


Article

Towards Application-Driven Task Offloading in Edge Computing Based on Deep Reinforcement Learning

Ming Sun ^{1,2,*}, Tie Bao ¹, Dan Xie ¹, Hengyi Lv ²  and Guoliang Si ²

¹ College of Computer Science and Technology, Jilin University, Changchun 130012, China; baotie@jlu.edu.cn (T.B.); xiedan@jlu.edu.cn (D.X.)

² Changchun Institute of Optics, Fine Mechanics and Physics, Chinese Academy of Sciences, Changchun 130033, China; lv_hengyi@163.com (H.L.); siguol@163.com (G.S.)

* Correspondence: sunm19@mails.jlu.edu.cn

Abstract: Edge computing is a new paradigm, which provides storage, computing, and network resources between the traditional cloud data center and terminal devices. In this paper, we concentrate on the application-driven task offloading problem in edge computing by considering the strong dependencies of sub-tasks for multiple users. Our objective is to jointly optimize the total delay and energy generated by applications, while guaranteeing the quality of services of users. First, we formulate the problem for the application-driven tasks in edge computing by jointly considering the delays and the energy consumption. Based on that, we propose a novel Application-driven Task Offloading Strategy (ATOS) based on deep reinforcement learning by adding a preliminary sorting mechanism to realize the joint optimization. Specifically, we analyze the characteristics of application-driven tasks and propose a heuristic algorithm by introducing a new factor to determine the processing order of parallelism sub-tasks. Finally, extensive experiments validate the effectiveness and reliability of the proposed algorithm. To be specific, compared with the baseline strategies, the total cost reduction by ATOS can be up to 64.5% on average.

Keywords: task offloading; edge computing; application-driven task; deep reinforcement learning



Citation: Sun, M.; Bao, T.; Xie, D.; Lv, H.; Si, G. Towards Application-Driven Task Offloading in Edge Computing Based on Deep Reinforcement Learning.

Micromachines **2021**, *12*, 1011.
<https://doi.org/10.3390/mi12091011>

Academic Editors: Roberto Cavicchioli and Paolo Burgio

Received: 20 June 2021

Accepted: 16 August 2021

Published: 26 August 2021

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



Copyright: © 2021 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

With the increasing amount and variety of application data, users' demand for high-quality services has been growing. As a new computing model of IoT, edge computing has become a highly virtualized platform to provide computing, storage, and network services between terminal devices and traditional cloud data centers. As an important infrastructure of edge computing network, edge node includes the switch, router, and embedded server. With the continuous development of Internet terminal devices, smartphones and other terminal devices are widely used. So far, the penetration rate of smartphones in the United States has reached 80%. According to the results released by Cisco, the average number of connected devices per capita will reach 3.6 by 2023 [1]. In edge computing, due to the increasing variety and quantity of data producing from IoT devices, the demand of end-users for high-quality mobile services also increases. In addition, due to the increasing number of connected devices on edge nodes, insufficient resource supply will also lead to high costs and serious load imbalance between edge nodes. Therefore, a complete and comprehensive task offloading strategy is particularly critical for the development of the edge computing network and better application performance promotion.

Figure 1 shows the overview of the system architecture that referring to in this paper. We suppose that the system architecture contains three layers which are cloud, edge, and user. In the edge layer, there are several edge nodes with different limited capacities. We suppose that the connections and locations of the edge nodes have been fixed by the third-party service providers or the cloud data centers. For each user, the connection scope is within a certain area, as shown in the dotted circle in Figure 1. In this area, users can offload

tasks to the corresponding edge node or process through a local device. This paper studies the application-driven tasks constructed by several sub-tasks with strong dependencies, and the sub-tasks that belong to one application can process on different devices. In our system architecture, the provisioning process of the application is to decide the locations of sub-tasks. For example, we use A_i to denote the i^{th} application that contains three sub-tasks: a_1 , a_2 , and a_3 . The dependencies of these sub-tasks is $a_1 \rightarrow a_2 \rightarrow a_3$. One extreme solution is offloading strategy by minimizing the transmission cost for A_i , which processes all sub-tasks on local devices with the order $a_1 \rightarrow a_2 \rightarrow a_3$. However, this solution has the highest delay due to the limited capacity of local devices. Another extreme solution is the offloading strategy by minimizing the processing delay for A_i , which processes all sub-tasks on edge nodes. The user transmits the sub-tasks to the edge node for processing through the wireless channel, and the results are returned after all sub-tasks are completed. However, when the sizes of sub-tasks are large, the total transmission cost will be correspondingly large due to the dependencies between the precursor and the successor, which will lead to a decreasing in the quality of service of users. In this paper, we propose an efficient offloading strategy for multiple users that jointly considering the processing delay and transmission cost.

