Hindawi
Wireless Communications and Mobile Computing
Volume 2019, Article ID 3816237, 20 pages
https://doi.org/10.1155/2019/3816237

*Research Article*

# Security and Cost-Aware Computation Offloading via Deep Reinforcement Learning in Mobile Edge Computing

**Binbin Huang**[1], **Yangyang Li**[2], **Zhongjin Li**[1], **Linxuan Pan**[3], **Shangguang Wang**[4], **Yunqiu Xu**[5] **and Haiyang Hu**[1]

[1]*School of Computer, Hangzhou Dianzi University, Hangzhou 310018, China*
[2]*National Engineering Laboratory for Public Safety Risk Perception and Control, China Academy of Electronics and Information Technology, Beijing, China*
[3]*State Key Laboratory for Novel Software Technology, Software Institute, Nanjing University, Nanjing, China*
[4]*State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing, China*
[5]*School of Software, University of Technology Sydney, Ultimo, NSW 2007, Australia*

Correspondence should be addressed to Zhongjin Li; lizhongjin@hdu.edu.cn

Received 17 September 2019; Accepted 3 December 2019; Published 23 December 2019

Academic Editor: Mohammed El-Hajjar



With the explosive growth of mobile applications, mobile devices need to be equipped with abundant resources to process massive and complex mobile applications. However, mobile devices are usually resource-constrained due to their physical size. Fortunately, mobile edge computing, which enables mobile devices to offload computation tasks to edge servers with abundant computing resources, can significantly meet the ever-increasing computation demands from mobile applications. Nevertheless, offloading tasks to the edge servers are liable to suffer from external security threats (e.g., snooping and alteration). Aiming at this problem, we propose a security and cost-aware computation offloading (SCACO) strategy for mobile users in mobile edge computing environment, the goal of which is to minimize the overall cost (including mobile device's energy consumption, processing delay, and task loss probability) under the risk probability constraints. Specifically, we first formulate the computation offloading problem as a Markov decision process (MDP). Then, based on the popular deep reinforcement learning approach, deep Q-network (DQN), the optimal offloading policy for the proposed problem is derived. Finally, extensive experimental results demonstrate that SCACO can achieve the security and cost efficiency for the mobile user in the mobile edge computing environment.

## 1. Introduction

With the increasing popularity of mobile devices (e.g., smart phones and tablets), the number of various mobile applications (i.e., virtual reality (VR) and face recognition) is explosively growing [1]. To process such a large number of complex mobile applications efficiently, mobile devices need to be equipped with considerable resources (i.e., high computing capacity and battery power) [2, 3]. Unfortunately, due to the limited physical size of mobile devices, they are usually resource-constrained. The conflict between the resource demand for executing complex tasks and the limited resource capacity of mobile devices impose a big challenge for mobile application execution, which drives the transformation of computing paradigm [4].

To reduce this conflict, mobile edge computing has emerged as a promising computing paradigm with the objective of bringing the computing and storage capacities close to mobile devices [5, 6]. Within one-hop communication range of mobile devices, there are a number of edge servers who have enormous computation and storage resources. Therefore, mobile devices can offload computation tasks to edge servers directly through wireless channels [7], thereby significantly meeting the ever-increasing computation

demands from mobile applications, reducing their processing delay, and saving the mobile devices' energy consumption.

Despite its advantages, computation task offloading via mobile edge computing inevitably encounters two challenges as follows: one is the offloading environmental dynamics, such as the time-varying channel quality and the task arrival, which can impact the offloading decisions. And the other is the security. According to several surveys, security is one of the critical issues in mobile edge computing [8–19]. Due to the open nature of the mobile edge computing environment, these tasks offloaded to the edge servers are susceptible to hostile attacks from outside. For example, the offloaded computation tasks from the mobile device to the edge servers can be intentionally overheard by a malicious eavesdropper. Hence, it needs to employ various types of security services to effectively defend against the hostile attacks and protect these tasks. However, using security services inevitably incurs lots of extra security time and security workload, which will increase the mobile device's energy consumption and task's processing time, and thereby influencing the offloading decisions. Therefore, it is a big challenge to design a proper task offloading policy to optimize the long-term weighted cost of mobile device's energy consumption, task processing delay, and number of dropping tasks while satisfying risk rate constraint.

To meet the aforementioned challenges, we propose a security and cost-aware computation offloading (SCACO) strategy in the mobile edge computing environment, the goal of which is to minimize the long-term cost under the risk rate constraint. More specifically, we formulate the computation offloading problem as a Markov decision process (MDP). Our formulated offloading problem is a high-dimensional decision-making problem. The DQN algorithm has achieved excellent performance in solving this kind of problem. To achieve this, a deep Q-network- (DQN-) based offloading scheme is proposed. Figure 1 illustrates a DQN framework for computation task offloading in mobile edge computing. As shown in Figure 1, the environment state which consists of the number of arriving tasks, the mobile device and edge servers' execution queue states, and the channel quality states can be observed. Based on the current state, an optimal action, e.g., how many tasks should be assigned to execute locally, how many tasks should be offload to edge servers, and which security level should be employed, is chosen by the agent. After taking an action at current state, the reward can be calculated, and the current state, the action taken, and the reward are stored into replay memory. The main contributions of this paper can be summarized as follows:

(i) We select appropriate security servers to guarantee the offloaded tasks' security. The security overhead model [20] is exploited to quantify the security time. Based on it, the security workload can be measured. In our architecture, the total workload of the offloaded task consists of task execution workload and security workload.

(ii) We formulate the security-aware task offloading problem as an infinite Markov decision process with the risk rate constraint, the main goal of which is to minimize the long-term computation offloading costs while satisfying the risk rate constraint in the dynamic environment.

(iii) We propose a SCACO strategy based on the DQN algorithm to solve the proposed formulation. To demonstrate the efficiency of the SCACO strategy and the impact of the security requirement, we conduct extensive experiments, with respect to (w.r.t.) various performance parameters (such as the task arrival rate, the task workload, the risk coefficient, and the risk rate). The experimental results demonstrate that the SCACO strategy can minimize the long-term cost while satisfying the risk rate constraint.

We organize this paper as follows: Section 2 summarizes the related work. Section 3 describes the system models. Section 4 formulates security-aware computation offloading problem formulation. Section 5 describes the SCACO strategy based on the DQN algorithm. Section 6 describes the experimental setup and analyzes experimental results. Section 7 concludes this paper and identifies future directions.

## 2. Related Work

To meet the quality of service for different types of mobile applications, the computation tasks can be offloaded to the edge servers with sufficient computation capacity. Accordingly, an increasing amount of research has focused on the problem of computation offloading in mobile edge computing. Specifically, in [21], an efficient one-dimensional search algorithm is designed to minimize the computation task scheduling delay under the power consumption constraint. In [2], a computation task offloading optimization framework is designed to jointly optimize the computation task execution latency and the mobile device's energy. In [2], an online dynamic task offloading scheme is proposed to achieve a trade-off between the task execution delay and the mobile device's energy consumption in mobile edge computing with energy harvesting devices. In [22], a suboptimal algorithm is proposed to minimize the maximal weighted cost of the task execution latency and the mobile device's energy consumption while guaranteeing user fairness. In [23], a workload allocation scheme is proposed to jointly optimize the energy consumption and the execution delay in mobile edge computing with grid-powered system. However, all the above works mainly focus on optimizing the one-shot offloading cost and fail to characterize long-term computation offloading performance. Accordingly, these offloading schemes may not suit for some applications which the long-term stability is more important than the profits of handing one task.

To optimize the long-term computation offloading performance, a lot of related works have been done. In particular, in [24], an efficient reinforcement learning-based algorithm is proposed to minimize the long-term computation task offloading cost (including both service delay and

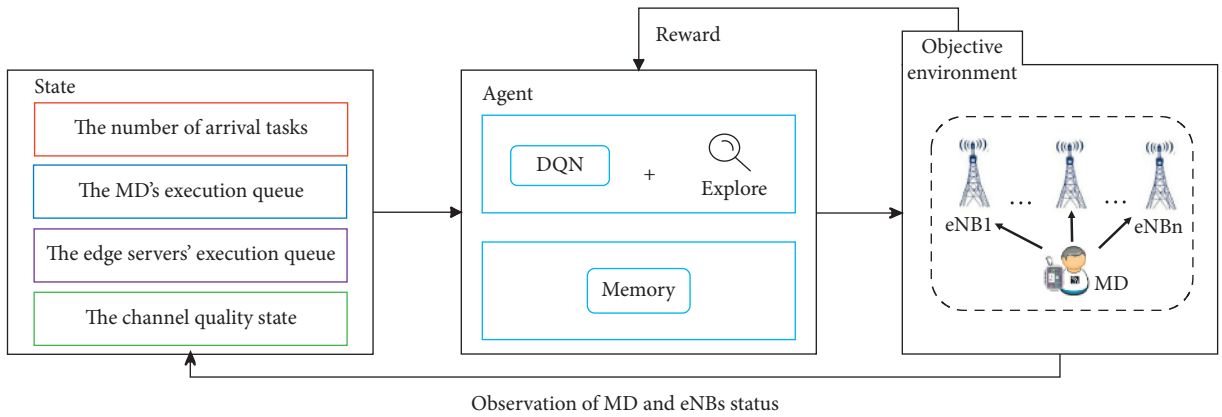


Figure 1: DQN framework for computation task offloading in mobile edge computing.

operational cost) in energy harvesting mobile edge computing systems. In [25], a deep Q-network-based strategic computation offloading algorithm is proposed to minimize the long-term cost which is the weighted sum of the execution delay, the handover delay, and the computation task dropping cost. In [26], the Lyapunov optimization-based dynamic computation offloading policy is proposed to minimize the long-term execution latency and task failure for a green mobile edge computing system with wireless energy harvesting-enabled mobile devices. In [27], the Lyapunov optimization method is utilized to minimize the long-term average weighted sum energy consumption of the mobile devices and the edge server in a multiuser mobile edge computing environment. In [28], the game theory and reinforcement learning is utilized to efficiently manage the distributed resource in mobile edge computing. However, none of the above work considers the impact of security issue on computation offloading. In fact, security cannot be ignored because it is a key issue in mobile edge computing. Therefore, the above schemes are not suitable for security-aware dynamic computation offloading in mobile edge computing.

With the escalation of the security threatens of data in the cloud, mobile cloud environments, and mobile edge computing [8–19], some measures have been implemented to protect security-critical applications. Specifically, in [29], a task-scheduling framework with three features is presented for security sensitive workflow framework. In [30], an SCAS scheduling scheme is proposed to optimize the workflow execution cost under the makespan and security constraints in clouds. In [31], an SABA scheduling scheme is designed to minimize the makespan under the security and budget constraints. In [32], a security-aware workflow scheduling framework is designed to minimize the makespan and execution cost of workflow while meeting the security requirement. However, to the best of our knowledge, all of the above methods are mainly designed for the workflow scheduling in the cloud computing or mobile cloud computing environment. They are not suitable for computation offloading in mobile edge computing. In [17], a deep-learning-based approach is proposed to detect malicious applications on the mobile device, which greatly enhances the security of mobile edge computing. However, this paper mainly focuses on the detection of security threats. In [33], a joint optimization problem which includes the secrecy provisioning, computation offloading, and radio resource allocation is formulated, and an efficient algorithm is proposed to solve this joint optimization problem. However, this paper failed to optimize the long-term computation task offloading cost.

All the above studies have focused on the security problem of workflow scheduling. Little attention has been paid to the effect of the security problem of task offloading on the long-term offloading cost. Motivated by that, in this paper, we mainly focus on security and cost-aware dynamic computation offloading in mobile edge computing. We try to minimize the long-term computation offloading cost while satisfying risk rate constraint.

## 3. System Models

In this section, we first provide an overview of the security-aware mobile edge computing model. Then, the security overhead model and network and energy model are presented. At last, we formulate the problem of security-aware computation task offloading. To improve the readability, the main notations and their semantics used in this paper are given in Table 1.

*3.1. Mobile Edge Computing Model.* As depicted in Figure 2, we consider in this paper a mobile edge computing system consisting of a single mobile user and $n$ edge servers. The mobile user can generate a series of independent computation tasks [34, 35] which need to be scheduled to execute locally or to execute remotely. Due to the mobile device's limited computing resource and battery capacity, all computation tasks cannot be executed locally within a timely manner. Therefore, a part of these can be offloaded to $n$ edge servers with relatively rich resources. These offloaded tasks are first stored in a dedicated executing queue and then are executed sequentially. The system time is logically divided to equal length time slots, and the duration of each time slot is $T_{\text{slot}}$ (in seconds). For convenience, we denote the index sets of the time slots as $\tau \in \mathcal{T} = \{0, 1, \ldots\}$. The value of $T_{\text{slot}}$ is

Table 1: Notations.

| Symbols | Definition |
|---|---|
| $n$ | Number of edge servers |
| $\lambda_{\text{task}}$ | Tasks' arrival rate |
| $T_{\text{slot}}$ | The duration of the time slot |
| $R_i$ | The link rate between MD and edge server $e\text{NB}_i$ |
| $\text{sl}_{c,i}$ | The set of security levels of task $t$ offloaded to the $i$th edge server |
| $\mathcal{Q}_{\text{arr}}(\tau)$ | The number of arrival tasks at the $\tau$th time slot |
| $\mathcal{Q}_u^e(\tau)$ | The user executing queue's state at the $\tau$th time slot |
| $\mathcal{Q}_c^e(\tau)$ | The edge server executing queue's state at the $\tau$th time slot |
| $C(\tau)$ | The immediate cost at the $\tau$th time slot |
| $E(\tau)$ | The immediate energy consumption at the $\tau$th time slot |
| $T(\tau)$ | The immediate delay at the $\tau$th time slot |
| $D(\tau)$ | The number of dropping tasks at the $\tau$th time slot |
| $P_{\max}$ | The risk probability constraint of each offloaded task |

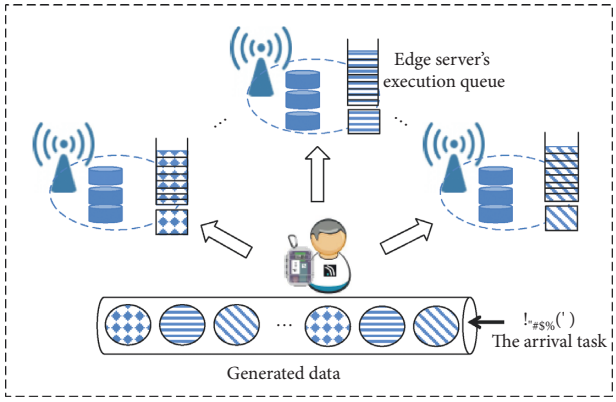


Figure 2: Computation task offloading framework with multiple edge servers.

usually decided by the channel coherence time. It means that the channel remains static within a time slot but varies among different time slots [26]. At the beginning of each time slot, the user makes an optimal offloading decision.

