed at local, i.e., $\phi_i = 0$, $\sigma_i = 0$ and $\psi_i = 0$. Without loss of generality, delay of task i at local can be expressed as

$$t_i^{loc} = \frac{c_i}{f_i^{loc}}, \tag{1}$$

where represents the computing capability of device i and the energy consumption of processing this task is given by

$$e_i^{loc} = P_i t_i^{loc} = P_i \frac{c_i}{f_i^{loc}}, \tag{2}$$

where represents the computing power of device i and the local computing capability varies from device to device. There is no delay and energy consumption for tasks' transmission.

Combining delay (1) and energy consumption (2) so that the total cost of task i computing locally can be given as

$$C_{loc} = \partial t_i^{loc} + (1-\partial) e_i^{loc} \tag{3}$$

where ∂ represents the weight coefficient of delay. To increase the robustness and dynamic adaptability of the model, we set a constrain for so that tasks completion time and energy consumption weights can be changed depending on IoT devices' demands and the rang ∂ of is [0, 1].

- Computation offloading with cached task i: We set $\phi_i = 1$ as the offloading decision of task i. Device i will check ES whether has the cached task i. If the task cached that is $x_i = 1$, device i will hit the cached task and doesn't have to offload the task and this task can be computed at ES directly. Delay and energy consumption for computing task i can be denoted respectively as

$$t_i^e = \frac{c_i}{\psi_i f_i^e}, \tag{4}$$

and

$$e_i^e = P_e t_i^e = P_e \frac{c_i}{\psi_i f_i^e}, \tag{5}$$

where P_e is the computation power of the ES, f_i^e represents the computing capability of ES. According to equations (4) and (5), we could compute the total cost of the computation offloading with cached tasks i including delay and energy consumption as

$$C_{cached} = \partial t_i^e + (1-\partial) e_i^e \tag{6}$$

- Computation offloading without cached task i: Accordingly, if task i is not cached, we will upload task i from device i to ES through wireless access. In this case, $\phi_i = 1$ and $x_i = 0$. The device does not hit the task cache and the task can only be offloaded to ES. Therefore, delay can be expressed as

$$t_i^{e\_off} = t_i^{up} + t_i^e, \tag{7}$$

and the energy consumption is

$$e_i^{e\_off} = e_i^{up} + e_i^e, \tag{8}$$

where t_i^{up} represents the time taken to offload task i to ES. Similarly, e_i^{up} is the energy consumption by offloading task i. They are

$$t_i^{up} = \frac{w_i}{R_u^i} = \frac{w_i}{\sigma_i B \log_2\left(1 + \frac{P_{up} h^2}{g_o \sigma_i B}\right)}, \tag{9}$$

where R_u represents the transmission speed of wireless channel, the bandwidth of wireless channel is defined as B, the transmission power of IoT devices is defined as P_{up}, the wireless channel gain between task i and ES is defined as h and the spectral density of the channel noise power is defined as g_o, and

$$e_i^{up} = P_{up} t_i^{up} = P_{up} \frac{w_i}{R_u} = P_{up} \frac{w_i}{\sigma_i B \log_2\left(1 + \frac{P_{up} h^2}{g_o \sigma_i B}\right)} \tag{10}$$

Therefore, according to equations (4), (7) and (9), the total delay for task i is

$$t_i^{e\_off} = \frac{w_i}{\sigma_i B \log_2\left(1 + \frac{P_{up} h^2}{g_o \sigma_i B}\right)} + \frac{c_i}{\psi_i f_i^e} \tag{11}$$

and according to equations (5), (8) and (10), the energy consumption of offloading task i is

$$e_i^{e\_off} = P_{up} \frac{w_i}{\sigma_i B \log_2\left(1 + \frac{P_{up} h^2}{g_o \sigma_i B}\right)} + P_e \frac{c_i}{\psi_i f_i^e} \tag{12}$$

Combining equations (11) and (12), we can give the total cost of offloading task without cache hit as

$$C_{e\_off} = \partial t_i^{e\_off} + (1-\partial) e_i^{e\_off} \tag{13}$$

Li et al. (2018) and Chen (2015) they do not fundamentally consider delay and energy consumption for data downlink because the cost of transmitting and computing tasks are much more than downlink. Therefore, we do not take the cost of tasks downlink into consideration as well.