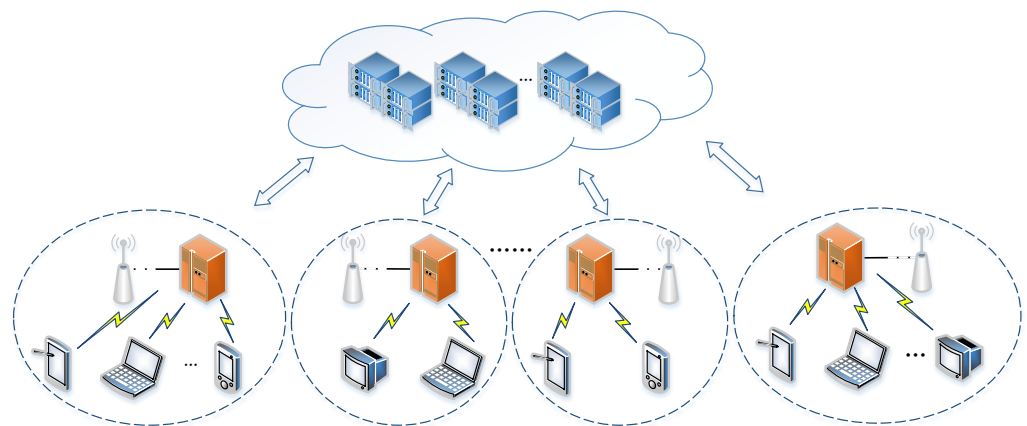


Figure 1. An overview of the system architecture.

In this paper, we concentrate on the application-driven task offloading problem in edge computing by considering the strong dependencies of sub-tasks. The most important point under this problem is to determine the offloading locations of sub-tasks for users, so as to joint optimize the total delay and energy consumption generated by applications while guaranteeing the quality of services for users. Our problem poses several unique challenges as follows: (i) Since the capabilities of edge nodes and local devices are limited and different, it is nontrivial that finding a feasible strategy to complete the sub-task within improving the total cost for users during the offloading process. (ii) In our problem, we consider the applications with strong dependencies which can not achieve complete parallelism. Thus, it is challenging to find a solution that satisfies the dependency relationship with lower cost. The major novel contributions of this paper are as follows:

- We discuss the offloading problem for the application-driven tasks in edge computing, and we optimize the total cost of users which jointly consider the delays and energy consumption.
- We propose an application-driven task offloading strategy (ATOS) based on deep reinforcement learning (DRL) by adding a preliminary sorting mechanism to realize the joint optimization of the delays and energy consumption. Specifically, we analyze the characteristics of application-driven tasks and propose a heuristic algorithm by introducing a new factor to determine the processing order of parallelism sub-tasks. Based on that, we propose a task offloading strategy based on the deep Q-learning network (DQN) by training a fully connected neural network.

- We design a simulator to evaluate our strategy ATOS by comparing it with several state-of-the-art ones. The results are presented from different perspectives to provide conclusions.

The remainder of this paper is organized as follows. Section 2 surveys related works. Section 3 describes the model and then formulates the problem. Section 4 investigates the application-driven task offloading strategy based on DRL. Section 5 presents the experiments. Finally, Section 6 concludes the paper.

2. Related Work

The concept of edge computing was introduced to extend the cloud to the edge of the network, thus enabling a new breed of applications and services. There are numerous novel strategies on the task offloading problem in edge computing that have been proposed. Mao et al. [2,3] introduced an online learning assignment strategy based on dynamic computing offloading that is applicable to a single user. During the offloading process, the execution cost (time delay and offload failure rate) of performing the offloading was measured at each time interval. The online learning allocation strategy only depends on the current system state and did not need to calculate the task feedback results, the distribution information of the wireless channel, and the energy collection. Chen et al. [4] discussed the solution of moving edge computing to meet the low latency requirements in an ultra-dense network. Using the idea of a software-defined network, the task offloading problem was expressed as a mixed-integer nonlinear computing process, and the delay reduction problem was transformed into two sub-problems: task offloading problem and resource allocation problem. Yu et al. [5] considered the application scenarios of the IoT (Internet of Things) and reduced the computing delay by allocating resources reasonably for the program. Then, a complete polynomial-time approximation scheme was proposed, which was more effective in shortening the computing delay than the heuristic algorithm. Spatharakis et al. [6] proposed a two-level Edge Computing architecture to offer computing resources for the remote execution of location based services (LBS). Xu et al. [7] proposed a distributed computing offloading algorithm designed with the method of game theory, and the calculation delay index was quantified to achieve a lower calculation time cost.

In recent years, with the continuous development of machine learning methods, it has gradually infiltrated into various fields, among which reinforcement learning has also found a good application in the offloading decision to reduce the time delay. Meng et al. [8] proposed a delay-sensitive task offloading algorithm based on deep reinforcement learning (DRL) to improve the task processing speed and reduce the task timeout. A new reward function was designed to guide the algorithm to learn offloading decisions from the environment by combining timeout signals and deceleration signals. In addition, intelligent algorithms have also been applied to various fields. Li et al. [9] proposed a joint optimization method of task allocation ratio, channel bandwidth, and computing resources of mobile edge servers based on genetic algorithm, aiming at the situation that part of computing tasks can be partially allocated to the mobile edge server. Under the constraints of wireless transmission resources and mobile edge server processing resources, a genetic algorithm is used to solve the optimization problem of minimizing user task completion time, and the optimal offloading task strategy and resource allocation scheme were obtained. All the above offloading decisions have achieved the purpose of reducing time delay, but they failed to consider the energy consumption at the end of terminal devices during the calculation of the task offloading. The terminal devices may not be able to operate normally due to the lack of power, which has a huge impact on users' experience.