The mobile device can be denoted by a triple $\text{MD} = \langle f_u, N_u, P_u \rangle$, where $f_u$ denotes the mobile device's CPU cycle frequency, $N_u$ denotes the number of the mobile device's processor cores, and $P_u$ denotes the mobile device's power. Especially, $P_u$ can be further represented by a tuple $P_u = \langle P_u^e, P_u^{tx} \rangle$, where $P_u^e$ and $P_u^{tx}$ denote the mobile device's computation power (in Watt) and transmitting power, respectively. The mobile device has an execution queue with its size $|\mathcal{Q}_u^e|$. If the queue is already full, the new arrival tasks will be dropped.

The computation task model widely used in the existing literature [34, 35] is adopted in this paper. According to it, a computation task can be abstracted into a three-field notation $t = (W, D_{tx}, D_{rx})$, in which $W$ denotes the task computation workload (in CPU cycles per bit), $D_{tx}$ denotes the task input data size (in MB), and $D_{rx}$ denotes the task output data size. In addition, we assume that the computation tasks' arrival process for the mobile device MD follows a Poisson distribution with a parameter $\lambda_{\text{task}}$.

The set of $n$ edge servers can be denoted by $e\text{NB} = \{e\text{NB}_1, \ldots, e\text{NB}_i, \ldots, e\text{NB}_n\}$, where $e\text{NB}_i$ denotes the $i$th edge server. Each edge server $e\text{NB}_i$ has different configurations, such as the number of processor cores and the processor frequency. We use a two tuple $e\text{NB}_i = \langle f_{c,i}, N_{c,i} \rangle$ to represent the edge server $e\text{NB}_i$, where $f_{c,i}$ denotes its processor frequency and $N_{c,i}$ denotes the number of processor cores. Each edge server has an execution queue with the size $|\mathcal{Q}_c^i|$. When the edge server receives the offloaded tasks from the mobile device, it firstly stores the tasks in its execution queue and then executes them sequentially. Let $\mu_c^i$ denote the task processing rate of the $i$th edge server. Therefore, $\mu_c^i$ can be calculated by the following equation:

$$\mu_c^i = \frac{f_{c,i}}{W}. \tag{1}$$

*3.2. Security Overhead Model.* The computation tasks offloaded to the edge servers are confronted with security threats. Fortunately, confidentiality service and integrity service can guard against these common security threats [29–33, 36, 37]. Confidentiality service can protect data from stealing by enciphering methods. Meanwhile, integrity service can ensure that data are not tampered. By flexibly selecting these two security services, an integrated security protection is formed to protect against a diversity of security threats. Based on these above services, the offloading process of a task with security protection is shown in Figure 3. We can observe from Figure 3 that confidential service (denoted as E) is first employed to encrypt the offloaded task. The security levels and processing rates of cryptographic algorithms for confidential service [20, 30, 36, 37] can be found in Table 2. Then, to protect the offloaded computation task from alteration attacks, integrity service (denoted as H) is successively employed to implement a hash algorithm to it. The security levels and processing rates of hash functions for integrity service [20, 30, 36, 37] can be seen in Table 3. After the encrypted task is delivered to the $i$th edge server, it is firstly decrypted (denoted as DE) and its integrity is verified (denoted as IV), and then it is executed, and finally the computation result is sent to the mobile device.

Each offloaded computation task may require the confidentiality service and integrity service with various security levels. For the sake of simplicity, let cf and ig represent the confidentiality service and integrity service, respectively. Let $\text{sl}_{c,i} = \left\{\text{sl}_{c,i}^{\text{cf}}(\tau), \text{sl}_{c,i}^{\text{ig}}(\tau)\right\}$ to be the set of security levels of task $t$ offloaded to the $i$th edge server, where $\text{sl}_{c,i}^{\text{cf}}(\tau)$ represents the security level of confidentiality service and $\text{sl}_{c,i}^{\text{ig}}(\tau)$ represents the security level of integrity service. Security service incurs sometime overhead. According to [20], the total encryption overheads of task $t$ offloaded to the $i$th edge server can be calculated by the following equation:

$$T_{c,i}^{\text{E}} = \sum_{\text{type}\in\{\text{cf,ig}\}} \frac{k \cdot D_{tx}}{f_u \cdot N_u \cdot V\left(\text{sl}_{c,i}^{\text{type}}\right)}, \tag{2}$$

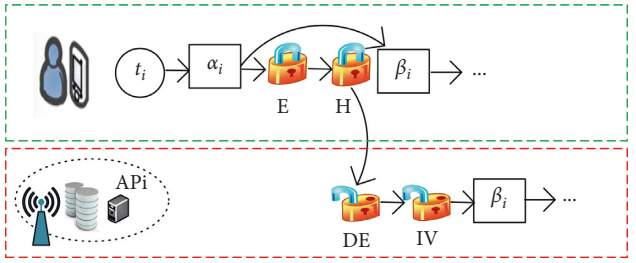


Figure 3: The task execution process with security services.

Table 2: The encryption algorithms for confidential service.

| Encryption algorithms | $\mathrm{sl}_{cf}$: security level | $V(\mathrm{sl}_{cf})$: processing rate (Mb/s) |
|---|---|---|
| IDEA | 1.0 | 11.76 |
| DES | 0.85 | 13.83 |
| Blowfish | 0.56 | 20.87 |
| AES | 0.53 | 22.03 |
| RC4 | 0.32 | 37.17 |

Table 3: The hash functions for integrity service.

| Hash functions | $\mathrm{sl}_{\mathrm{cf}}$: security level | $V(\mathrm{sl}_{\mathrm{cf}})$: processing rate (Mb/s) |
|---|---|---|
| TIGER | 1.0 | 75.76 |
| RipeMD160 | 0.75 | 101.01 |
| SHA-1 | 0.69 | 109.89 |
| RipeMD128 | 0.63 | 119.05 |
| MD5 | 0.44 | 172.41 |

where $k = 2.2$ and $V(\mathrm{sl}_{c,i}^{\mathrm{cf}})$ and $V(\mathrm{sl}_{c,i}^{\mathrm{ig}})$ denote the processing rates of the security levels $\mathrm{sl}_{\mathrm{cf}}$ and $\mathrm{sl}_{\mathrm{ig}}$, respectively. The total decryption overheads of the offloaded task $t$ in the $i$th edge server can be computed by the following equation:

$$T_{c,i}^{\mathrm{DE}} = \frac{f_u \cdot N_u \cdot T_u^{\mathrm{E}}}{f_{c,i} \cdot N_{c,i}}. \tag{3}$$

*3.3. Workload Shaping with Security Guarantee.* The security services incur not only time overhead but also security workload. The security workload of task $t$ offloaded to the $i$th edge server is incurred by encrypting it in mobile device and decrypting it in edge server. Based on the security time overhead of task $t$, the security workload which is incurred by encrypting task $t$ in the mobile device can be computed by the following equation:

$$\mathrm{SW}_{c,i}^{\mathrm{E}}(t) = T_{c,i}^{\mathrm{E}} \cdot f_u \cdot N_u. \tag{4}$$

The security workload which is incurred by the decrypting task $t$ in the $i$th edge server can be computed by the following equation:

$$\mathrm{SW}_{c,i}^{\mathrm{DE}}(D_{tx}) = T_{c,i}^{\mathrm{DE}} \cdot f_{c,i} \cdot N_{c,i}. \tag{5}$$

Based on the security services introduced above, we further quantitatively calculate the risk probability of task $t$ with different security levels. Without loss of generality, we assume that the distribution of risk probability follows a *Poisson* probability distribution for any given time interval. The risk probability of an offloaded computational task $t$ is related to the set $\mathrm{sl}_{c,i} = \{\mathrm{sl}_{c,i}^{\mathrm{cf}}(\tau), \mathrm{sl}_{c,i}^{\mathrm{ig}}(\tau)\}$ of security levels. Let $P(\mathrm{sl}_{c,i}^{k})$ denote the risk probability of an offloaded task which employs either of the two security services. $P(\mathrm{sl}_{c,i}^{k})$ can be denoted by the following equation [38, 39]:

$$P\left(\mathrm{sl}_{c,i}^{k}\right) = 1 - \exp\left(-\lambda_k\left(1 - \mathrm{sl}_k\right)\right), \quad k \in \{\mathrm{cf}, \mathrm{ig}\}. \tag{6}$$

The risk probability $P(t)$ of an offloaded task $t$ which employs these two security services with different levels can be computed by the following equation:

$$P(t) = 1 - \prod_{k \in \{\mathrm{cf},\mathrm{ig}\}} \left(1 - P\left(\mathrm{sl}_{c,i}^{k}\right)\right). \tag{7}$$

Given the risk probability constraint of each offloaded computational task is $P_{\max}$, the risk probability $P(t)$ must meet the following constraint:

$$P\left(a_c^k\right) \le P_{\max}. \tag{8}$$

In order to minimize the security workload while satisfying security requirement, it is a critical problem to select the levels of security services for each task. We formulate this problem as follows:

$$\begin{aligned} &\text{minimize} \quad \mathrm{SW}(t) = \mathrm{SW}_u^E(t) + \mathrm{SW}_{c,i}^{DE}(t) \\ &\text{subject to} \quad (8). \end{aligned} \tag{9}$$

As shown in Tables 2 and 3, the levels of each security service are discrete. For a task $t$, there are $5 \times 5$ types of security service composition. Hence, we can traverse these compositions to find the optimal service levels.

*3.4. Communication Model.* The computation tasks offloaded from the mobile device to edge servers are performed by wireless channels. Due to user's mobility, the wireless channel gain state is dynamically changing at each time slot, which induces dynamic change of wireless channel transmission rate. Let $R_i(\tau)$ denote the transmission rate of wireless channel (i.e., data transmission capacity) from the mobile device to the $i$th edge server $e\mathrm{NB}_i$ in the $\tau$th time slot, which is given as follows:

$$R_i(\tau) = B_i \log_2\left(1 + \frac{P_u G_i}{\sigma^2}\right), \quad \forall i \in \mathcal{N}, \tag{10}$$

where $B_i$ is the wireless channel bandwidth of the $i$th edge server, $P_u$ is the transmission power of the mobile device, $\sigma^2$ is the Gaussian white noise power, and $G_i$ denotes the wireless channel gain of $e\mathrm{NB}_i$.

*3.5. Problem Statement.* The computation offloading process can be formulated as an infinite Markov decision process. At the beginning of each time slot, an offloading decision is

made based on the current system state, which consists of the number of arrival tasks, the remaining number of tasks in the user's executing queue, the transmission rates between user and $n$ edge servers, and the remaining number of tasks in $n$ edge servers' executing queues. The offloading decision mainly determines the number of tasks assigned to execute locally and the number of tasks offloaded to different edge servers. To protect the offloaded computation tasks from malicious attacks, security services need to be employed. Security services incur time overheads and security workload. Hence, the objective of this paper is to minimize the long-term cost while meeting the risk probability constraint.

## 4. Security-Aware Computation Offloading Problem Formulation

In this section, we first define the state and action spaces. Then, the system state transition and reward function are derived. Finally, we define the objective and constraints.

*4.1. State Space.* At the $\tau$th time slot, the system state for the security-aware computation offloading problem can be denoted by the following equation:

$$s(\tau) = \left(\mathcal{Q}_{\text{arr}}(\tau), \mathcal{Q}_u^e(\tau), R(\tau), \mathcal{Q}_c^e(\tau)\right) \in \mathbb{S}, \tag{11}$$

where $\mathcal{Q}_{\text{arr}}(\tau) \in \{0, 1, \ldots\}$ denotes the number of arrival tasks at the $\tau$th time slot. $\mathcal{Q}_u^e(\tau) \in \{0, 1, \ldots, |\mathcal{Q}_u^e(\tau)|\}$ denotes the user's executing queue's state at the $\tau$th time slot. $R(\tau) = \{(R_1(\tau), \ldots, R_i(\tau), \ldots, R_n(\tau)) \mid R_i(\tau) \in [10, 100]\} \in \mathbb{R}$ is a vector of the transmission rate state between user and $n$ edge servers at the $\tau$th time slot. $\mathcal{Q}_c^e(\tau) = \{(\mathcal{Q}_{c,1}^e(\tau), \ldots, \mathcal{Q}_{c,i}^e(\tau), \ldots, \mathcal{Q}_{c,n}^e(\tau)) \mid \mathcal{Q}_{c,i}^e(\tau) \in \{0, 1, \ldots, |\mathcal{Q}_{c,i}^e(\tau)|\}\} \in \mathcal{Q}$ is the vector of the edge servers' execution queue state at the $\tau$th time slot. $R_i(\tau)$ and $\mathcal{Q}_{c,i}^e(\tau)$ denote the $i$th edge server's transmission rate and the number of the remaining tasks, respectively. Note that $|\mathcal{Q}_u^e(\tau)|$ and $|\mathcal{Q}_{c,i}^e(\tau)|$ denote the size of the mobile device and the $i$th edger server's execution queue, respectively. The system state $s(\tau)$ can be observed at the beginning of the $\tau$th time slot. $\mathcal{Q}_u^e(\tau)$ and $\mathcal{Q}_c^e(\tau)$ dynamically evolve according to the offloading policy, while $R_i(\tau)$ can be dynamically calculated.