Through above analysis, we define delay and energy consumption for task i are T_i and E_i respectively. They are

$$T_i = \begin{cases} t_i^{loc}, & \phi_i = 0 \\ t_i^{e}, & \phi_i = 1 \quad and \quad x_i = 1 \\ t_i^{e\_off}, & \phi_i = 1 \quad and \quad x_i = 0 \end{cases} \tag{14}$$

and

$$E_i = \begin{cases} e_i^{loc}, & \phi_i = 0 \\ e_i^{e}, & \phi_i = 1 \quad and \quad x_i = 1 \\ e_i^{e\_off}, & \phi_i = 1 \quad and \quad x_i = 0 \end{cases} \tag{15}$$

3 Problem formulation

In this section, we formulate an optimisation problem. The objective is to minimise the weighted sum of the total completion time and energy consumption of the task set κ. The weighted sum cost function for the task set κ can be expressed as

$$C(\phi, x_i, \sigma_i, \psi_i) = \partial T + (1-\partial) E \tag{16}$$

We define the total task set delay as

$$T = \max[T_1, T_2, ..., T_N] \tag{17}$$

and the total task set energy consumption as

$$E = \sum_{i=1}^{N} E_i = \sum_{i=1}^{N} (1-\phi) e_i^{loc} + \phi \left(x_i e_i^{e} + (1-x_i) e_i^{e\_off} \right) \tag{18}$$

Through the above system model and associated definitions, we can formulate the following optimisation problem by jointly optimising the offloading decision ϕ_i, caching decision x_i, the wireless channel bandwidth occupation ratio θ_i and the computing resource occupation ratio ψ_i with a set of constraints. The specific optimisation problem is given as follows:

$$\text{P:} \quad \min_{\phi, x_i, \sigma_i, \psi_i} \sum_{i=1}^{N} \partial T + (1-\partial) E \tag{19}$$

$$s.t. \quad \sum_{i=1}^{N} \sigma_i \le 1 \tag{19a}$$

$$\sum_{i=1}^{N} \psi_i \le 1 \tag{19b}$$

$$T \le T^{tolerate} \tag{19c}$$

$$E \le E^{tolerate} \tag{19d}$$

$$x_i, \phi \in \{0,1\} \tag{19e}$$

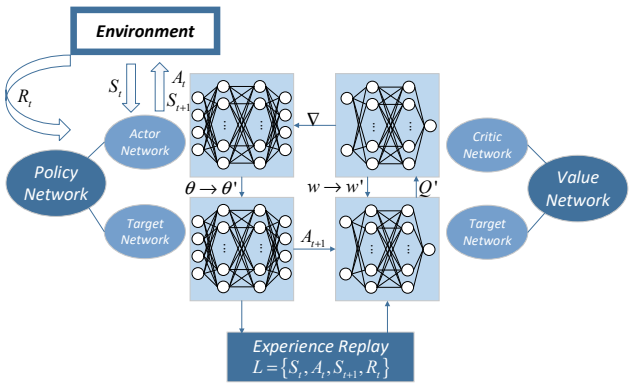
$$\sigma_i, \psi_i \in [0,1] \tag{19f}$$

In (19), the objective function implies the minimisation of the weighted sum of completion and energy consumption for the task set κ. Equations (19a) and (19b) ensure that occupation ratios of communication and computing resources of ES do not exceed unity. Equations (19c) and constraint (19d) represent that delay and energy consumption of completing all tasks should not exceed the tolerable values. Equation (19e) demonstrates the value of caching decision and offloading decision is limited to 0 or 1. When they are 0, task i is not cached and computed at local. Equation (19f) indicates the range of the weight coefficient ∂, σ_i and ψ_i. When θ_i and ψ_i are equal to 1, the meaning is that task i is cached and the task is offloaded to ES. Our computation offloading algorithm is based on DRL. We will explain in more detail below.

4 Deep reinforcement learning-based collaborative computation offloading and caching decision

The above non-convex optimisation problem P1 is MINLP, which is a non-deterministic polynomial hard (NP-hard) problem as well. Due to the demands of dynamic adaptability, it has difficulty in solving this problem with the conventional algorithm. Thus, we propose a DRL-CCOC algorithm to solve the weighted sum of the total completion time and energy consumption of the task set, i.e., to get the minimum cost. The specific solution procedures of the proposed algorithm are illustrated in Figure 2.