There are also many solutions for the task offloading problems in different environmental scenarios from the perspective of optimizing energy consumption. Zhang et al. [10] adopt the artificial fish swarm algorithm to design the offloading strategy for energy consumption optimization under the constraint of delay. This strategy takes full account of the link conditions in the task data transmission network and effectively reduces the energy consumption of the equipment. However, this strategy has the defect of too high

algorithm complexity. In a multi-resource environment, Xu et al. [11] designed an energy-minimization particle swarm task scheduling algorithm for multi-resource matching to reduce the energy consumption of edge terminal devices. Wei et al. [12] proposed that the task offloading problem can be divided into mobile management and energy-saving optimization, and they use a greedy algorithm to minimize the energy consumption of mobile devices. Lu et al. [13] focus on minimizing the total cost for multiple mobile users to provide an efficient resource provisioning scheme by considering three different cases in edge computing. Yu et al. [14] studied the problem of task offloading in ultra-dense network scenarios. They proposed a task offloading algorithm based on Lyapunov optimization theory, which minimizes the overall energy consumption of the base station and equipment. In order to solve the privacy leakage problem that may occur in the computing offloading decision of mobile edge computing, Zhao et al. [15] proposed a privacy perception computing offloading algorithm. This algorithm has low computational complexity and maintains low terminal energy consumption while ensuring high privacy security. Liu et al. [16] studied the offloading problem based on deep learning, and they proposed a group sparse beamforming framework to optimize network power consumption.

Some studies jointly considered the energy consumption and delay in offloading trade-off optimization problems and put forward some ideas and solutions. Zhang et al. [17] proposed an offloading mechanism assisted by SDN-V, which is suitable for the scenario of the IoV (Internet of Vehicles). The mechanism considered the task diversity, establishes the mathematical model of importance degree, and designed the task offloading sorting algorithm according to the model. Finally, an offloading algorithm based on Q-learning is constructed to optimize the energy consumption and time delay during task offloading. In the case of mobile edge computing, there are many reinforcement learning methods to solve optimization. Zhang et al. [18] proposed a policy-based DRL scheme to solve the problem that a single mobile device offloads tasks to a single mobile edge server. However, there is a question of how much to tweak the network each time that the policy is updated. Too large an amplitude may lead to the problem of non-convergence, while too small an amplitude may lead to the problem of slow convergence. Song et al. [19] proposed a semi-online computational offloading model soCoM based on dueling deep-Q network to explore the user behaviors in sophisticated action space by reinforcement learning for catching unknown environment information. Liu et al. [20] proposed an improved scheme. In this scheme, an artificial neural network was firstly used to learn strategies and make decisions, and another artificial neural network was used to score this decision [21]. In order to improve this problem, Zhan et al. [22] proposed a scheme of disengagement strategy. Firstly, two artificial neural networks were used to approximate the behavior strategy and the target strategy respectively. Then, learning data was generated from behavioral strategies to train the neural network of target strategies. Finally, the parameters of the trained target policy were assigned to the behavior policy. After repeated iterative learning of the target strategy [23], it introduced more artificial neural networks and more parameters. In this paper, we are committed to designing an offloading strategy based on DRL for application-driven tasks that jointly optimize the total delay and energy consumption.

3. Model and Problem Formulation

In this section, we first describe our system model which includes the application model, execution model, and transmission model. Then, we present our problem formulation.

3.1. System Model

The system model is abstracted by the architecture in Figure 1, which is constructed by three layers. The cloud layer is located at the top that is a core in the whole system model which is far from the users. In order to avoid the long-distance transmission and relieve the pressure of the cloud, this paper considers the offloading decision of applications between edge and user layers. In the edge layer, users connect with edge nodes through base stations and wireless channels. In our model, the edge layer is composed of several

small areas according to the locations of edge nodes, and each of them is independent. The edge nodes are heterogeneous, in that they own different capacities. Let $\mathbf{V} = \{V_i\}$ denote the set of edge nodes, where V_i represents the i^{th} one. We use C_i to represent the computing capacity of edge node V_i . In the user layer, users are connecting with the edge node located in their area. Here, the users are local devices, such as mobile phones, laptops, smart bracelets, and so on. Let $\mathbf{U}_i = \{u_i^k\}$ represent the set of users that connecting with edge node V_i . We use u_i^k to denote the k^{th} user in set \mathbf{U}_i . We use c_i^k to denote the computing capacity of u_i^k . The main notations that are commonly used throughout the paper are listed in Table 1.

Table 1. Notations.