*4.2. Action Space.* The action $a(\tau)$ at the $\tau$th time slot can be defined by the following equation:

$$a(\tau) = \left(a_u^e(\tau), a_c^e(\tau), \text{sl}_c^{\text{cf}}(\tau), \text{sl}_c^{\text{ig}}(\tau)\right) \in \mathbb{A}, \tag{12}$$

where $a_u^e(\tau)$ is the number of tasks for executing locally. $a_c^e(\tau) = (a_{c,1}^e(\tau), \ldots, a_{c,i}^e(\tau), \ldots, a_{c,n}^e(\tau))$ is a vector of the number of tasks to be offloaded to $n$ edge servers. $a_{c,i}^e(\tau)$ is the number of tasks to be offloaded to the $i$th edge servers. Therefore, the total number of tasks to be processed at the $\tau$th time slot is $m(a(\tau)) = a_u^e(\tau) + \sum_{i=1}^{n} a_{c,i}^e(\tau)$. $\text{sl}_c^{\text{cf}}(\tau) = (\text{sl}_{c,1}^{\text{cf}}(\tau), \ldots, \text{sl}_{c,i}^{\text{cf}}(\tau), \ldots, \text{sl}_{c,n}^{\text{cf}}(\tau))$ is a vector of the confidentiality service's level which is employed by tasks offloaded to $n$ edge servers. $\text{sl}_{c,i}^{\text{cf}}(\tau) \in \{0, 1.0, 0.85, 0.56, 0.53, 0.32\}$ denotes the security level of confidentiality service employed by the tasks offloaded to the $i$th edge server. $\text{sl}_c^{\text{ig}}(\tau) = (\text{sl}_{c,1}^{\text{ig}}(\tau), \ldots, \text{sl}_{c,i}^{\text{ig}}(\tau), \ldots, \text{sl}_{c,n}^{\text{ig}}(\tau))$ is a vector of the integrity service's level which employed by tasks offloaded to $n$ edge servers. $\text{sl}_{c,i}^{\text{ig}}(\tau) \in \{0, 1.0, 0.75, 0.69, 0.63, 0.44\}$ denotes the security level of integrity service employed by the tasks offloaded to the $i$th edge server. Note that an action $a(\tau)$ must satisfy the constraint condition that the number of assigned tasks is equal to the current number of arrival tasks.

*4.3. State Transition and Reward Function.* Given the current state $s(\tau) = ((\mathcal{Q}_u^w(\tau), \mathcal{Q}_u^e(\tau), R(\tau), \mathcal{Q}_c^e(\tau)) \in \mathbb{S}$, after taking an action $a(\tau) \in \mathbb{A}$, the state is transferred to the next state $s\_(\tau) = (\mathcal{Q}_u^w\_(\tau), \mathcal{Q}_u^e\_(\tau), R\_(\tau), \mathcal{Q}_c^e\_(\tau)) \in \mathbb{S}$, and the transition probability can be denoted as $p(s\_(\tau) \mid s(\tau), a(\tau))$. The immediate cost obtained by taking the action $a(\tau) \in \mathbb{A}$ can be denoted as $C(\tau)$. The immediate cost $C(\tau)$ is the weighted sum of immediate energy consumption $E(\tau)$, immediate processing delay $T(\tau)$, and immediate task's dropping probability $D(\tau)$. Based on the above, the average long-term cost function can be denoted by equation (14):

$$C(\tau) = -\left(\varphi_1 E(\tau) + \varphi_2 T(\tau) + \varphi_3 D(\tau)\right), \tag{13}$$

$$\overline{C} = \lim_{\tau \longrightarrow \infty} \sup \mathbb{E}_\pi \left[\sum_{\tau=0}^{n-1} C(\tau)\right], \tag{14}$$

where $\varphi_1$, $\varphi_2$, and $\varphi_3$ are the weights of immediate energy consumption, immediate processing delay, immediate task's dropping probability, respectively. We further derive $E(\tau)$, $T(\tau)$, and $D(\tau)$ as follows:

(1) *Immediate Processing Delay.* When action $a(\tau)$ is taken at state $s(\tau)$, the immediate processing delay is induced by the waiting time at user and edge servers' execution queue, the encryption and transmission time by user, the decryption time by edge servers, and the execution time by user and edge servers.

According to Little's law, the average waiting time $t_u^w$ at user's execution queue can be calculated by equation (15). Therefore, after taking an action $a(\tau) \in \mathbb{A}$, the total local waiting time $tt_u^w$ and total local execution time $tt_u^e$ can be calculated by equations (16) and (17), respectively:

$$t_u^w = \frac{\mathcal{Q}_u^e(\tau)}{f_u \cdot N_u}, \tag{15}$$

$$tt_u^w = \left(a_u^e(\tau) - d_u^e(\tau)\right) * t_u^w, \tag{16}$$

$$tt_u^e = \frac{\left(a_u^e(\tau) - d_u^e(\tau)\right) \cdot W}{f_u \cdot N_u}, \tag{17}$$

where $d_u^e(\tau)$ denotes the number of dropping tasks at user's execution queue. It can be calculated by equation (30).

When $a_{c,i}^e(\tau)$ tasks are scheduled to the $\tau$th edge server's execution queue, tasks first need to employ security services to encrypted security-critical data. Let

$tt_c^s$ denote the total encryption time of $a_c^e(\tau)$ tasks. Therefore, $tt_c^s$ can be calculated by equation (18). And then, these encrypted tasks are transmitted to the edge servers by the wireless channel. Let $tt_c^{tr}$ denotes the total transmission time. Therefore, $tt_c^{tr}$ can be calculated by equation (19). Next, these encrypted tasks are received and stored in its execution queue by edge servers. Due to insufficient queue space, $d_{c,i}^e(\tau)$ tasks are dropped. The number $d_{c,i}^e(\tau)$ of dropping tasks can be calculated by equation (31). The remaining $(a_{c,i}^e(\tau) - d_{c,i}^e(\tau))$ tasks wait to be executed in execution queue. The total waiting time $tt_c^w$ can be calculated by equation (20). At last, they are further decrypted and executed. Let $tt_c^{ds}$ and $tt_c^e$ denote the total encryption time and the total execution time, respectively. Therefore, they can be calculated by equations (22) and (23):

$$tt_c^s = \sum_{i=1}^{n}\left(a_{c,i}^e(\tau)\cdot T_{c,i}^E\right), \tag{18}$$

$$tt_c^{tr} = \sum_{i=1}^{n}\frac{a_{c,i}^e(\tau)\cdot D_{tx}}{R_i(\tau)}, \tag{19}$$

$$tt_c^w = \sum_{i=1}^{n}\left(a_{c,i}^e(\tau) - d_{c,i}^e(\tau)\right)\cdot\frac{\mathbb{Q}_u^e(\tau)}{\left(f_{c,i}\cdot N_{c,i}/W\right)}, \tag{20}$$

$$tt_{c,i}^{ds} = \left(a_{c,i}^e(\tau) - d_{c,i}^e(\tau)\right)\cdot\frac{\left(T_{c,i}^{\mathrm{DE}}\cdot f_u\cdot N_u\right)}{\left(f_{c,i}\cdot N_{c,i}\right)}, \tag{21}$$

$$tt_c^{ds} = \sum_{i=1}^{n} tt_{c,i}^{ds}, \tag{22}$$

$$tt_c^e = \sum_{i=1}^{n}\left(a_{c,i}^e(\tau) - d_{c,i}^e(\tau)\right)\cdot\frac{W}{\left(f_{c,i}\cdot N_{c,i}\right)}. \tag{23}$$

The total immediate processing time $T(\tau)$ at the $k$th time slot is defined as an average value of the sum of the encryption time, the transmission time, the waiting time, the execution time, and the decryption time. Based on the aforementioned values, $T(\tau)$ can be calculated by the following equation:

$$T(\tau) = \frac{\left(tt_u^w + tt_u^e + tt_c^s + tt_c^{tr} + tt_c^w + tt_c^{ds} + tt_c^e\right)}{m(a(\tau))}. \tag{24}$$

(2) *Immediate Energy Consumption.* When action $a(\tau)$ is taken at state $s(\tau)$, the immediate mobile device's energy is consumed by executing tasks locally, encrypting offloaded tasks and transmitting the offloaded tasks to edge server. We define the immediate energy consumption as the average energy consumed by performing action $a(\tau)$.

When the user decides to execute $a_u^e(\tau)$ tasks locally, the local execution energy consumption $E_u^e(\tau)$ can be calculated by the following equation:

$$E_u^e(\tau) = P_u^e\cdot tt_u^e. \tag{25}$$

When $m(a(\tau))$ number of tasks are to be offloaded to $n$ edge servers, the encryption energy consumption $E_c^s(\tau)$ and the transmission energy consumption $E_c^{tr}(\tau)$ can be calculated by equations (26) and (27), respectively:

$$E_c^s(\tau) = P_u^e\cdot tt_u^s, \tag{26}$$

$$E_c^{tr}(\tau) = P_u^{tx}\cdot tt_c^{tr}. \tag{27}$$

Therefore, the immediate energy consumption $E(\tau)$ can be denoted by the following equation:

$$E(\tau) = \frac{\left(E_u^e(\tau) + E_u^s(\tau) + E_c^{tr}(\tau)\right)}{m(a(\tau))}. \tag{28}$$

(3) *Immediate Task Dropping Number.* The arriving tasks will be dropped if the mobile device and edge servers' execution queue are full or have insufficient space at the $\tau$th time slot. Let $D(\tau)$ denote the number of dropped tasks. It can be calculated by the following equation:

$$D(\tau) = d_u^e(\tau) + \sum_{i=1}^{n} d_{c,i}^e(\tau), \tag{29}$$

$$d_u^e(\tau) = \max\left(0, \left(a_u^e(\tau) - \mathbb{Q}_u^{\mathrm{ava}}(\tau)\right)\right), \tag{30}$$

$$d_{c,i}^e(\tau) = \max\left(0, \left(a_{c,i}^e(\tau) - \mathbb{Q}_{c,i}^{\mathrm{ava}}(\tau)\right)\right), \tag{31}$$

where $\mathbb{Q}_u^{\mathrm{ava}}(\tau) = |\mathbb{Q}_u^e(\tau)| - \mathbb{Q}_u^e(\tau) + e_u(\tau)$ denotes the residual available space of the mobile device's execution queue. $e_u(\tau) = \min\left((T_{\mathrm{slot}} - tt_c^s - tt_c^{tr})\cdot(f_u\cdot N_u/W), \mathbb{Q}_u^e(\tau)\right)$ denotes the number of tasks actually executed for the mobile device's execution queue at the $\tau$th time slot. $\mathbb{Q}_{c,i}^{\mathrm{ava}}(\tau) = |\mathbb{Q}_{c,i}^e(\tau)| - \mathbb{Q}_{c,i}^e(\tau) + e_{c,i}(\tau)$ denotes the residual available space of the $i$th edge server. $e_{c,i}(\tau) = \min\left((T_{\mathrm{slot}} - tt_{c,i}^{ds})\cdot(f_{c,i}\cdot N_{c,i}/W), \mathbb{Q}_{c,i}^e(\tau)\right)$ denotes the number of tasks actually executed at the $i$th edge server at the $\tau$th time slot.

*4.4. Problem Formulation.* The objective of this paper is to find an offloading policy $\pi$, which minimizes the average long-term cost over the infinite time horizon while meeting the risk probability constraint. Thus, the problem can be formally formulated as

$$\text{minimize} \quad \overline{C}, \tag{32}$$

$$\text{subject to} \quad P(\tau) \le P_{\max}, \tag{33}$$

$$tt_c^s + t_c^{tr} \le T_{\mathrm{slot}}, \tag{34}$$

where equation (32) is the objective of this paper, equation (33) is the risk probability constraint, and equation (34)

indicates that the task transmission and security service should be conducted in each time slot.

Lacking of a prior knowledge of the task arrival and channel state, it is very difficult to solve this optimization problem by traditional methods. Fortunately, without the prior knowledge of network state transition, deep Q-network (DQN) can solve this kind of stochastic optimization problem. In the next section, a SCACO strategy based on deep Q-network is introduced to solve our security-aware computation offloading problem.

## 5. Algorithm Implementation

The formulated computation offloading optimization problem in Section 4 is essentially an infinite-horizon Markov decision process with the discounted cost criterion. To solve the optimal computation offloading policy, we propose a SCACO scheme based on deep Q-network (DQN) [40]. In this section, we first introduce the traditional solutions for the Markov decision process and then introduce the SCACO strategy based on deep Q-network.

*5.1. Optimal MDP Solution.* To solve the optimal policy $\pi^*$, an optimal mapping from any state $s(\tau)$ to the optimal action $a(\tau)$ need to be achieved. Since the optimal state-value function $V_{\pi^*}(s)$ can be achieved by solving Bellman's optimality equation, Bellman's optimality equation of the formulated computation offloading problem is defined by the following equation:

$$V_{\pi^*}(s) = \max_{a\in\mathbb{A}}\left\{ C(s,a) + \gamma \sum_{s\_\in\mathbb{S}} p(s\_ \mid s,a) V_{\pi^*}(s\_) \right\}. \tag{35}$$

According to Bellman's optimality equation, the optimal policy of state $s(\tau)$ can be obtained by the following equation:

$$\pi^*(s) = \arg\max_{a\in\mathbb{A}}\left\{ C(s,a) + \gamma \sum_{s\_\in\mathbb{S}} p(s\_ \mid s,a) V_{\pi^*}(s\_) \right\}. \tag{36}$$

The traditional solutions to solve Bellman's optimality equation are based on the value or the policy iteration [41]. These two solutions usually need complete knowledge of the system-state transition probabilities. However, these knowledges are difficult to obtain in advance in the dynamic system. Moreover, the network state-space with even a reasonable number of edge servers is extremely huge. Facing these problems, the two traditional solutions are inefficient. Thus, the DQN-based learning scheme which is a model-free reinforcement learning method is introduced to approach the optimal policy.