Figure 2 Framework of DRL-CCOC (see online version for colours)



4.1 Elements definition

At a certain time slot t, we can define that the environment state is S_t, the agent performs the action A_t. With the certain possibility, the environment is transferred to a new state S_{t+1} and the agent gets an immediate reward R_t. The aim is to maximise R_t by our optimal policy. The three elements are described as follows:

- State space:

$$S_t = (C(t)) \tag{20}$$

where is the weighted sum of the total completion time and energy consumption of the task set κ.

- Action space

$$A_t = \left(A_t^a, A_t^b\right) \tag{21}$$

where $A_t^a = \left(\sigma_i(t), \psi_i(t), i \in \kappa\right)$, σ_i and $\psi_i(t)$ represent the wireless channel bandwidth occupation ratio and the computing resource occupation ratio, respectively. $A_t^b = \left(\phi_i(t), x_i(t), i \in \kappa\right)$, $\phi_i(t)$ and $x_i(t)$ represent offloading decision and caching decision for task i.

- Reward space: ES (i.e., the agent) will receive an immediate reward $R_t(S_t, A_t)$ when the action A_t performed in a certain state S_t. Thus, the reward plays a critical role in the process of DRL-CCOC. Our optimisation problem is related to the reward function and the goal of the agent is to minimise C_i, i.e., maximise $R_t(S_t, A_t)$, that is

$$R_t = \begin{cases} -\sum_{i \in \kappa} C_i(t) & (17a)-(17g), \\ v, & otherwise. \end{cases} \tag{22}$$

where v ($v \le 0$ and $v < -\sum_{i \in \kappa} C_i(t)$) denotes a constant that indicates environment gives a bad feedback for agent to study better.

4.2 DRL-CCOC algorithm

Based on deep deterministic policy gradient (DDPG) algorithm, the framework of the algorithm we design is composed of an agent, an environment and three elements. Three elements are state S, action A and reward function R. The network is divided into policy network and value network. They can be described into four networks: current actor network, current critic network, target actor network and target critic network.

Current and target actor networks are policy network. Policy network outputs policy in which the current network outputs current action A_t according to current state S_t. The target network has the same networks structure as the current network. The responsibility of it is predicting the next action A_{t+1} based on the next state S_{t+1}.

Current and target critic networks are value network. We can obtain the corresponding value function $Q(S, A, \omega)$ according to current state S_t or next state S_{t+1} and current action A_t or next action A_{t+1} to critic the policy network. The current and target critic network have the same structure.

When a random strategy is used, the action taken is based on a probability distribution and we will obtain an uncertain action. The deterministic strategy our algorithm used is simple. Although the probability of action is different in the same state, there is only one maximum probability. If only taking the action with the maximum probability, we remove the probability distribution. At a time slot t, the action is uniquely deterministic, that is, the offloading, caching and resources allocation strategy becomes $\pi_\theta(S_t)$. The output of policy network is a deterministic action, i.e.

$$A_t = \pi_\theta(S_t) \tag{23}$$

Our algorithm adopts deterministic policy getting from function π_θ. We can get the current action from network π_θ and θ is the parameters of its neural network. In order to add the randomness to the learning process, we will add Ornstein-Uhlenbeck (OU) noise H_t to the action A_t^a selected by our algorithm, that is, the expression for the final resources allocation action interacting with the environment is

$$A_t^a = \pi_\theta(S_t) + \xi H_t \tag{24}$$

where ξ represents the annealing factor of noise. The value of ξ decreases with the increase of iterations, the convergence speed of the training process can be improved by using the noise.

We define a probability range of Ω to represent the offloading and caching decision, which is discrete action A_t^b. By Ω, we can transform continuous inputs into discrete inputs.

$$A_t^b = \begin{cases} 0, & \Omega \le 0.5, \\ 1, & \Omega > 0.5. \end{cases} \tag{25}$$

ES, as the agent, performs the action A_t, then action A_t interacts with the environment to get the reward R_t and the initial state S_t switches to next state S_{t+1}. (S_t, A_t, S_{t+1}, R_t) will be stored into experience replay buffer. At each training session, G samples are randomly sampled from the experience replay buffer to train. After that, the target actor network will take next action A_{t+1} according to the next state S_{t+1} sampled in the experience replay buffer.