Symbols	Definitions
\mathbf{V}	Set of edge nodes, where $\mathbf{V} = \{V_i\}$.
V_i	The i^{th} edge node set \mathbf{V} .
C_i	The computing capacity of V_i .
\mathbf{U}_i	Set of users that connecting with edge node V_i , where $\mathbf{U}_i = \{u_i^k\}$.
u_i^k	The k^{th} user in set \mathbf{U}_i .
c_i^k	The computing capacity of u_i^k .
\mathbf{A}_i^k	The application generated by u_i^k , where $\mathbf{A}_i^k = \{A_i^k, E_i^k\}$.
A_i^k	The set of sub-tasks in \mathbf{A}_i^k , where $A_i^k = \{a_i^k(1), a_i^k(2), \dots, a_i^k(l), \dots, a_i^k(n)\}$.
$r_{a_i^k(l), V_j}$	The rate of u_i^k that transmits $a_i^k(l)$ to V_j .
D_i^k	The total delay of user u_i^k .
E_i^k	The total energy consumption of user u_i^k .

3.1.1. Application Model

In this paper, we assume that the applications are generated by the set of users which composed of several fine-grained sub-tasks. We use a Directed Acyclic Graph (DAG) to represent the application. Let $\mathbf{A}_i^k = \{A_i^k, E_i^k\}$ denote the application generated by u_i^k , where $A_i^k = \{a_i^k(1), a_i^k(2), \dots, a_i^k(l), \dots, a_i^k(n)\}$ is the set of sub-tasks. $a_i^k(l)$ denotes the l^{th} sub-task. We use a vector to describe the demand of $a_i^k(l)$, where $a_i^k(l) = \langle w_i^k(l), \delta_i^k(l), t_i^k(l) \rangle$. Here, $w_i^k(l)$ refers to the workload of $a_i^k(l)$, which indicates the CPU clock cycles required to execute sub-task $a_i^k(l)$. $\delta_i^k(l)$ indicates the ratio of the output data size to the sub-task $a_i^k(l)$. $t_i^k(l)$ refers to the maximum tolerant delay. We use a boolean variable $\zeta_i^k(l)$ to record the offloading decision, where $\zeta_i^k(l) = \{0, 1\}$. When the sub-task $a_i^k(l)$ execute locally, $\zeta_i^k(l) = 0$; otherwise, $\zeta_i^k(l) = 1$.

3.1.2. Transmission Model

The transmission model is defined for the condition that sub-tasks offloading on the edge nodes. According to the Rayleigh fading channel model in Reference [24], the rate of u_i^k that transmits $a_i^k(l)$ to the edge node V_j is defined as

$$r_{a_i^k(l), V_j} = B_{i,j} \cdot \log_2 \left(1 + \frac{p_{i,j} h_{i,j}}{d_{i,j}^{\omega_i} N_i} \right), \quad (1)$$

where $B_{i,j}$ represents the transmission bandwidth between u_i^k and V_j , and $p_{i,j}$ represents the transmission power from u_i^k to V_j . $h_{i,j}$ and $d_{i,j}^{\omega_i}$ represent the channel gain and distance between u_i^k and V_j , respectively. ω_i denotes the path loss exponent, and N_i denotes the Gaussian noise.

3.1.3. Execution on Local Devices

We consider the offloading problem for the fine-grained sub-tasks that decide to perform either locally or edge nodes. We first discuss the total delay when the sub-tasks execute on local devices. For each sub-task, the total delay consists of two components, which are the computing delay and the waiting delay. In the application model, we use $w_i^k(l)$ to denote the workload of $a_i^k(l)$, which indicates the CPU clock cycles required to execute. The computing delay $D_{local}^e(a_i^k(l))$ is defined as:

$$D_{local}^e(a_i^k(l)) = \frac{w_i^k(l)}{c_i^k}. \quad (2)$$

There are two scenarios of waiting delays in local: one is the waiting delay for the execution of the k predecessor sub-tasks and returning the results, and the other one is the delay of waiting for the local execution of the sub-tasks. We use $D_{local}^p(a_i^k(l))$ to denote the waiting delay for the execution of the k predecessor sub-tasks, and $D_{local}^r(a_i^k(l))$ to denote the delay of returning the results.

$$D_{local}^r(a_i^k(l)) = \frac{pre(\frac{w_i^k(l)}{\eta_i^k(l)}) \cdot \delta_i^k(l)}{r_{a_i^k(l), V_j}}. \quad (3)$$

$pre(\frac{w_i^k(l)}{\eta_i^k(l)})$ represents the data size of precursor sub-tasks of $a_i^k(l)$, where $\eta_i^k(l)$ denotes the CPU cycles required for each MB of the sub-task $a_i^k(l)$. $\delta_i^k(l)$ represents the ratio of the output data size to the sub-task $a_i^k(l)$. $r_{a_i^k(l), V_j}$ is the transmission rate that transmits $a_i^k(l)$ to the edge node V_j . We use $D_{local}^q(a_i^k(l))$ to denote the queuing delay of local execution of k predecessor sub-tasks.