*5.2. DQN-Based Offloading Decision Algorithm.* The optimal action-value function $\mathcal{Q}^*(s,a)$ which is on the right-hand side of equation (35) can be defined as follows:

$$\mathcal{Q}^*(s,a) = \max_{a\in\mathbb{A}}\left\{ C(s,a) + \gamma \sum_{s\_\in\mathbb{S}} p(s\_ \mid s,a) V_{\pi^*}(s\_) \right\}. \tag{37}$$

To address these challenges mentioned in Section 5.1, a model-free deep reinforcement learning algorithm called deep $\mathcal{Q}$-network (DQN) is proposed. A deep neural network can be used to approach the optimal action-value function $\mathcal{Q}^*(s,a)$ without any information of dynamic network statistics. The DQN-based offloading decision algorithm can be illustrated in Figure 4.

At each time slot $\tau$, the system state $s(\tau) = (\mathcal{Q}_{\text{arr}}(\tau), \mathcal{Q}_u^e(\tau), R(\tau), \mathcal{Q}_c^e(\tau))$ is first fed into the neural network. Based on the system state, the $\mathcal{Q}$_values $\mathcal{Q}(s(\tau), \cdot)$ for all possible actions $a(\tau) \in \mathbb{A}$ are obtained. Then, the action $a(\tau)$ corresponding to the state $s(\tau)$ can be selected according to the $\varepsilon$-greedy method, and the immediate cost can be calculated by using equation (13). Next, the resulting system state $s\_(\tau+1)$ at the next time slot $(\tau+1)$ can be observed. And last, the transition experience $\text{tr}(\tau) = (s(\tau), a(\tau), C(s(\tau), a(\tau)), s\_(\tau+1))$ can be obtained and stored into replay memory of a finite size. The replay memory can be denoted by $\Omega^\tau = \{m^{\tau-U+1}, m^{\tau-U+2}, \ldots, m^\tau\}$ at $\tau$th time slot, where $U$ is the size of the replay memory. To address the instability of the $\mathcal{Q}$-network due to the use of nonlinear approximation function, the experience replay technique is adopted as the training method. At the end of each episode $k$, a minibatch $\widetilde{\Omega}^k \subset \Omega^k$ of transitions from the replay memory are randomly chosen to train the $\mathcal{Q}$-network in the direction of minimizing a sequence of the loss functions, where the loss function at time step $k$ is defined as

$$L(\theta_k) = \mathbb{E}\left[ C\Big( s(\tau), a(\tau)) + \gamma \max_{a(\tau)\in\mathbb{A}} \mathcal{Q}(s(\tau+1), a(\tau+1); \theta_{k+1}) - \mathcal{Q}(s(\tau), a(\tau); \theta_k) \Big)^2 \right]. \tag{38}$$

In other words, given a transition $\text{tr}(\tau) = (s(\tau), a(\tau), C(s(\tau), a(\tau)), s\_(\tau+1))$, the weight $\theta$ of the $\mathcal{Q}$-network are updated in a way that the squared error loss is minimized between the current predicted $\mathcal{Q}$ value $\mathcal{Q}(s(\tau), a(\tau); \theta_k)$ and the target $\mathcal{Q}$ value $C(s(\tau), a(\tau)) + \gamma \max_{a(\tau)\in\mathbb{A}} \mathcal{Q}(s\_(\tau+1), a(\tau+); \theta_{k+1})$.

The DQN-based offloading decision algorithm is described in Algorithm 1 in detail.

## 6. Experiments

In this section, we perform simulation experiments to demonstrate the effectiveness of our proposed SCACO strategy. First, we present our experiment parameters setup. Then, we analyze the performance of our proposed strategy with the varying of different parameters.

*6.1. Experiment Parameters.* In this section, to evaluate the performance of the proposed SCACO strategy, we

Begin
(1) Initialize the replay memory $\Omega$ with a size of $U$, the minibatch $\tilde{\Omega} \subset \Omega$ with a size of $\tilde{U}$.
(2) **for** $k = 1, 2, \ldots$ **do**
(3) At beginning of the $\tau$th time slot, observe the system state $s(\tau) \in \mathbb{S}$
(4) Select an action $a(\tau) \in \mathbb{A}$ randomly with probability $\varepsilon$ or $y^k = \arg\max_{a(\tau)\in\mathbb{A}} \mathcal{Q}(s(\tau), a(\tau); \theta_\tau)$ with probability $1 - \varepsilon$
(5) Offload the tasks according to action $a(\tau)$ and observe the cost $C(s(\tau), a(\tau))$ and the new system state $s_-(\tau) \in \mathbb{S}$ at the next time slot $\tau + 1$.
(6) Store the transition experience $(s(\tau), a(\tau), C(s(\tau), a(\tau)), s_-(\tau + 1))$ into replay memory $\Omega$
(7) Randomly sample a minibatch of transition experience $\tilde{\Omega}^k$ from replay memory $\Omega^k$
(8) Train the $\mathcal{Q}$-network based on the loss function of the selected transition experiences
(9) Calculate the loss value between the current predicted $\mathcal{Q}$ value $\mathcal{Q}(s(\tau), a(\tau); \theta_k)$ and the target $\mathcal{Q}$ value $C(s(\tau), a(\tau)) + \gamma \max_{a(\tau)\in\mathbb{A}} \mathcal{Q}(s_-(\tau + 1), a(\tau + 1))$
(10) **end for**
(11) Record the set of optimal weights $\theta$
End

Algorithm 1: DQN-based computation task offloading algorithm.

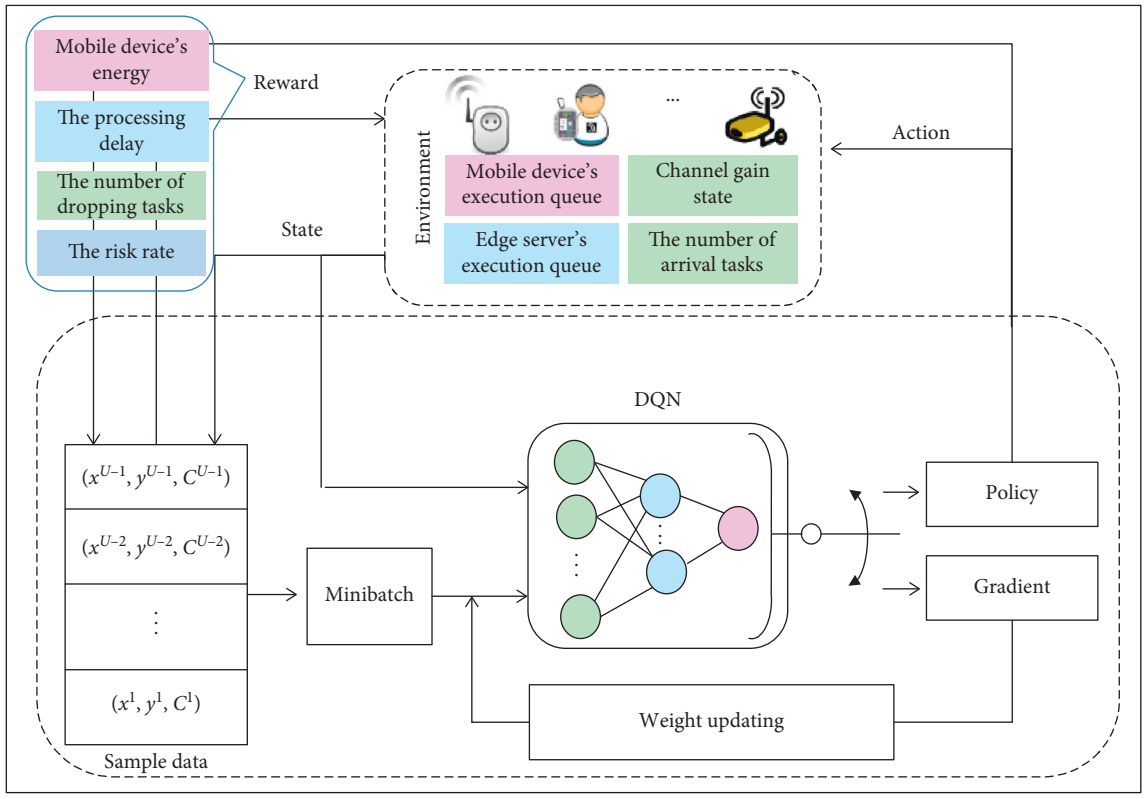


Figure 4: DQN-based offloading decision algorithm.

implement and simulate our strategy on Python 3.6 using a Dell R530 server configured with one CPU (2.2 GHz 8 cores). We set the experimental parameters referring to the literatures [2, 25, 42, 43]. Major simulation parameters are described in detail as follows.

In this paper, we mainly consider the scenario where a mobile device safely and efficiently offloads computation tasks to $n$ edge servers. Initially, the mobile device's CPU frequency $f_u$ is set to $f_u = 2.5$ GHz, and the number $N_u$ of its processor cores is set to $N_u = 4$. The mobile device's computation power $P_e$ (in Watt), transmitting power $P_{tx}$, and receiving power $P_{rx}$ are set to $P_e = 5$ W, $P_{tx} = 0.25$ W, and $P_{rx} = 0.1$ W, respectively. The mobile device execution queue's size is set to $|\mathcal{Q}_u^e| = 6$. We assume that there are 2 edge servers. The CPU frequencies of these two edge servers are set $f_c^1 = f_c^2 = 2.5$ GHz. The processor cores of them are $N_c^1 = N_c^2 = 6$. The execution queues' size of them is set to $|\mathcal{Q}_c^1| = |\mathcal{Q}_c^2| = 10$. Moreover, the risk coefficients of confidentiality service for these two edge servers are set $\lambda_{cf}^1 = 2.4$ and $\lambda_{cf}^2 = 2.7$, respectively. The risk coefficients of integrity

service for these two edge servers are $\lambda_{ig}^1 = 1.5$ and $\lambda_{ig}^2 = 1.8$, respectively.

For the communication model, the transmission power of the edge server is $P_{e\mathrm{NB}} = 40$ W, the maximum bandwidth is $B_i = 100$ MHz, the wireless channel gain is $G_i = [0.3, 4.5]$, and the Gaussian white noise power is $\sigma^2 = -174$ dbm/Hz.

For the computation task model, we assume that the tasks' arrival process follows a Poisson distribution with a parameter $\lambda_{\mathrm{task}} = 10$, which is considered as the estimated value of the average number of arrived tasks during a period $T_{\mathrm{slot}} = 1$ S. The task input is set to $D_{tx} = 0.1$ MB. The task's workload is $W = 2.5$G Hz·S. In addition, the maximum risk rate for task execution is set to $P_{\max} = 0.4$.

The weights of energy consumption, processing delay, and task's dropping probability are $\varphi_1 = 8$, $\varphi_2 = 0.2$, and $\varphi_3 = 3$, respectively.

In the DQN-based learning algorithm, the discount factor is $\gamma = 0.9$, the replay memory is assumed to have a capacity of $U = 5000$, and the size of minibatch is set to $\mathrm{batch}_{\mathrm{size}} = 32$. We implement the DQN learning algorithm based on the TensorFlow APIs and choose a gradient descent optimization algorithm called RMSProp as the optimization method to train the Q-network. During the training, we select the action based on $\varepsilon$-greedy with probability $\varepsilon = 0.1$.

*6.2. Performance Analysis.* To demonstrate the effectiveness of our proposed SCACO scheme, the following peer schemes are conducted on four performance metrics, such as total cost, total energy, total delay, and total number of dropping tasks:

(i) *Local*. At each time slot, the arriving tasks are processed locally so that the risk probability of task execution is 0.

(ii) *Max_Level*. This scheme employs various security services and sets the security level of all security services to the highest level 1.

(iii) *SCACO*. This abbreviation stands for the security and cost-aware computation offloading scheme, the objective of which is to minimize the long-term cost under the risk probability constraints.

*6.2.1. Performance Impact of Risk Rate.* Figure 5 illustrates the performance of three schemes as the risk rate varying from 0.1 to 1.0. Figure 5(a) presents the influence of different risk rates on the long-term cost. As observed from Figure 5(a), the long-term cost obtained by *SCACO* decreases gradually with increasing risk probability, while the curves for the long-term cost of *Local* and *Max_Level* are flat. This is because the security levels of security services gradually decrease with increasing risk probability. With a lower security level, both the task processing time and the mobile device's energy consumption will be less. Consequently, the long-term cost will be reduced as well. The long-term costs of *Local* and *Max_Level* are independent of the risk rate so that their long-term costs are constant and the curves are flat. Moreover, the long-term cost obtained by the *SCACO* scheme is lower than those of *Local* and *Max_Level* schemes. The reason is that a lower security level can meet a higher risk rate constraint. The lower the security service level, the less the task processing time and mobile device's energy consumption, and thereby the less the long-term cost.

Figures 5(b)–5(d) show the energy consumption, delay, and number of dropping tasks for three schemes when the risk rate increases. As shown in Figures 5(b)–5(d), the total energy consumption, the delay, and the number of dropping tasks obtained by *SCACO* decrease gradually with increasing risk rate, while these values of *Local* and *Max_Level* are constant. In addition, we can observe from Figures 5(c) and 5(d) that the energy consumption and the number of dropping tasks obtained by *SCACO* are lower than those by *Local* and *Max_Level*. The reason is the same to the long-term cost above. We can observe from Figure 5(b) that the delay of *Max_Level* is the maximum and that of *Local* is minimium. The delay of *SCACO* is between *Local* and *Max_Level*. The reason is that the optimization of energy consumption is more important than the delay. To reduce the energy consumption, more tasks are offloaded to the edge servers, thereby incurring more longer waiting time.

*6.2.2. Performance Impact of Computing Capacity of Edge Server.* The computing capacity of the edge server is mainly relative to the number of CPU cores. To investigate the impact of computing capacity of the edge server, we vary the CPU cores from 4 to 8 with the increment of 1. Figure 6(a) shows that the long-term costs obtained by *SCACO* and *Max_Level* decrease gradually with increasing CPU cores. This is due to the fact that the more the computing resource, the shorter the tasks' processing delay, which leads to lower long-term cost. However, when the number of CPU cores is equal to or larger than 8, the long-term costs of *SCACO* and *Max_Level* become stable and no longer increase. The main reason is that the arriving tasks can be processed timely under this computing capacity. Specially, the long-term cost of *SCACO* is lower than that of *Max_Level*. That is because that the security level of *SCACO* is lower than that of *Max_Level*. Moreover, the curve of *Local* is flat. That is because *Local* executes all tasks locally and it is independent of edge server's computing capacity.