The target critic network computes value function $Q'(S_{t+1}, A_{t+1}, \omega')$ for sample $\iota + 1$, $\iota \in \{1, 2, \ldots, G\}$, so the target Q value is

$$y_t = R + \varpi Q'\left(S_{t+1}, A_{t+1}, \omega'\right) \tag{26}$$

where ϖ is the discount factor.

In the value network, we use the mean square error loss function

$$Loss(\omega) = \frac{1}{G} \sum_{t=1}^{G} \left(y_t - Q(S_t, A_t, \omega)\right)^2 \tag{27}$$

where G is the number of samples of batch. We make use of gradient descent method to minimise $Loss(\omega)$ to update current critic network parameter ω as follow:

$$\omega \leftarrow \omega + \vartheta \nabla_\omega Loss(\omega) \tag{28}$$

where ϑ is the learning rate of ω.

As for actor network, it updates the current actor network parameter θ by the feedback Q value of the current critic network. Therefore, the loss of actor network is defined as

$$J(\theta) = -\frac{1}{G}\sum_{t=1}^{G} Q(S_t, A_t, \theta) \tag{29}$$

To make the loss smaller and Q value larger, we use gradient descent to update parameter θ as follows:

$$\theta \leftarrow \theta + \chi \nabla_\theta J(\theta) \tag{30}$$

where χ is the learning rate of θ.

The target network parameter θ' and ω' are both periodically replicated from θ and ω by soft update as

$$\theta' \leftarrow \lambda\theta + (1-\lambda)\theta' \tag{31}$$

and

$$\omega' \leftarrow \lambda\omega + (1-\lambda)\omega' \tag{32}$$

where is the update speed coefficient, which is usually small, like 0.1 or 0.01. By it, the process of training can be very stable. After constantly learning, ES can fetch the optimal offloading, caching and resources allocation strategy by leveraging network parameters that are optimal. In order to better understand the process of the solution, we concise the process as follows:

Algorithm 1 Deep reinforcement learning-based collaborative computation offloading and caching decision algorithm

Input: w_i and c_i, $i \in \kappa$

Output: Optimal policy $(\phi_i^*, x_i^*, \sigma_i^*, \psi_i^*)$, $i \in \kappa$.

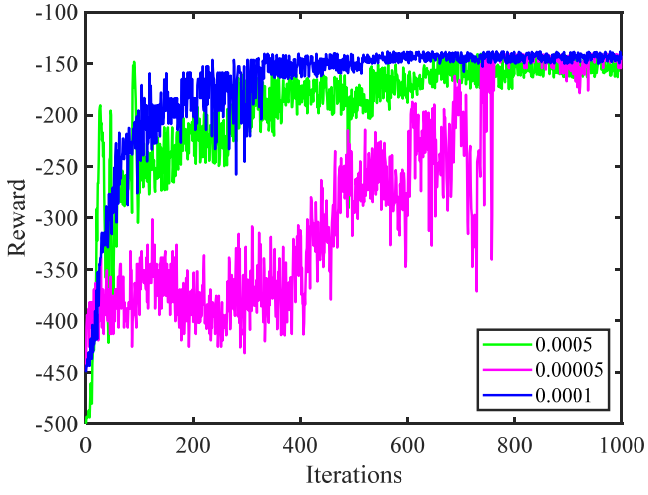
BEGIN
Initialise target network parameters θ' and ω' with $\theta' = \theta$, $\omega' = \omega$;
Initialise experience replay buffer
FOR ζ **from** 0 to maximum episode **DO**
  Initialise OU noise and initial state S_t;
  FOR $t = 0$ to $\aleph$ **DO**
    Based on the current state S_t, select action $A_t = (A_t^a, A_t^b)$
    Execute A_t
    Obtain next state S_{t+1}, reward R_t by interacting with environment
    Save (S_t, A_t, S_{t+1}, R_t) into experience replay buffer;
    Update state S_t to S_{t+1}
    Sample G samples from the experience replay buffer, compute target Q value based on equation (26)
    Update current critic network parameter ω by equations (27) and (28)
    Update current actor network parameter θ by equations (29) and (30)
    Soft update target critic network parameter θ' by equation (31)
    Soft update target critic network parameter ω' by equation (32)
  END FOR
END FOR
Obtain optimal **policy** $(\phi_i^*, x_i^*, \sigma_i^*, \psi_i^*)$ and minimum C^*.
END

5 Simulation results and discussions

In this section, we evaluate the effectiveness of the DRL-CCOC algorithm through simulation experiments. Compared with several typical related schemes, our algorithm is superior to the other.