Therefore, the total waiting delay $D_{local}^w(a_i^k(l))$ is defined as

$$D_{local}^w(a_i^k(l)) = \max\{D_{local}^p(a_i^k(l)) + D_{local}^r(a_i^k(l)), D_{local}^q(a_i^k(l))\}. \quad (4)$$

The total delay is defined as

$$D_{local}(a_i^k(l)) = D_{local}^w(a_i^k(l)) + D_{local}^e(a_i^k(l)). \quad (5)$$

The total energy consumption for sub-task $a_i^k(l)$ on u_i^k is defined as

$$E_{local}(v_i^k(l)) = \epsilon_i^k \cdot w_i^k(l) \cdot (c_i^k)^2. \quad (6)$$

Here, ϵ_i^k is the coefficient factor [25] of chip architecture on u_i^k .

3.1.4. Execution on Edge Nodes

Comparing with the sub-tasks executing locally, the total delay under the edge nodes includes the transmission delay. We use $D_{edge}^{t_{i,j}}(a_i^k(l))$ to represent the transmission delay from u_i^k to V_j .

$$D_{edge}^{t_{i,j}}(a_i^k(l)) = \frac{w_i^k(l)}{\eta_i^k(l) \cdot r_{a_i^k(l), V_j}}. \quad (7)$$

We use $\eta_i^k(l)$ to denote the CPU cycles required for each MB of the sub-task $a_i^k(l)$. Since $w_i^k(l)$ is the workload of sub-task $a_i^k(l)$, the data size is $\frac{w_i^k(l)}{\eta_i^k(l)}$.

The computing delay $D_{edge}^e(a_i^k(l))$ is defined as:

$$D_{edge}^e(a_i^k(l)) = \frac{w_i^k(l)}{C_j}. \tag{8}$$

C_j is the computing capacity of edge node V_j .

In this case, the waiting delay for the sub-task $a_i^k(l)$ involves the preparation time for the precursor sub-tasks of $pre(a_i^k(l))$ and the return time of the result. In addition, we suppose that the capacities of the edge nodes are also limited, and one edge node can only execute one sub-task at the same time. The abstract model is shown in Figure 2. The waiting delay is defined as

$$D_{edge}^w(a_i^k(l)) = \max\{D_{edge}^p(a_i^k(l)) + D_{edge}^r(a_i^k(l)), D_{edge}^q(a_i^k(l)), D_{edge}^{t_{ij}}(a_i^k(l))\}. \tag{9}$$

$D_{edge}^p(a_i^k(l))$ and $D_{edge}^r(a_i^k(l))$ are the preparation time for the precursor sub-tasks of $pre(a_i^k(l))$ and the return time of the result, respectively. The total delay is defined as

$$D_{edge}(a_i^k(l)) = D_{edge}^w(a_i^k(l)) + D_{edge}^e(a_i^k(l)). \tag{10}$$

The total energy consumption for sub-task $a_i^k(l)$ on edge node V_j is defined as

$$E_{edge}(a_i^k(l)) = p_i \cdot D_{edge}^{t_{ij}}(a_i^k(l)) + p_j \cdot D_r(a_i^k(l)). \tag{11}$$

Here, p_i and p_j are the transmission power of local device u_i^k and edge node V_j , respectively.

Since the types of sub-tasks vary according to the application scenarios. Some of them are sensitive to the delay, while others are more sensitive to the energy consumption. Therefore, we jointly consider the total delay and energy consumption. Let D_i^k denote the total delay of user u_i^k , where

$$D_i^k = \zeta_i^k(l) \cdot D_{edge}(a_i^k(l)) + (1 - \zeta_i^k(l))D_{local}(a_i^k(l)). \tag{12}$$

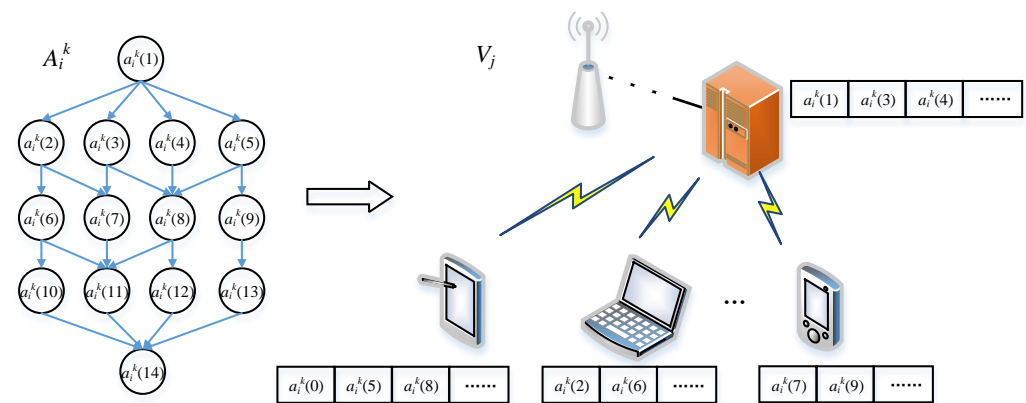


Figure 2. An abstraction of the system model.