Figures 6(b)–6(d) show that the energy consumptions, delay, and number of dropping tasks for *SCACO* and *Max_Level* decrease when the number of CPU cores increases, while these values obtained by the *Local* scheme are constant. The reason is the same to the long-term cost above. Moreover, as shown in Figures 6(c) and 6(d), the energy consumption and number of dropping tasks for *SCACO* are lower than those of *Local*. It is due to the fact that *SCACO* employs a lower security level than *Max_Level* while meeting the risk rate. We can further observe from Figure 6(b) that the delay of *Max_Level* is the maximum and that of *Local* is minimum. The delay of *SCACO* is between *Local* and *Max_Level*. The reason is that the optimization of energy consumption is more important than that of the delay. To reduce the energy consumption, more tasks are offloaded to

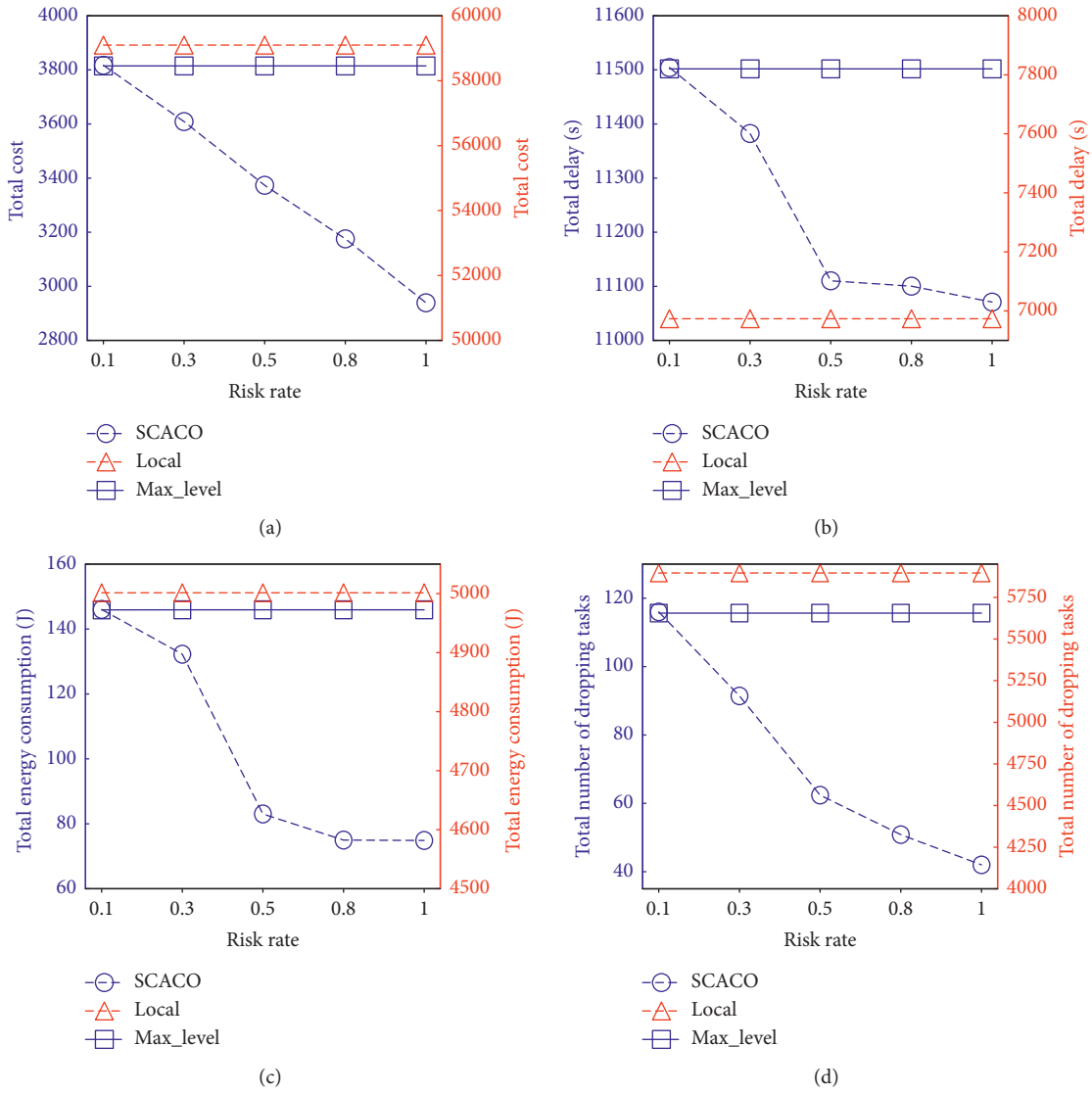


Figure 5: Reward (a), energy consumption (b), delay (c), and the number of dropping tasks (d) w.r.t. different risk rates.

the edge servers, thereby incurring more longer waiting time.

*6.2.3. Performance Impact of Task Workload.* To examine the influence of different task workloads on the long-term cost, we vary the value of task workload from 1.5 to 3.5 with the increment of 0.5. Figure 7 illustrates the long-term cost of three schemes. As shown in Figure 7(a), the long-term costs of three schemes increase with increasing task workload. The reason is that the larger the task workload, the larger the processing time and energy consumption, and thereby the larger the long-term cost. Moreover, *SCACO* shows a lower cost than *Local* and *Max_Level* schemes. That is because the optimization objective of *SCACO* is to minimize the long-term cost while satisfying the risk rate constraints.

Figures 7(b)–7(d) show that the energy consumptions, delay, and number of dropping tasks for three schemes increase gradually with increasing task workload. Especially, as shown in Figures 7(c) and 7(d), the energy consumption and number of dropping tasks obtained by *SCACO* are lower than those of *Local* and *Max_Level*. The reason is the same to the long-term cost above. However, we can further observe from Figure 7(b) that the delay obtained by *SCAC* is between *Local* and *Max_Level*. The reason is the same as the previous section.

*6.2.4. Performance Impact of Task Data Size.* Figure 8 illustrates the impact of task data size on the performance. We discuss about the performance of three schemes when the task data size is varied from 0.1 to 0.5 with the increment of 0.1. As observed from Figure 8(a), the long-term costs of *SCACO* and *Max_Level* schemes increase gradually with increasing task data size. It is because that the larger the task data size, the longer the tasks' processing delay and the higher the mobile device's energy consumption, which leads

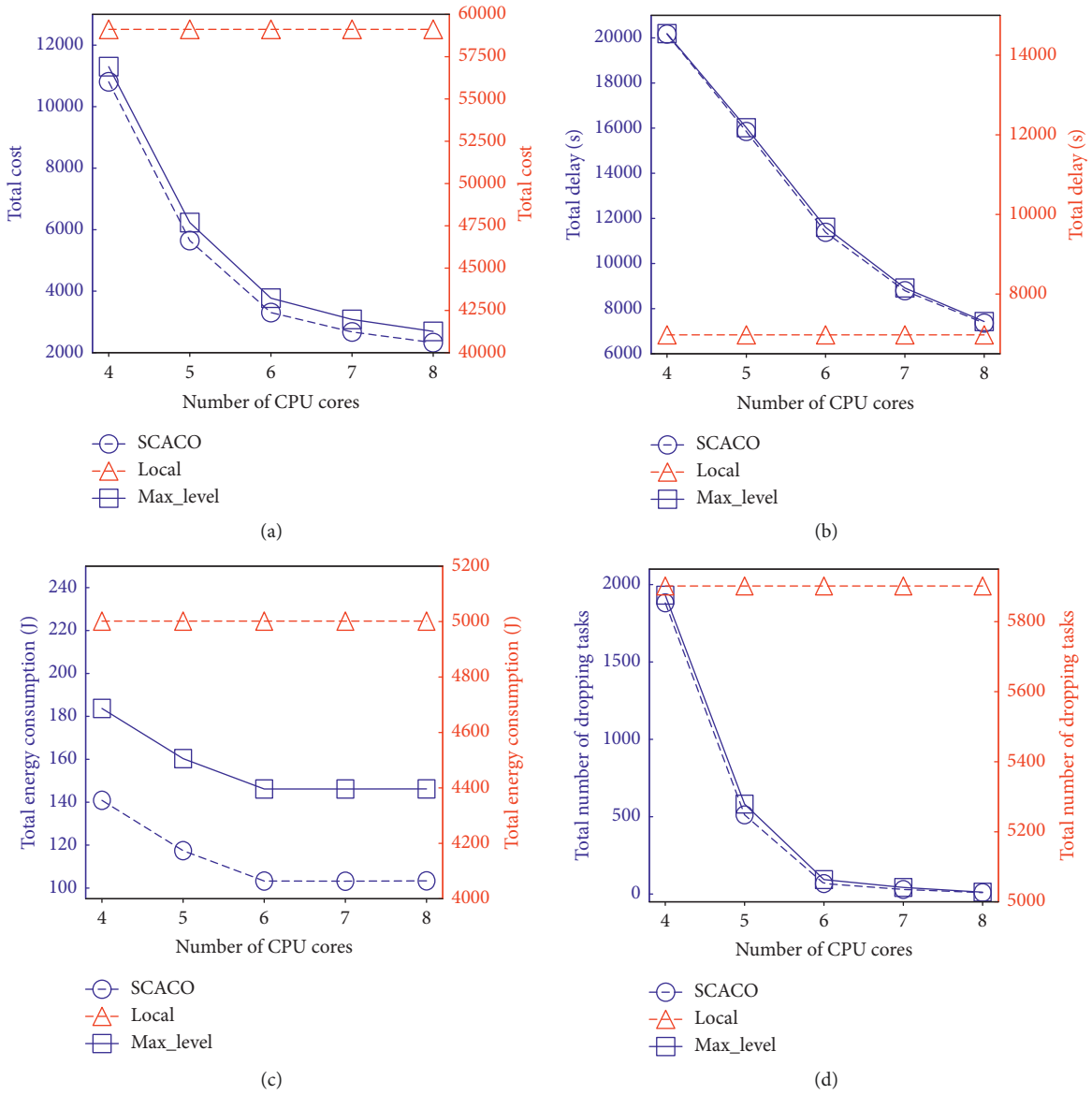


Figure 6: Reward (a), energy consumption (b), delay (c), and the number of dropping tasks (d) w.r.t. different CPU cores.

to higher long-term cost. Moreover, the long-term cost of *SCACO* is lower than that of *Max_Level*. The reason is that comparing with *Max_Level* scheme, a lower security level is selected by *SCACO* while satisfying the risk constraint. Finally, the curve of the *Local* scheme is flat. It is due to that the *Local* scheme executes all tasks locally, and it is independent of task data size.

Figures 8(b)–8(d) show that the energy consumption, delay, and number of dropping tasks for *SCACO* and *Max_Level* schemes increase gradually with increasing task data size, while these curves of *Local* scheme are flat. In addition, the energy consumption and number of dropping tasks for *SCACO* are lower than these of *Local* and *Max_Level*, while the delay of *SCACO* is between *Local* and *Max_Level*.

*6.2.5. Performance Impact of Task Arrival Rate.* To investigate the impact of the task arrival rate, the experiments are conducted with the task arrival rate varying from 10 to 18. Figure 9(a) shows the long-term cost which is obtained by three schemes. We observe from Figure 9(a) that the long-term costs of three schemes increase with increasing task arrival rate. That is because over the increase of the task arrival rate, a higher number of tasks need to be processed at each time slot, thereby incurring a higher long-term cost. Moreover, the *SCACO* shows a lower cost than *Local* and *Max_Level* schemes. That is because the main objective of the proposed schemes is to minimize the obtained average long-term cost while satisfying the risk rate constraints.