In our simulation environment, we choose 7 different tasks from 7 different IoT devices. The size of tasks w are randomly assigned from 100 Kb to 400 Kb. The numbers of CPU cycles c are set randomly from 10 to 30 Gcycle. The wireless channel bandwidth B is set to 100 Mb/s; the local computing capability of IoT devices f_i^{loc} is range from 3 Gcycle/s to 25 Gcycle/s; the computing capability of ES f_i^e is 180 Gcycle/s. We set every IoT device with the same computing and transmission power, i.e., P_i and P_{up} are 0.04 J and 0.06 J, respectively. The computing power of the ES P_e is 0.1 J. The weight coefficient ∂ is 0.5. The update coefficient of the target network hyperparameter χ is set to 0.001, and the discount factor γ is 0.99. TensorFlow and MATLAB are jointly used in our simulation.

Figure 3 Convergence effect of the reward function under different learning rates of actor network (see online version for colours)



5.1 The comparison of DRL-CCOC convergence

We compare the reward of actor network under different learning rates: 0.00005, 0.0001 and 0.0005. We can conclude the following points. Firstly, to a certain extent, the reward converges more and more slowly as the learning

rate decreases; the efficiency of iterative optimisation is deficient. In contrast, as the learning rate increases, the agent will learn rather fast leading to missing the optimal reward and the reward will oscillate around the optimal reward. Therefore, the learning rate of actor network cannot be set really low or high. According to many simulation results, we set the learning rate to 0.0001.

Figure 4 Convergence effect of the loss function under different learning rates of critic network (see online version for colours)

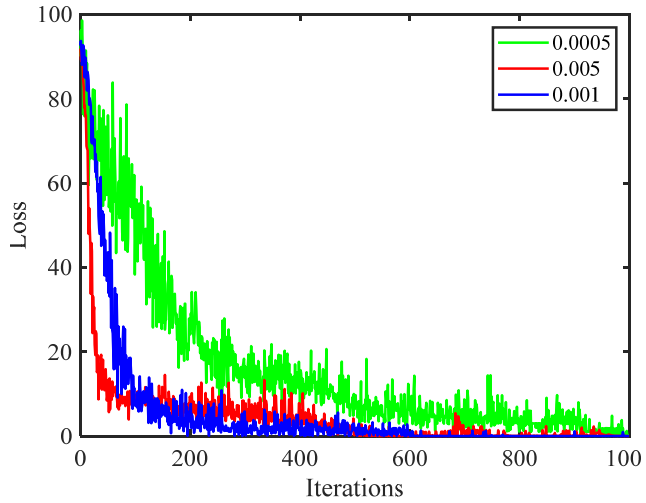


Figure 4 depicts the loss of the critic network with different learning rates: 0.0005, 0.001 and 0.005. The higher the learning rate is, the faster the loss converges. However, if the learning rate is high, the loss function could fluctuate greatly. When the learning rate is rather low, the convergence rate will not reach the desired effect and the loss function cannot converge to the optimal value. Through many simulation experiments analysis of the results, setting the learning rate to 0.001 is more appropriate.

5.2 The comparison of different methods

In this subsection, we compare several methods with our method, which are 'offloading fully', 'local fully', 'offloading based on DRL-CCOC', and 'greedy'. 'offloading fully' represents that all tasks are offloaded to ES without optimisation; 'local fully' expresses that tasks are computed at local totally; 'offloading based on DRL-CCOC' means that offloading all tasks using our algorithm without computing locally; 'Greedy' takes the best result from each iteration under current state. We compare the total cost by varying ES's computing capability, bandwidth between the fog node and devices, and the number of tasks.