Let E denote the total energy consumption, where

$$E_i^k = \zeta_i^k(l) \cdot E_{edge}(a_i^k(l)) + (1 - \zeta_i^k(l))E_{local}(a_i^k(l)). \tag{13}$$

Therefore, the total cost of user u_i^k is defined as

$$T_i^k = \rho \cdot D_i^k + (1 - \rho)E_i^k. \tag{14}$$

In this paper, we consider the delay and energy consumption of applications generated by users. Our objective is to minimize the total cost, and the formulation is shown as follows:

$$\text{minimize} \quad \sum_{i=1}^{|\mathbf{V}|} \sum_{k=1}^{|\mathbf{U}_i|} T_i^k, \tag{15}$$

$$\text{subject to} \quad D_i^k \leq \tau_i^k, \forall u_i^k \in \mathbf{U}_i, \tag{16}$$

$$(a_i^k(l-1), a_i^k(l)) \in A_i^k, \tag{17}$$

$$\zeta_i^k(l) = \{0, 1\}. \tag{18}$$

Equation (15) represents the objective function, and Equations (16)–(18) are the constraints. Equation (16) represents the total delay of an application requires that should not exceed the maximum required delay. Equation (17) represents the dependency of the sub-tasks in the application, and Equation (18) represents the constraints on the locations of the offloading, where 1 denotes that representing to offload on the edge node, otherwise to the local devices.

4. An Application-Driven Task Offloading Strategy Based on DRL

In this section, we propose an Application-driven Task Offloading Strategy (ATOS) based on DRL. The main idea of ATOS is to add a preliminary sorting mechanism and realize the jointly optimization of the delay and energy consumption by proposing a task offloading strategy based on the deep Q-learning. The detailed description of ATOS is shown as follows.

4.1. Preliminary Sorting Mechanism (PSM)

In this paper, we assume that the applications are generated by the users which are composed of several fine-grained sub-tasks. Although these sub-tasks have strong dependencies, there are still existing some parallel sub-tasks whose execution order will affect the result of subsequent task offloading. An illustration of PSM for the application A_i^k is shown in Figure 3. In this subsection, we introduce a Preliminary Sorting Mechanism (PSM) to determine the sequences of sub-tasks. We first initialize the preliminary sorting set $\omega_i^k =: \Phi$ in line 1. For each sub-task a_i^k in application A_i^k , we check the in-degree $I(a_i^k)$. If the value of in-degree is 0, we add this sub-task into queue S_i^k . Otherwise, we check the out-degree $O(a_i^k)$. If the value of out-degree is 0, it represents that this is the last sub-task in the application. Then, we return the sequence queue S_i^k . If neither of the above cases is true, we add the subsequent and sibling sub-tasks of $sub(a_i^k)$ to the preliminary sorting set ω_i^k . According to the structure and characteristics of the application, we define a priority factor f_i^k .

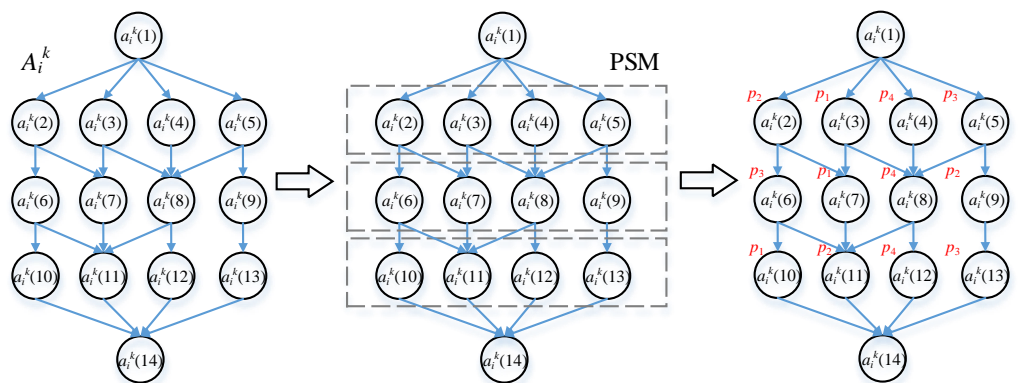


Figure 3. An illustration of PSM for the application.

Definition 1 (Priority Factors). The priority factors f_i^k is to decide the execution order for the parallel sub-tasks in application A_i^k , where $f_i^k(l) = \frac{I(a_i^k(l))}{O(a_i^k(l)) \cdot w_i^k(l)}$.

Based on that, we calculate the priority factors f_i^k for subsequent sub-tasks in set ω_i^k . In line 9, we update ω_i^k with descending order of f_i^k , where $\omega_i^k := \text{descending}(\text{sub}(a_i^k))$. Then, we update S_i^k by adding the preliminary sorting set ω_i^k into queue. In line 11, we return sequence queue S_i^k .

4.2. Task Offloading Based on Deep Q-Learning

In this subsection, we introduce our task offloading strategy based on DQN. To describe the environment of the DCN correctly and concisely for the agent, the state space should include the knowledge of applications and the status of the total cost. So, the state is designed as follows.