Figure 9(b) shows that the delay of *SCACO* and *Max_Level* schemes gradually increases with increasing task

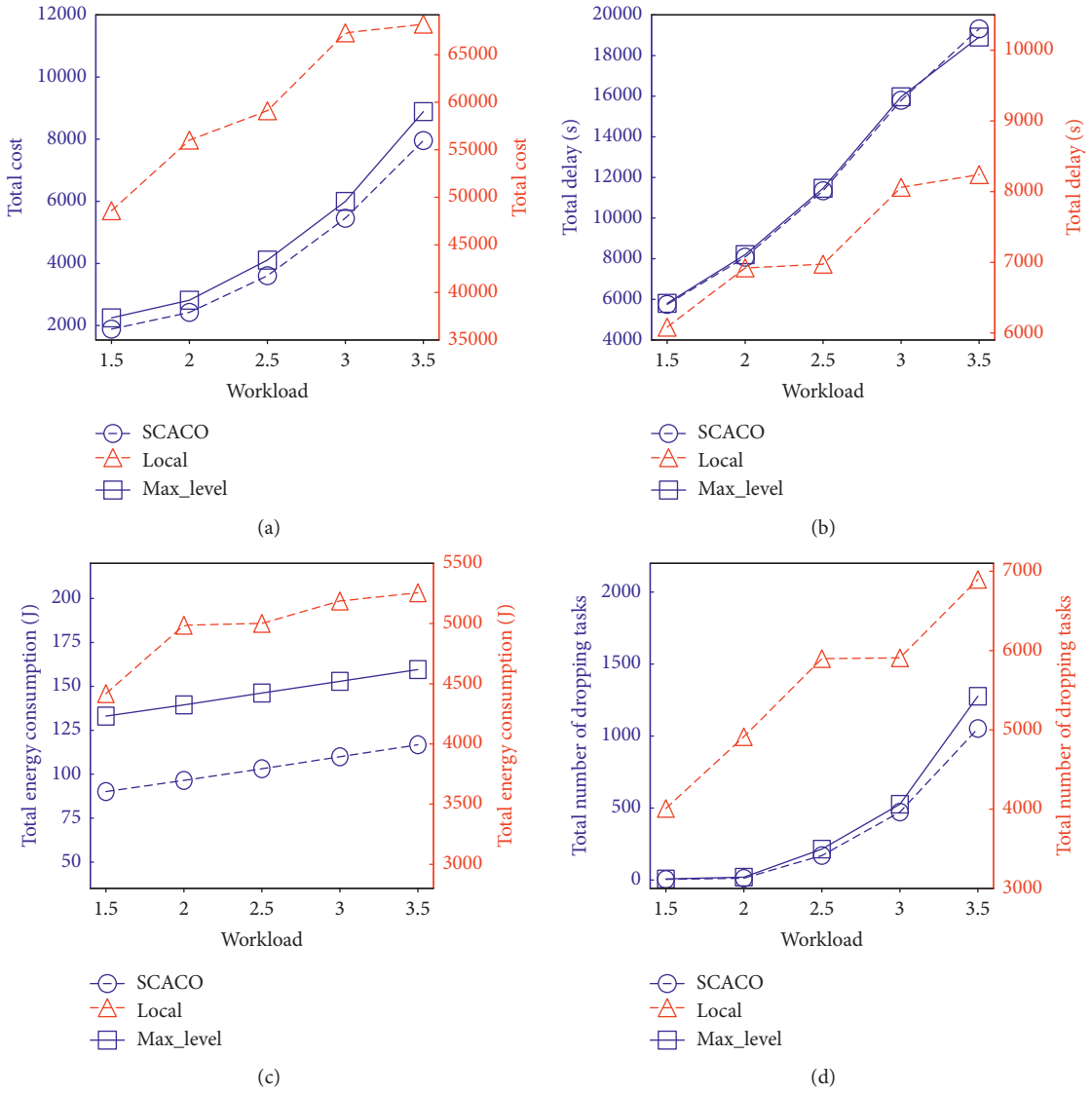


Figure 7: Reward (a), energy consumption (b), delay (c), and the number of dropping tasks (d) w.r.t. different workloads.

arrival rate, while the delay of *Local* is constant. That is because the higher the task arrival rate, the more the tasks stored at the user's and edge servers' execution queue, and thereby the longer the waiting time. In addition, the delay of *SCACO* is between *Local* and *Max_Level*.

As shown in Figure 9(c), we observe that the mobile device's energy consumption of *SCACO* and *Max_Level* gradually increases with the increase of the task arrival rate, while the energy consumption of *Local* is constant. With the increase of the task arrival rate, there are much more tasks which are offloaded to execute remotely. The more the tasks offloaded to the edge servers, the more the encryption energy consumption and transmission energy consumption consumed. Therefore, the mobile device consumes a higher amount of energy. Moreover, *SCACO* has a lower cost than *Local* and *Max_Level* schemes.

Figure 9(d) shows that the number of dropping tasks gradually increases with increasing task arrival rate. The main reason is that the higher the task arrival rate, the higher the number of arrival tasks at a time slot. However, with the limited-size executing queue of the mobile device and edge servers, newly arrived tasks will be dropped due to the lack of queue space. The more the tasks generated, the higher the number of the dropping task.

*6.2.6. Performance Impact of the Number of Edge Server.* Figure 10 illustrates the performance of three schemes when the number of edge servers varying from 2 to 4. As shown in Figure 10, the long-term costs obtained by *SCACO* and *Max_Level* schemes decrease with increasing number of the edge server, while the long-term cost of the *Local* scheme is constant. The reason is that there are more edge servers

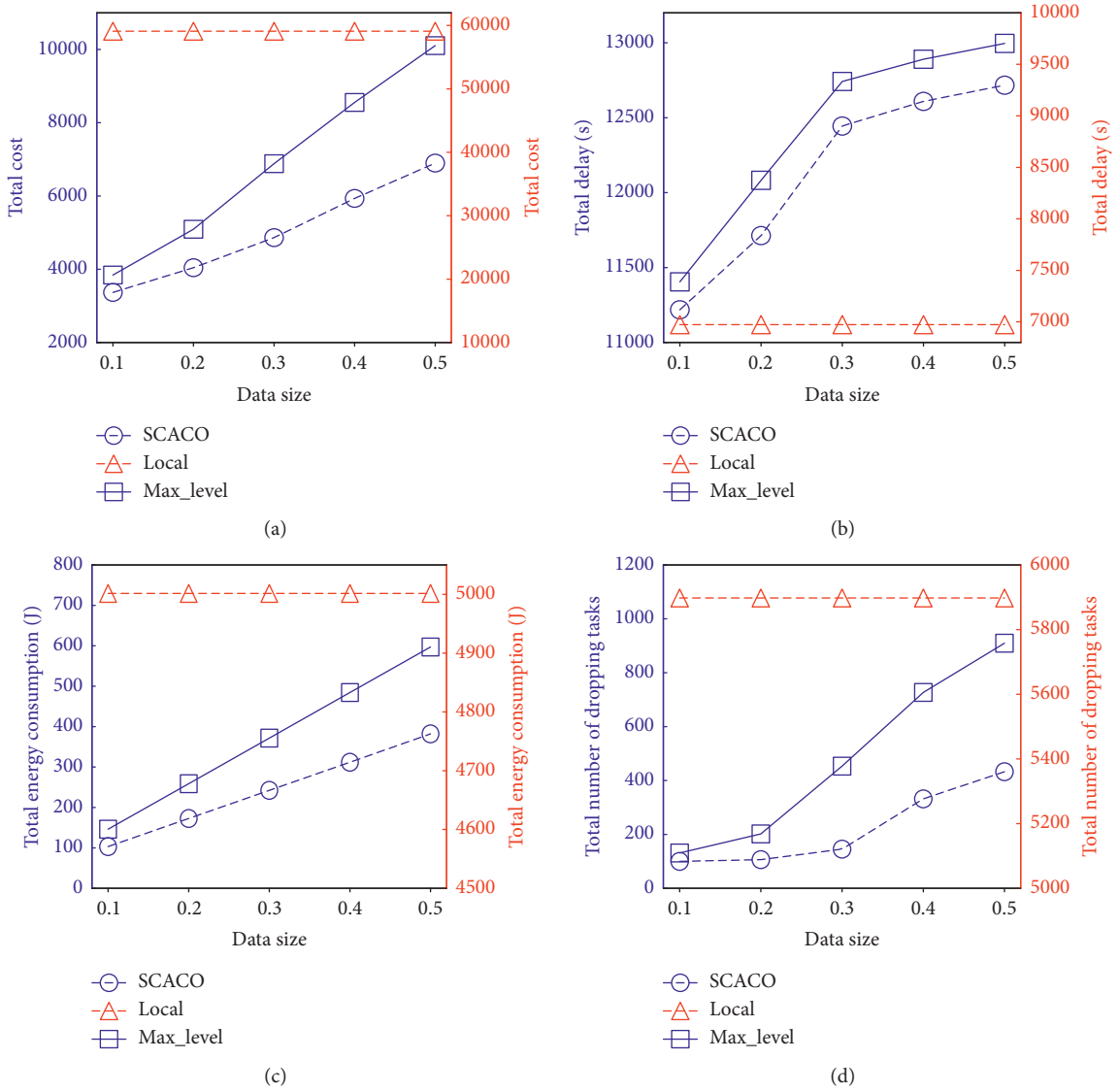


Figure 8: Reward (a), energy consumption (b), delay (c), and the number of dropping tasks (d) w.r.t. different data sizes.

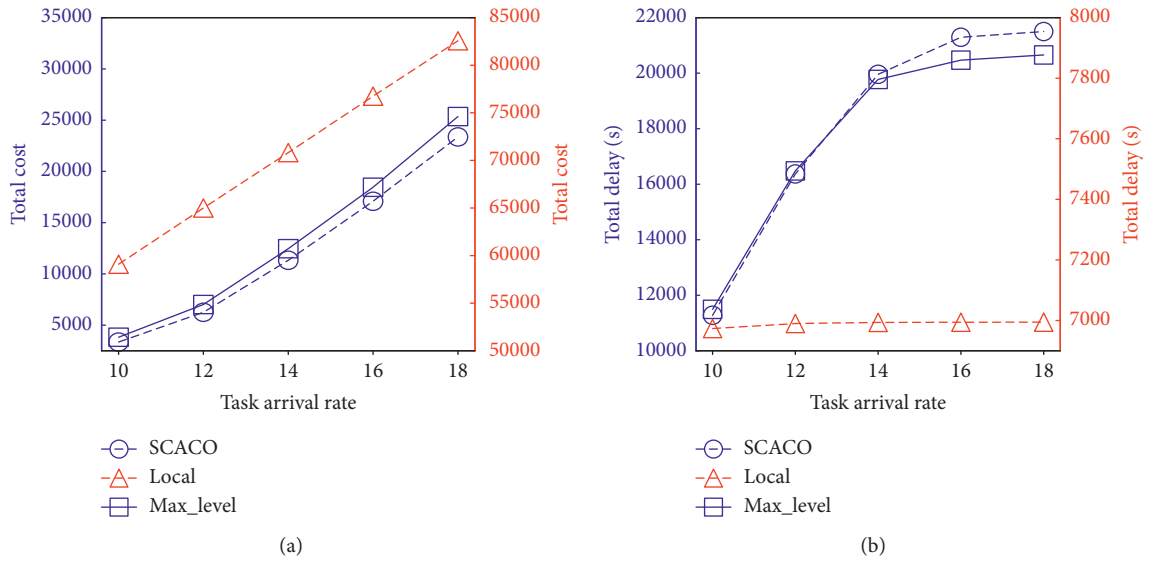


Figure 9: Continued.

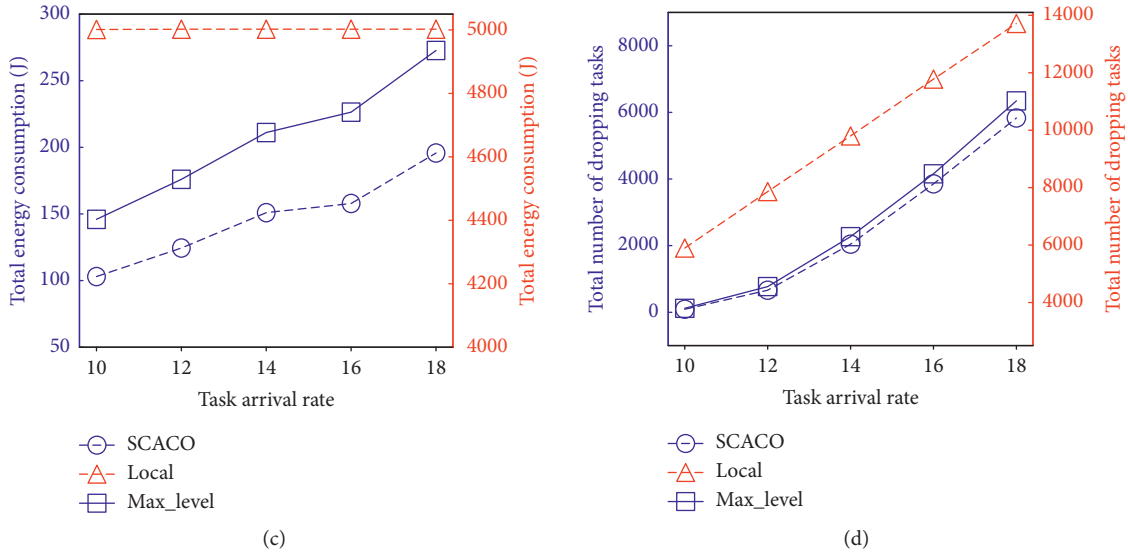


FIGURE 9: Reward (a), energy consumption (b), delay (c), and the number of dropping tasks (d) w.r.t. different task arrival rates.

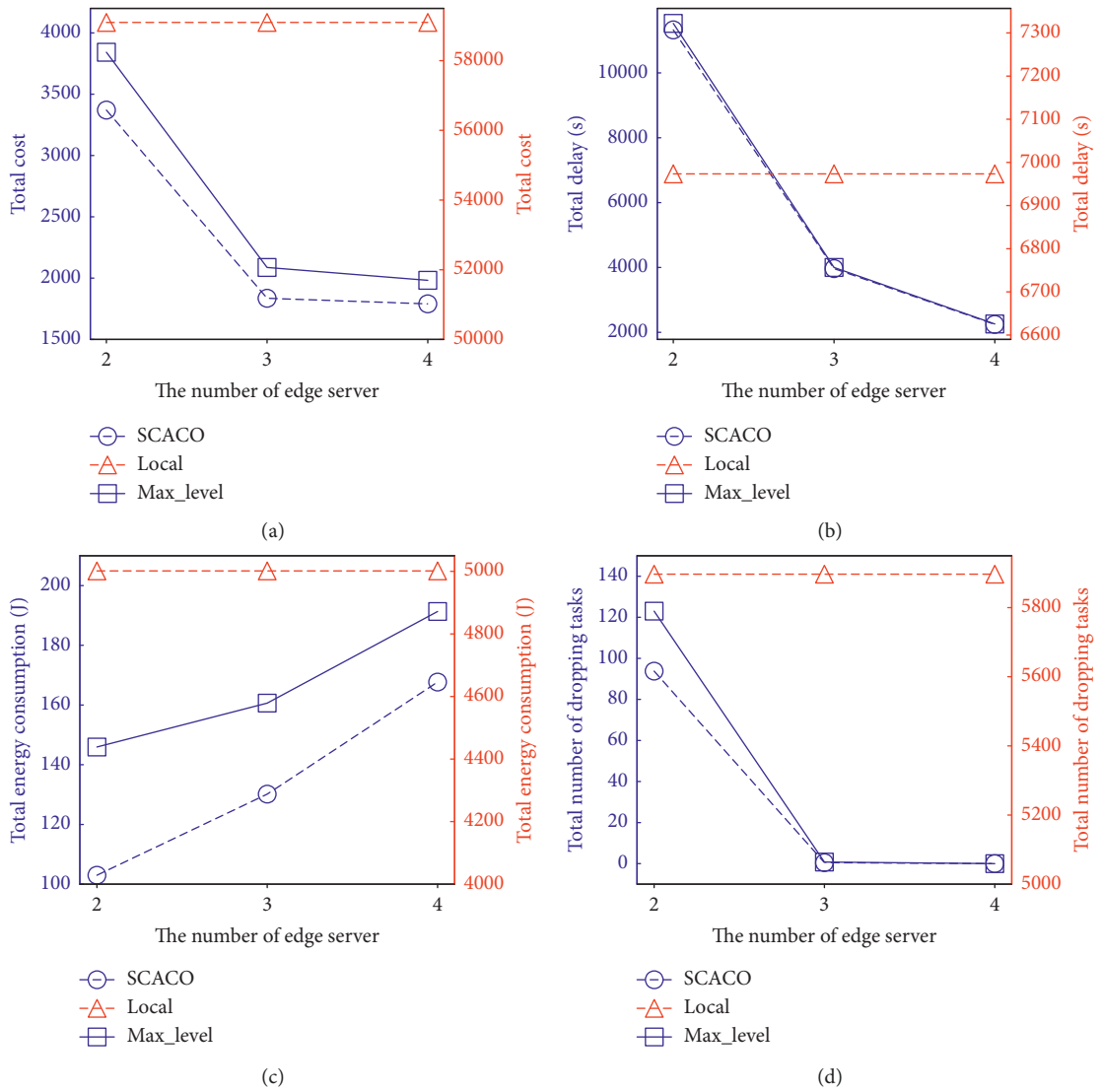


FIGURE 10: Reward (a), energy consumption (b), delay (c), and the number of dropping tasks (d) w.r.t. different number of edge servers.