Figure 5 describes the effect of different ES's computing capabilities on the total cost. We can discover that 'local fully' has no relation to ES's computing capability. Furthermore, the total costs of all methods decrease except 'local fully'. When the capability value grows to about 225 Gcycle/s, the total cost of every offloading-related method is all less than 'local fully'. Additionally, 'DRL-CCOC' is better than 'offloading based on DRL-CCOC' which offloads all tasks thus introducing significant transmission delay. Finally, 'offloading based on DRL-CCOC' is superior to 'offloading fully' since the former reduces calculation delay of cached tasks. The method we proposed is approximated to the ideal 'Greedy'.

Figure 5 ES's computing capabilities effect of total cost under different methods (see online version for colours)

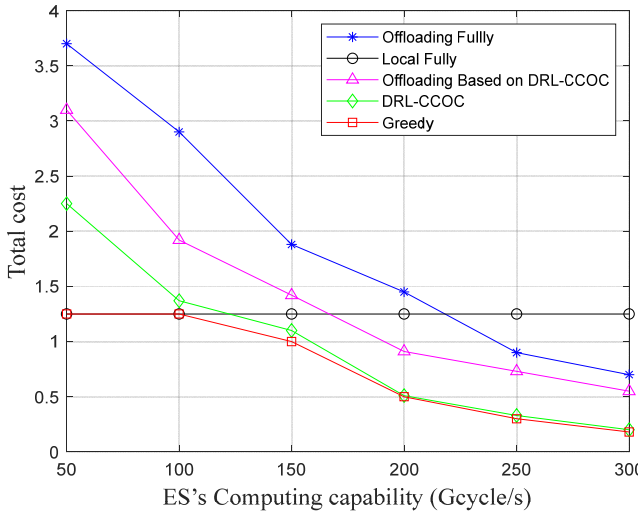
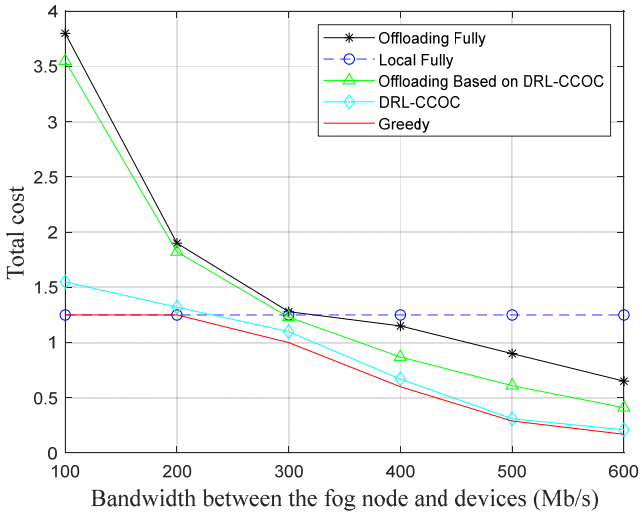


Figure 6 Bandwidth effect of total cost under different methods (see online version for colours)



In Figure 6, we compare the total cost under different bandwidth between the fog node and devices. From the figure, we can deduce that bandwidth has more effect on offloading-related methods. When bandwidth is large enough, offloading-related methods have a lower cost than local-related methods. Moreover, our method always follows 'Greedy' cost and achieves optimal results. The advantages of 'DRL-CCOC' are obvious when the bandwidth resource is scarce at first. As the bandwidth increases, the transmission speed increases and the trend of cost is more stable. Based on the consideration of computation offloading and caching, our method shows significant effectiveness and superiority.

Figure 7 Number of tasks effect of total cost under different methods (see online version for colours)