Definition 2 (State). The state s_t is a vector consisting of $s_t = [T_i^k, \mathbf{U}_i / |\hat{\mathbf{U}}_i|]_t$, where $\mathbf{U}_i / |\hat{\mathbf{U}}_i|$ are the sub-tasks waiting to be scheduled, and $T_i^k = \sum_{k=1}^{|\hat{\mathbf{U}}_i|} T_i^k$ is the total cost of the scheduled sub-tasks $\hat{\mathbf{U}}_i$.

We consider realizing the offloading by training the agent which needs to choose a destination (edge nodes or local devices) for the sub-tasks of each application. The action A_t is designed as follows.

Definition 3 (Action). The action space $a_t = [\zeta_i^k(l), 1 - \zeta_i^k(l)]_t$ is the adjusting action, where $\zeta_i^k(l) = 0$ or $\zeta_i^k(l) = 1$ means that the target location of adjustment is on edge node or local device.

At each time slot t , the agent will receive a reward $R(s_t, a_t)$ in a certain state s_t after executing action a_t . Since the objective is to minimize the total cost of delay and energy consumption which contract with the goal of RL that maximizing the long-term reward, the reward function should be negatively related to the weighted sum of delay and energy consumption. The reward function $R(s_t, a_t)$ is designed as follows.

Definition 4 (Reward). The immediate reward is $R(s_t, a_t) = \frac{T_{base} - T_i^k}{T_{base}}$, where T_i^k is the total cost of the scheduled sub-tasks, and T_{base} is a baseline cost that offloading with greedy strategy.

Algorithm 1 summarizes the ATOS, and the main idea is to use a deep reinforcement learning agent to perform the dynamic offloading of sub-tasks in applications to minimize the total cost of delay and energy consumption. We first initialize some preliminary parameters which include setting the replay memory \mathcal{D} to capacity N . Meanwhile, we initialize the action-value function Q with random weight θ and the target action-value function \hat{Q} with weights $\theta^- = \theta$. In lines 2 to 15, we start to train the agent by running a number of κ episodes with our environment. During each episode, Initialize sequence S_i^k based on Algorithm 2 in line 3. The training process starts from lines 4 to 14. In line 4, the agent selects a random action a_t with probability ε ; otherwise, it will select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$ with the maximum Q value in line 5. In line 6, we set $s_{t+1} = s_t$, a_t , x_{t+1} , and preprocess $\phi_{t+1} = \phi(S_{t+1})$, and we store the transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in the replay memory \mathcal{D} . After that, we sample a random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from \mathcal{D} in lines 7 to 8. The objective of our problem is to minimize the total cost of the users which is contrary to the cumulative reward received by the agent. In line 12 to 14, the agent performs a gradient descent step on $(y - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters θ , and resets $\hat{Q} = Q$ every C steps. The offloading results are returned in line 15.

Algorithm 1 Application-driven Task Offloading Strategy based on DQN (ATOS).**Input:** The applications \mathbf{A}_i^k generated by user \mathbf{U}_i with sequences S_i^k ;**Output:** Offloading strategy \mathbf{X}_i^k ;

- 1: Initialize \mathcal{D} to N, Q with random weights θ , and \hat{Q} with weights $\theta^- := \theta$;
- 2: **for** episode from 1 to κ **do**
- 3: Initialize sequence S_i^k based on Algorithm 2;
- 4: With probability ε select a random action a_t ;
- 5: Otherwise select $a_t = \operatorname{argmax}_a Q(\phi(S_t), a; \theta)$;
- 6: Set $S_{t+1} = S_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(S_{t+1})$.
- 7: Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in \mathcal{D} ;
- 8: Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from \mathcal{D} .
- 9: **if** episode terminates at step $j + 1$ **then**
- 10: Set $y_j = r_j$;
- 11: **else**
- 12: Set $y_j = r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-)$;
- 13: Perform a gradient descent step on $(y - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters θ .
- 14: Every C steps reset $\hat{Q} = Q$;
- 15: **return** Offloading strategy \mathbf{X}_i^k ;

Algorithm 2 Preliminary Sorting Mechanism (PSM).**Input:** The application \mathbf{A}_i^k generated by user \mathbf{U}_i ;**Output:** The sequence queue S_i^k of the sub-tasks in \mathbf{A}_i^k ;

- 1: Initialize the preliminary sorting set $\omega_i^k := \Phi$;
- 2: **for** each sub-task a_i^k in \mathbf{A}_i^k **do**
- 3: **if** $\mathbf{I}(a_i^k) = 0$ **then**
- 4: Adding sub-task a_i^k into queue S_i^k ;
- 5: **else if** $\mathbf{O}(a_i^k) = 0$ **then**
- 6: Go to line 11;
- 7: Adding subsequent and sibling sub-tasks of $\operatorname{sub}(a_i^k)$ to preliminary sorting set ω_i^k ;
- 8: Calculate the priority factors f_i^k for subsequent sub-tasks in set ω_i^k ;
- 9: Update ω_i^k with descending order of f_i^k , where $\omega_i^k := \operatorname{descending}(\operatorname{sub}(a_i^k))$;
- 10: Update S_i^k by adding set ω_i^k into queue;
- 11: **return** Sequence queue S_i^k ;

5. Experiment Evaluation

In this section, we will conduct experiments on the designing simulator to evaluate our strategy ATOS. We analyzed and shown the experimental results from different perspectives to provide insightful conclusions.