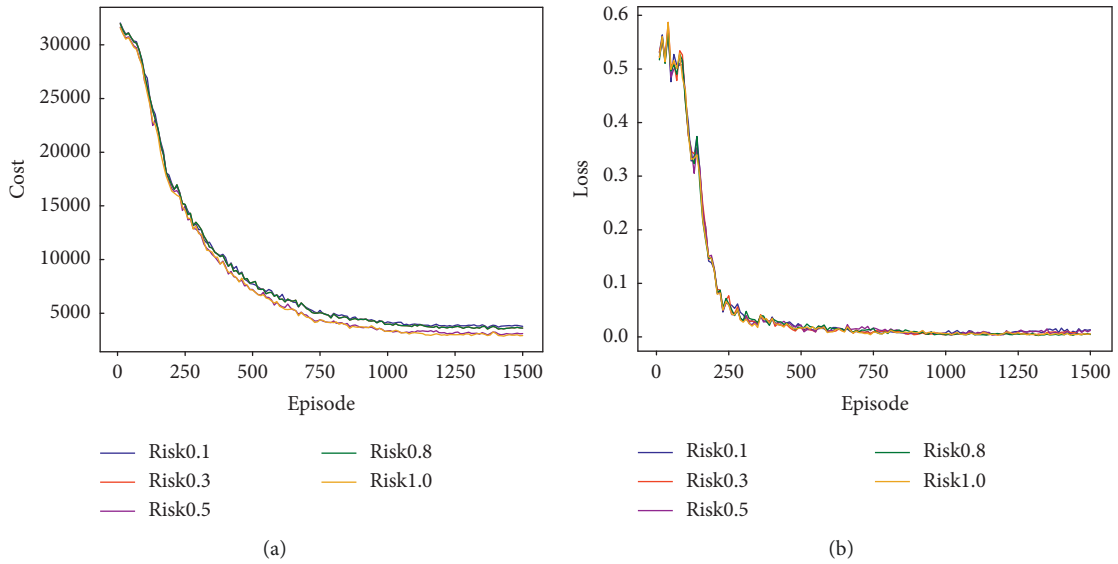


Figure 11: Learning curves w.r.t. different risk rates.

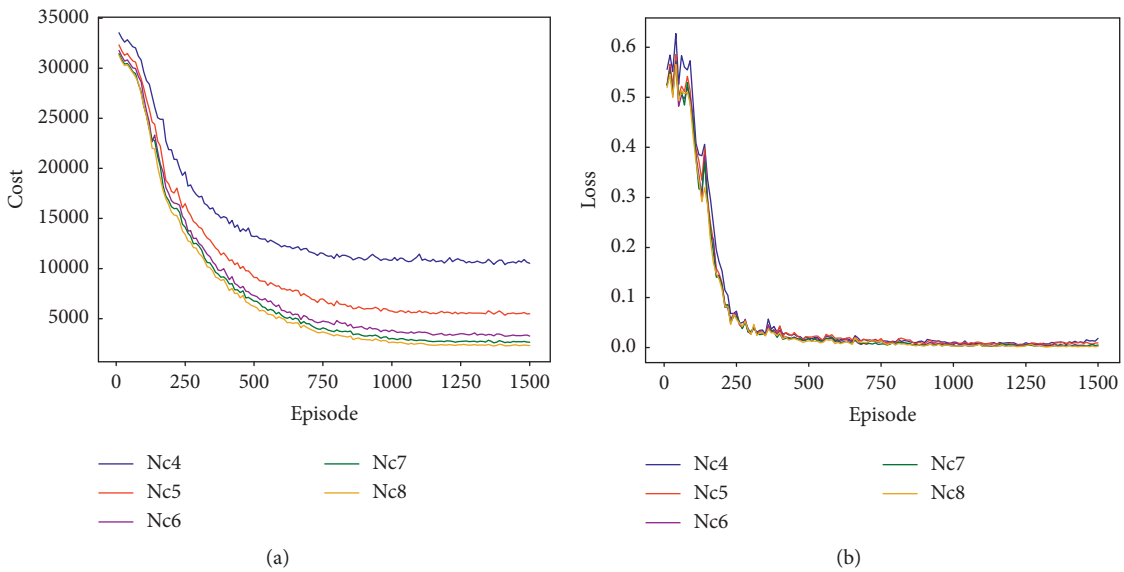


Figure 12: Learning curves w.r.t. different number of CPU cores.

which are available for task offloading. The more the edge servers, the shorter the tasks' processing delay, thereby incurring lower long-term cost. However, the performance of the *Local* scheme does not change with the variation of the number of edge servers since the *Local* scheme executes locally all tasks. Moreover, the long-term cost of *SCACO* is lower than that of *Max_Level*. That is because that *SCACO* employs a lower security level than *Max_Level* while meeting the risk rate.

Figures 10(b)–10(d) exhibit the delay and number of dropping tasks obtained by *SCACO* and *Max_Level* schemes decrease with increasing number of edge servers, while the curves of the *Local* scheme are flat. The reason is that when more edge servers are available, the user offloads its tasks to more edge servers, and the processing delay of tasks and the number of tasks dropping are decreased. Figure 10(c) shows that the energy consumptions of *SCACO* and *Max_Level* increase gradually with increasing number of edge servers. This is due to that the more the edge servers, the lesser the number of tasks dropping, and the more the number of tasks which are executed locally or remotely, thereby the more the energy consumption.

*6.2.7. Performance Impact of SCACO.* Figures 11–16 show the learning curves and loss curves of the *SCACO* scheme over variations of risk rate, computing capacity, task workload, task data size, task arrival rate, and number of

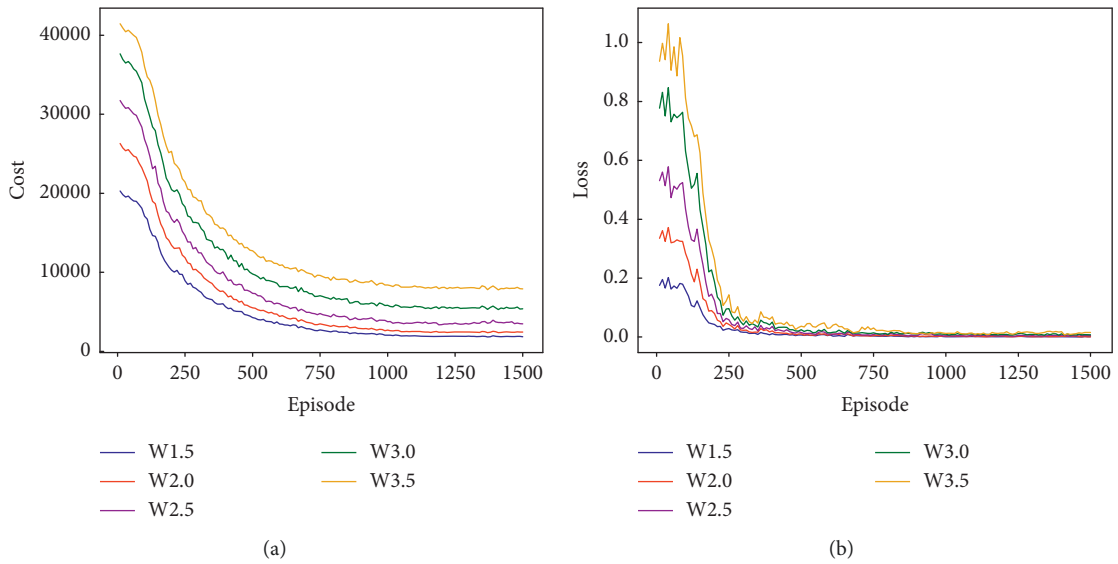


Figure 13: Learning curves w.r.t. different workloads.

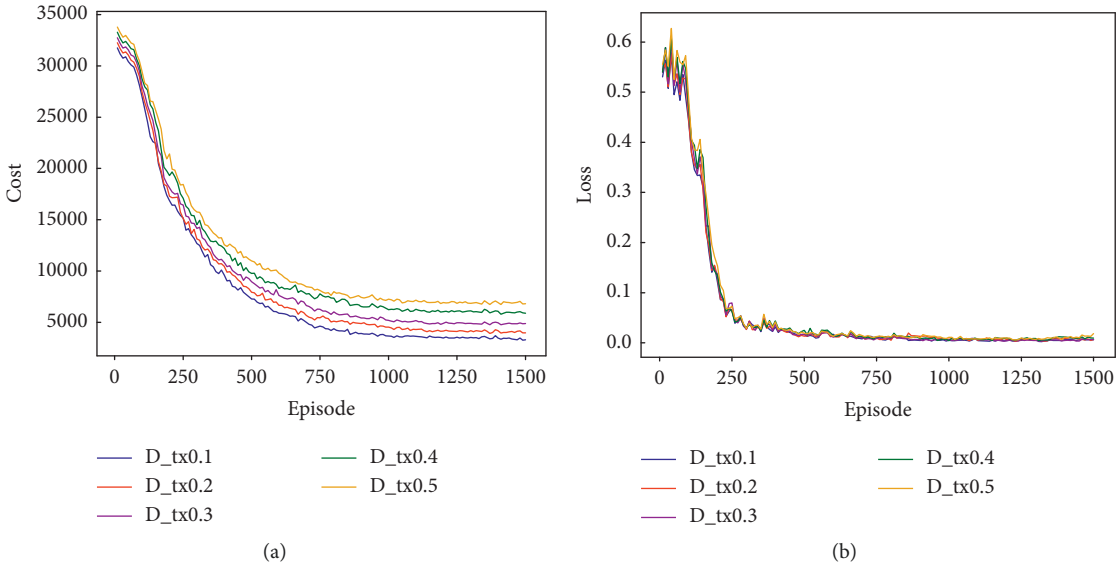


Figure 14: Learning curves w.r.t. different data sizes.

edge servers, respectively. For simplicity, we use Risk0.1, Risk0.3, Risk0.5, Risk0.8, and Risk1.0 to represent the long-term cost of *SCACO* for risk rates 0.1, 0.3, 0.5, 0.8, and 1.0, respectively. Nc4, Nc5, Nc6, Nc7, and Nc8 are used to represent the long-term costs of *SCACO* for the number of CPU cores 4, 5, 6, 7, and 8, respectively. W1.5, W2.0, W2.5, W3.0, and W3.5 are used to represent the long-term costs of *SCACO* for the workloads 1.5, 2.0, 2.5, 3.0, and 3.5, respectively. D_tx0.1, D_tx0.2, D_tx0.3, D_tx0.4, and D_tx0.5 are used to represent the long-term costs of *SCACO* for the task data sizes 0.1, 0.2, 0.3, 0.4, and 0.5, respectively. Lambda_tasks10, lambda_tasks12, lambda_tasks14, lambda_tasks16, and lambda_tasks18 are used to represent the long-term costs of *SCACO* for the task arriving rates 10, 12, 14, 16, and 18, respectively. Server_num2, server_num3, and server_num4 are used to represent the long-term costs of *SCACO* for the number of edge servers 2, 3, and 4, respectively. As shown in Figures 11(a)–16(a), the long-term cost obtained in an episode decreases gradually with increasing learning time (i.e., the number of episodes) from 1 to 1500. Moreover, Figures 11(b)–16(b) show that the loss value of cost decreases gradually with increasing number of episodes. When the episode number is higher than 1000, all learning curves become stable and no longer decrease. This result indicates that the proposed DQN-based learning algorithm converges after 1000 episodes. It means that the

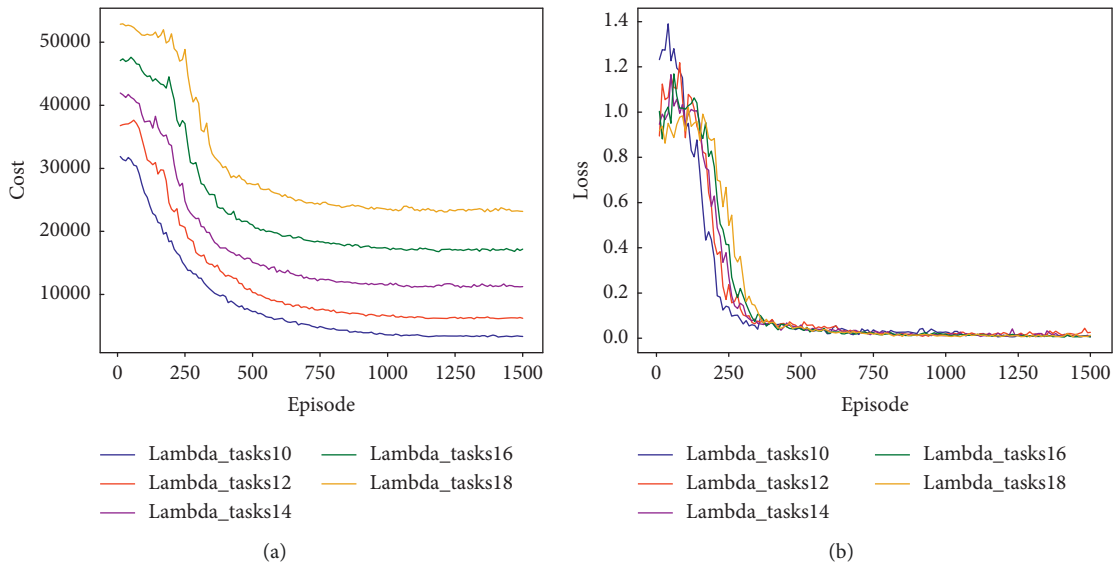


Figure 15: Learning curves w.r.t. different task arrival rates.