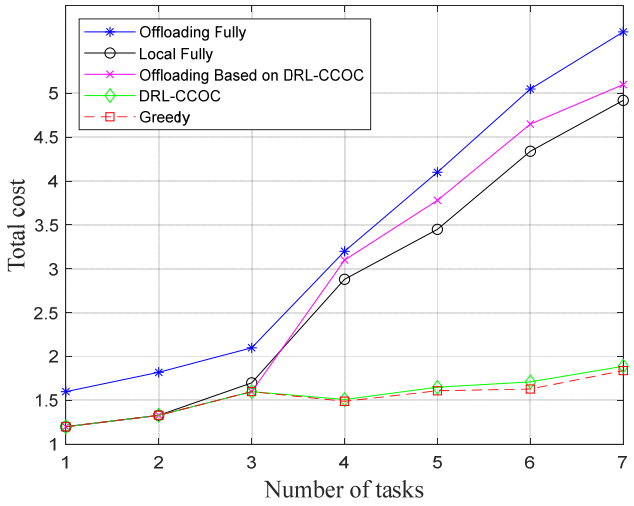


Figure 7 presents the number of tasks connected with the total cost. When the number of tasks increases, the total cost tends to rise overall. Initially, all methods' cost but 'offloading fully' are the same. All tasks are computed locally for the tasks that do not exceed the local computing capacity. On the whole, the cost of 'offloading fully' is the highest, 'offloading based on DRL-CCOC' is the second, and then is 'local fully'. 'DRL-CCOC' is the closest to the total cost of greedy algorithm. After massive experiences, our method is more suitable for large-scale users.

Based on the above summation results, 'DRL-CCOC' we proposed has much robustness and steadiness in different ES's computing capability, bandwidth between the fog node and devices, and the number of tasks. It is not affected by the dynamic environment and changing conditions. Offloading and caching decisions will be made optimally and the weighted sum of delay and energy consumption minimised.

6 Conclusions

For satisfying the demands of resource-intensive and delay-sensitive applications and solving the problems of existing computation offloading algorithms, such as inability to handle massive amounts of data in time and the need of strong computing capability, we propose an intelligent computation offloading, resource allocation and collaborative caching scheme to jointly optimise delay and energy consumption. The aim of our optimisation problem is to minimise the weighted sum of the total cost under constraints. Contingent on the optimisation problem, a DRL-CCOC algorithm is proposed which can generate the optimal policy to reduce the cost of all tasks. Finally, after lots of simulation experiments and results analysis, our algorithm can converge at a faster rate and reduce the total cost significantly compared with other benchmark schemes.

References

Ale, L., Zhang, N., Fang, X. et al. (2021) 'Delay-aware and energy-efficient computation offloading in mobile-edge computing using deep reinforcement learning', *IEEE Transactions on Cognitive Communications and Networking*, September, Vol. 7, No. 3, pp.881–892.

Bi, S., Huang, L. and Zhang, Y.A. (2020) 'Joint optimization of service caching placement and computation offloading in mobile edge computing systems', *IEEE Transactions on Wireless Communications*, July, Vol. 19, No. 7, pp.4947–4963.

Chen, M., Hao, Y., Hu, L. et al. (2018) 'Edge-CoCaCo: toward joint optimization of computation, caching, and communication on edge cloud', *IEEE Wireless Communications*, June, Vol. 25, No. 3, pp.21–27.

Chen, X. (2015) 'Decentralized computation offloading game for mobile cloud computing', *IEEE Transactions on Parallel and Distributed Systems*, April, Vol. 26, No. 4, pp.974–983.

Chen, Y., Zhang, N., Zhang, Y. et al. (2019) 'Dynamic computation offloading in edge computing for internet of things', *IEEE Internet of Things Journal*, June, Vol. 6, No. 3, pp.4242–4251.

Chowdhury, M. (2021) 'Flexible heuristic-based prioritized latency-sensitive IoT application execution scheme in the 5G era', *International Journal of Embedded Systems*, September, Vol. 14, No. 4, pp.363–377.

Dave, H.V. and Kotak, N.A. (2023) 'Critical analysis of cache memory performance concerning miss rate and power consumption', *International Journal of Embedded Systems*, March, Vol. 15, No. 6, pp.516–524.

Dinh, T.Q., La, Q.D., Que, T.Q.S. et al. (2018) 'Learning forcComputation offloading in mobile edge computing', *IEEE Transactions on Communications*, December, Vol. 66, No. 12, pp.6353–6367.

Hao, Y., Chen, M., Hu, L. et al. (2018) 'Energy efficient task caching and offloading for mobile edge computing', *IEEE Access*, March, Vol. 6, pp.11365–11373, DOI: 10.1109/ACCESS.2018.2805798.

Jiang, C., Cheng, X., Gao, H. et al. (2019) 'Toward computation offloading in edge computing: a survey', *IEEE Access*, August, Vol. 7, No. 1, pp.131543–131558.

Lan, Y., Wang, X., Wang, D. et al. (2019) 'Task caching, offloading, and resource allocation in D2D-aided fog computing networks', *IEEE Access*, July, Vol. 7, pp.104876–104891.