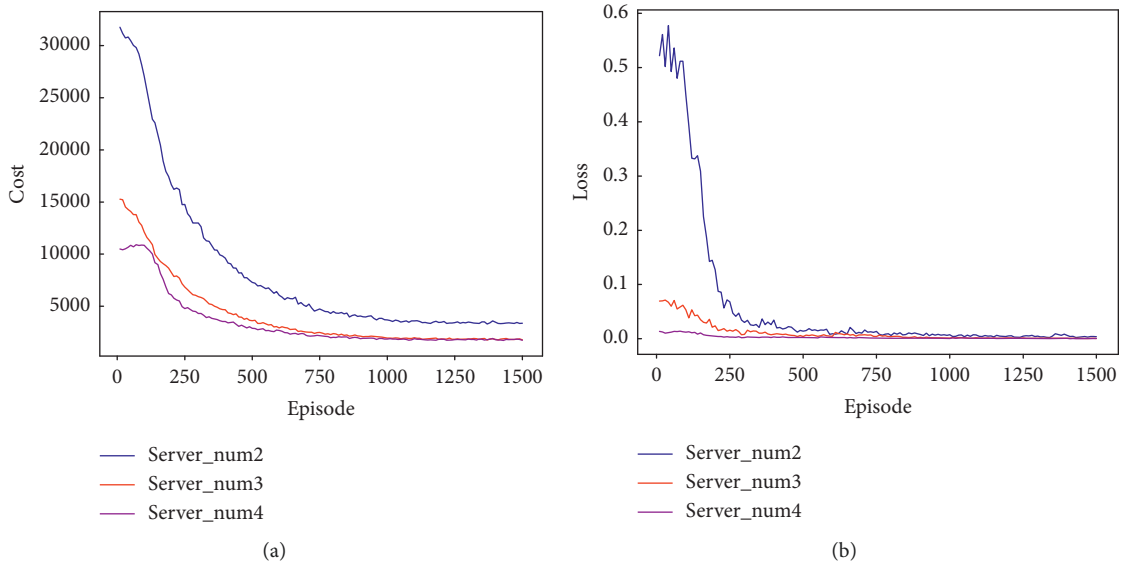


Figure 16: Learning curves w.r.t. different number of edge servers.

proposed DQN-based learning algorithm can learn an optimal strategy after 1000 episodes to minimize the long-term cost.

## 7. Conclusion and Future Work

In this paper, we investigate the security and cost-aware offloading problem and formulate it as a Markov decision process. To find the optimal offloading policy, we propose a security and cost-aware computing offloading (SCACO) strategy based on a deep Q-network (DQN), the objective of which is to minimize the total cost subjecting to the risk rate constraint in mobile edge computing. We evaluate the performance of the proposed offloading scheme under various performance metrics. Our experimental results show that the SCACO strategy can effectively decrease the total cost while the risk rate constraints are satisfied. Especially, the SCACO strategy can achieve the security guard for the security-critical tasks in mobile edge computing. In our experiment, we mainly investigate that the risk rate of security service, security service, risk coefficient, edge servers' computing capacity, tasks workload, task data size, task arrival rate, and the number of edge servers influence on the long-term cost. The extensive experiments demonstrate the effectiveness of the SCACO strategy. In future work, we will further investigate the offloading problem for multiple

mobile devices which offload computation tasks to multiple edge servers.

## Data Availability

The experiment data supporting this experiment analysis are from previously reported studies, which have been cited, and are also included within the article.

## Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

## Acknowledgments

This work was supported by the National Science Foundation of China (Nos. 61802095, 61572162, and 61572251), the Zhejiang Provincial National Science Foundation of China (Nos. LQ19F020011 and LQ17F020003), the Zhejiang Provincial Key Science and Technology Project Foundation (No. 2018C01012), and the Open Foundation of State Key Laboratory of Networking and Switching Technology (Beijing University of Posts and Telecommunications) (No. SKLNST-2019-2-15).

## References

[1] CISCO, *Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Up-Date, 2016–2021*, CISCO, San Jose, CA, USA, White Paper, Document ID: 1454457600805266, 2017.

[2] T. Q. Thinh, J. Tang, Q. D. La, and Q. S. Quek, "Offloading in mobile edge computing: task allocation and computational frequency scaling," *IEEE Transactions on Communications*, vol. 65, no. 8, pp. 3571–3584, 2017.

[3] S. Guo, B. Xiao, and Y. Yang, "Energy-efficient dynamic offloading and resource scheduling in mobile cloud computing," in *Proceedings of the IEEE INFOCOM 2016—The 35th Annual IEEE International Conference on Computer Communications*, IEEE, San Francisco, CA, USA, pp. 1–9, April 2016.

[4] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, 2017.

[5] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "Mobile edge computing: survey and research outlook," 2017, https://arxiv.org/abs/1701.01090.

[6] E. Ahmed, A. Ahmed, I. Yaqoob et al., "Bringing computation closer toward the user network: is edge computing the solution?," *IEEE Communications Magazine*, vol. 55, no. 11, pp. 138–144, 2017.

[7] J. Shuja, S. Mustafa, R. W. Ahmad, S. A. Madani, A. Gani, and M. Khurram Khan, "Analysis of vector code offloading framework in heterogeneous cloud and edge architectures," *IEEE Access*, vol. 5, pp. 24542–24554, 2017.

[8] I. Stojmenovic, S. Wen, X. Huang, and H. Luan, "An overview of fog computing and its security issues," *Concurrency and Computation: Practice and Experience*, vol. 28, no. 10, pp. 2991–3005, 2016.

[9] K. Lee, D. Kim, D. Ha, U. Rajput, and H. Oh, "On security and privacy issues of fog computing supported internet of things environment," in *Proceedings of the 6th International Conference on the Network of the Future*, pp. 1–3, IEEE, Montreal, Canada, October 2015.

[10] S. Yi, Z. Qin, and Q. Li, "Security and privacy issues of fog computing: a survey," in *Proceedings of the International Conference on Wireless Algorithms, Systems, and Applications*, pp. 685–695, Springer, Qufu, Shandong, China, August 2015.

[11] R. Roman, J. Lopez, and M. Mambo, "Mobile edge computing, fog et al.: a survey and analysis of security threats and challenges," *Future Generation Computer Systems*, vol. 78, pp. 680–698, 2018.

[12] S. N. Shirazi, A. Gouglidis, A. Farshad, and D. Hutchison, "The extended cloud: review and analysis of mobile edge computing and fog from a security and resilience perspective," *IEEE Journal on Selected Areas in Communications*, vol. 35, no. 11, pp. 2586–2595, 2017.

[13] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, "Mobile edge computing: a survey," *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 450–465, 2018.

[14] J. Ni, X. Lin, and X. Shen, "Efficient and secure service-oriented authentication supporting network slicing for 5G-enabled IoT," *IEEE Journal on Selected Areas of Communications*, vol. 36, no. 3, pp. 644–657, 2018.

[15] J. Ni, K. Zhang, Y. Yu, X. Lin, and X. Shen, "Providing task allocation and secure deduplication for mobile crowdsensing via fog computing," *IEEE Transactions on Dependable and Secure Computing*, 2018.

[16] Z. Su, Q. Xu, J. Luo, H. Pu, Y. Peng, and R. Lu, "A secure content caching scheme for disaster backup in fog computing enabled mobile social networks," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 10, pp. 4579–4589, 2018.

[17] Y. Chen, Y. Zhang, and S. Maharjan, "Deep learning for secure mobile edge computing," 2017, https://arxiv.org/abs/1709.08025.

[18] Y. Zou, J. Zhu, X. Wang, and L. Hanzo, "A survey on wireless security: technical challenges, recent advances and future trends," 2015, http://arxiv.org/abs/1505.07919.

[19] Y. Wu, L. P. Qian, H. Mao et al., "Secrecy-driven resource management for vehicular computation offloading networks," *IEEE Network*, vol. 32, no. 3, pp. 84–91, 2018.

[20] B. Huang, Z. Li, P. Tang et al., "Security modeling and efficient computation offloading for service workflow in mobile edge computing," *Future Generation Computer Systems*, vol. 97, pp. 755–774, 2019.

[21] J. Liu, Y. Mao, J. Zhang, and K. B. Letaief, "Delay-optimal computation task scheduling for mobile-edge computing systems," in *Proceedings of the IEEE International Symposium on Information Theory*, pp. 1451–1455, Barcelona, Spain, July 2016.

[22] G. Zhang, W. Zhang, Y. Cao, D. Li, and L. Wang, "Energy-delay tradeoff for dynamic offloading in mobile-edge computing system with energy harvesting devices," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 10, pp. 4642–4655, 2018.

[23] R. Deng, R. Lu, C. Lai, and T. Luan, "Towards power consumption-delay tradeoff by workload allocation in cloud-fog computing," in *Proceedings of the 2015 IEEE International Conference on Communications (ICC)*, pp. 3909–3914, IEEE, London, UK, June 2015.

[24] J. Xu, L. Chen, S. Ren, and S. Ren, "Online learning for offloading and autoscaling in energy harvesting mobile edge computing," *IEEE Transactions on Cognitive Communications and Networking*, vol. 3, no. 3, pp. 361–373, 2017.

[25] X. Chen, H. Zhang, C. Wu et al., "Performance optimization in mobile-edge computing via deep reinforcement learning," 2018, https://arxiv.org/abs/1804.00514.

[26] Y. Mao, J. Zhang, and K. B. Letaief, "Dynamic computation offloading for mobile-edge computing with energy harvesting devices," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 12, pp. 3590–3605, 2016.

[27] Y. Mao, J. Zhang, S. H. Song, and K. B. Letaief, "Stochastic joint radio and computational resource management for multi-user mobile-edge computing systems," *IEEE Transactions on Wireless Communications*, vol. 16, no. 9, pp. 5994–6009, 2017.

[28] S. Ranadheera, S. Maghsudi, and E. Hossain, "Mobile edge computation offloading using game theory and reinforcement learning," 2017, https://arxiv.org/abs/1711.09012.

[29] H. Chen, X. Zhu, D. Qiu, L. Liu, and Z. Du, "Scheduling for workflows with security-sensitive intermediate data by selective tasks duplication in clouds," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 9, pp. 2674–2688, 2017.

[30] Z. Li, J. Ge, H. Yang et al., "A security and cost aware scheduling algorithm for heterogeneous tasks of scientific workflow in clouds," *Future Generation Computer Systems*, vol. 65, pp. 140–152, 2016.

[31] L. Zeng, B. Veeravalli, and X. Li, "SABA: a security-aware and budget-aware workflow scheduling strategy in clouds," *Journal of Parallel and Distributed Computing*, vol. 75, pp. 141–151, 2015.

[32] W. Liu, S. Peng, W. Du, W. Wang, and G. S. Zeng, "Security-aware intermediate data placement strategy in scientific cloud workflows," *Knowledge and Information Systems*, vol. 41, no. 2, pp. 423–447, 2014.

[33] Y. Wu, J. Shi, K. Ni et al., "Secrecy-based delay-aware computation offloading via mobile edge computing for internet of things," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4201–4213, 2018.

[34] E. Meskar, T. D. Todd, D. Zhao, and G. Karakostas, "Energy aware offloading for competing users on a shared communication channel," *IEEE Transactions on Mobile Computing*, vol. 16, no. 1, pp. 87–96, 2017.

[35] C. You, K. Huang, H. Chae, and B.-H. Kim, "Energy-efficient resource allocation for mobile-edge computation offloading," *IEEE Transactions on Wireless Communications*, vol. 16, no. 3, pp. 1397–1411, 2017.

[36] Z. Li, J. Ge, C. Li et al., "Energy cost minimization with job security guarantee in Internet data center," *Future Generation Computer Systems*, vol. 73, pp. 63–78, 2017.

[37] T. Xie and X. Qin, "Scheduling security-critical real-time applications on clusters," *IEEE Transactions on Computers*, vol. 55, no. 7, pp. 864–879, 2006.

[38] T. Xie and X. Qin, "Performance evaluation of a new scheduling algorithm for distributed systems with security heterogeneity," *Journal of Parallel and Distributed Computing*, vol. 67, no. 10, pp. 1067–1081, 2007.

[39] X. Tang, K. Li, and Z. Zeng, "A novel security-driven scheduling algorithm for precedence-constrained tasks in heterogeneous distributed systems," *IEEE Transactions on Computers*, vol. 60, no. 7, pp. 1017–1029, 2011.

[40] V. Mnih, K. Kavukcuoglu, D. Silver et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[41] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, MIT Press, Cambridge, MA, USA, 1998.

[42] D. Le and C. Tham, "A deep reinforcement learning based offloading scheme in ad-hoc mobile clouds," in *Proceedings of the IEEE INFOCOM 2018—IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 760–765, IEEE, Honolulu, HI, USA, April 2018.

[43] D. Le and C. Tham, "Quality of service aware computation offloading in an ad-hoc mobile cloud," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 9, pp. 8890–8904, 2018.