Lei, L. et al. (2019) 'Learning-based resource allocation: efficient content delivery enabled by convolutional neural network', in *Proc. IEEE 20th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, pp.1–5.

Li, J., Gao, H., Lv, T. et al. (2018) 'Deep reinforcement learning based computation offloading and resource allocation for mec', in *Proc. IEEE Wireless Communications and Networking Conference (WCNC)*, pp.1–6.

Lin, H., Zeadally, S., Chen, Z. et al. (2020) 'A survey on computation offloading modeling for edge computing', *Journal of Network and Computer Applications*, November, Vol. 169, pp.1–25, DOI: 10.1016/j.jnca.2020.102781.

Lin, L., Liao, X., Jin, H. et al. (2019) 'Computation offloading toward edge computing', *Proceedings of the IEEE*, August, Vol. 107, No. 8, pp.1584–1607.

Liu, L., Lei, X. and Wang, Q. (2022) 'A multi-objective computation offloading algorithm in MEC environments', *International Journal of Computational Science and Engineering*, May, Vol. 25, No. 3, pp.298–307.

Liu, M., Yu, F.R., Teng, Y. et al. (2018) 'Computation offloading and content caching in wireless blockchain networks with mobile edge computing', *IEEE Transactions on Vehicular Technology*, November, Vol. 67, No. 11, pp.11008–11021.

Ma, X., Zhou, A., Zhang, S. et al. (2020) 'Cooperative service caching and workload scheduling in mobile edge computing', in *Proc. IEEE Conference on Computer Communications (ICCC)*, pp.2076–2085.

Ndikumana, A., Ullah, S., LeAnh, T. et al. (2017) 'Collaborative cache allocation and computation offloading in mobile edge computing', in *Proc. 19th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, pp.366–369.

Ning, Z., Dong, P., Kong, X. et al. (2019) 'A cooperative partial computation offloading scheme for mobile edge computing enabled internet of things', *IEEE Internet of Things Journal*, June, Vol. 6, No. 3, pp.4804–4814.

Simon, W.A., Qureshi, Y.M., Rios, M. et al. (2020) 'BLADE: an in-cache computing architecture for edge devices', *IEEE Transactions on Computers*, September, Vol. 69, No. 9, pp.1349–1363.

Wang, Y. and Friderikos, V. (2020a) 'A survey of deep learning for data caching in edge network', *Informatics*, October, No. 43, pp.1–29.

Wang, Y. and Friderikos, V. (2020b) 'Network orchestration in mobile networks via a synergy of model-driven and AI-based techniques', in *Proc. IEEE Eighth International Conference on Communications and Networking (ComNet)*, pp.1–5.

Wei, H., Luo, H., Sun, Y. et al. (2020) 'Cache-aware computation offloading in IoT systems', *IEEE Systems Journal*, March, Vol. 14, No. 1, pp.61–72.

Xu, J., Chen, L. and Zhou, P. (2018) 'Joint service caching and task offloading for mobile edge computing in dense networks', in *Proc. IEEE INFOCOM*, pp.207–215.

Yang, Z., Liu, Y., Chen, Y. et al. (2020) 'Cache-aided NOMA mobile edge computing: a reinforcement learning approach', *IEEE Transactions on Wireless Communications*, October, Vol. 19, No. 10, pp.6899–6915.

Yu, S., Langar, R., Fu, X. et al. (2018) 'Computation offloading with data caching enhancement for mobile edge computing', *IEEE Transactions on Vehicular Technology*, November, Vol. 67, No. 11, pp.11098–11112.

Zhang, N., Cheng, N., Gamage, A.T. et al. (2015) 'Cloud assisted hetnets toward 5G wireless networks', *IEEE Commun. Mag.*, June, Vol. 53, No. 6, pp.59–65.

Zhao, G., Xu, H., Zhao, Y. et al. (2020) 'Offloading dependent tasks in mobile edge computing with service caching', in *Proc. IEEE INFOCOM*, pp.1997–2006.

Zhao, Z., Bu, S., Zhao, T. et al. (2019) 'On the design of computation offloading in fog radio access networks', *IEEE Transactions on Vehicular Technology*, July, Vol. 68, No. 7, pp.7136–7149.