5.1. Basic Setting of the Synthetic Dataset

In this subsection, we develop a simulator using python and evaluate the performance of our algorithms by building a synthetic dataset. In our simulator, the number of edge nodes ranges from 5 to 10. For each edge node, we consider an area with 500 square meters, and there are existing 1 to 5 users. The setting of parameters in our paper are listed in Table 2, which refer to References [24,26]. Each user deploys one application, and each application consists of 12 to 21 sub-tasks. In our experiments, we test several groups of hyperparameters that the learning rates range from 0.0005 to 0.001, and the e-greedy factors range from 0.7 to 0.95. In addition, we test the reward decay between 0.6 and 0.9 at 0.05 intervals, and we test the replacing target iterations between 20 and 500 at 10 intervals. According to the test results, we choose the group of hyperparameters listed in Table 3 as the experimental setting. We consider the four baseline algorithms to be the comparisons

as follows: (i) Offloading all sub-tasks on the edge nodes (Offloading_edge): for each application, we offload the sub-task on the edge nodes iteratively. (ii) Offloading all sub-tasks on the local devices (Offloading_local): for each application, we offload the sub-task on the local devices iteratively. (iii) Offloading all sub-tasks on the edge nodes or the local devices randomly (Offloading_random): for each application, we offload the sub-task to the edge node or the local device randomly in each iteration. (iv) Offloading all sub-tasks on the edge nodes or the local devices through greedy strategy (Offloading_greedy): for each application, we greedy choose the offloading destination by considering the queueing time and the capacities in each iteration. We compare ATOS with these four baseline algorithms, and the effectiveness of ATOS is verified.

Table 2. Setting of parameters.

Parameters	Values
Transmission bandwidth $B_{i,j}$	180 kHz
Path loss exponent ω_i	3
Gaussian noise N_i	10^{-13}
Data size of sub-tasks	0.3 Mb~1 Mb.
Transmission power of local device $p_{i,j}$	3 W
Computing capacity of local devices	0.5–1 GHz
Computing capacity of edge nodes	5 GHz
The coefficient of channel fading	10^{-6}
The coefficient factor of chip architecture	10^{-20}

Table 3. Hyperparameter settings.

Hyperparameter	Settings
learning rate α	0.0005
e-greedy ϵ	0.9
reward decay γ	0.7
replacing target iterations	30
replay memory \mathcal{D}	500

5.2. Evaluations on the Performance

In this subsection, we discuss the total cost of multiple users with the application-driven task offloading requests in edge computing, the results are shown in Figure 4. Four baseline algorithms (Offloading_edge, Offloading_local, Offloading_random, Offloading_greedy) are used to compare with our algorithm. We choose 6 groups of topologies that the edge nodes in the edge layers are ranging from 5 to 10. In order to facilitate the analysis of the results, each group ran 10 times and calculated the average value. According to the results, we obtain the following observations: (i) For each group, the total cost is the largest when all tasks are executed locally (Offloading_local) or the edge nodes (Offloading_edge). As shown in Figure 4a–f, the total cost of each group for the users in both cases will reach the highest value of the ordinate. Here, in order to show the difference between the results of these two strategies and those of other ones, we set the highest limitation of the ordinate. The total costs under these six groups are listed in Table 4. We can see that, since the limited capacities of edge nodes and local devices, the total costs of these two strategies are much higher than that of other ones. In addition, the total cost of Offloading_edge is lower than Offloading_local. The reason is that, although they will produce transmission energy consumption for the sub-tasks that are offloading to the edge nodes, the high computation delay caused by limited computing capacities of local devices is the key factor of the high total cost for the users. (ii) The impact of algorithms on the total costs is related to the number of edge nodes. We compared the last three columns of the six experiments in Figure 4a–f, the trend of the total costs decreases. For the topology with a small number of edge nodes (5 and 6 edge nodes in Figure 4a,b), the gap in the total cost between Offloading_random and Offloading_greedy is not large. However,

with the scaling of the topology, the total cost of Offloading_greedy is significantly lower than Offloading_random. We can see that ATOS can better reduce the total cost in the six groups. Compared with Offloading_random and Offloading_greedy, the optimization rates of ATOS are improving 78.3% and 50.6% on average, respectively.

Table 4. Total costs of Offloading_edge and Offloading_local.

# of Edge Nodes	Offloading_Edge	Offloading_Local
5	1,094,706	5,472,686
6	1,095,779	5,478,050
7	1,451,771.14	7,257,740.15
8	1,766,995.48	8,833,583.48
9	2,322,826.54	11,612,323.16
10	2,638,105.86	13,188,441.59

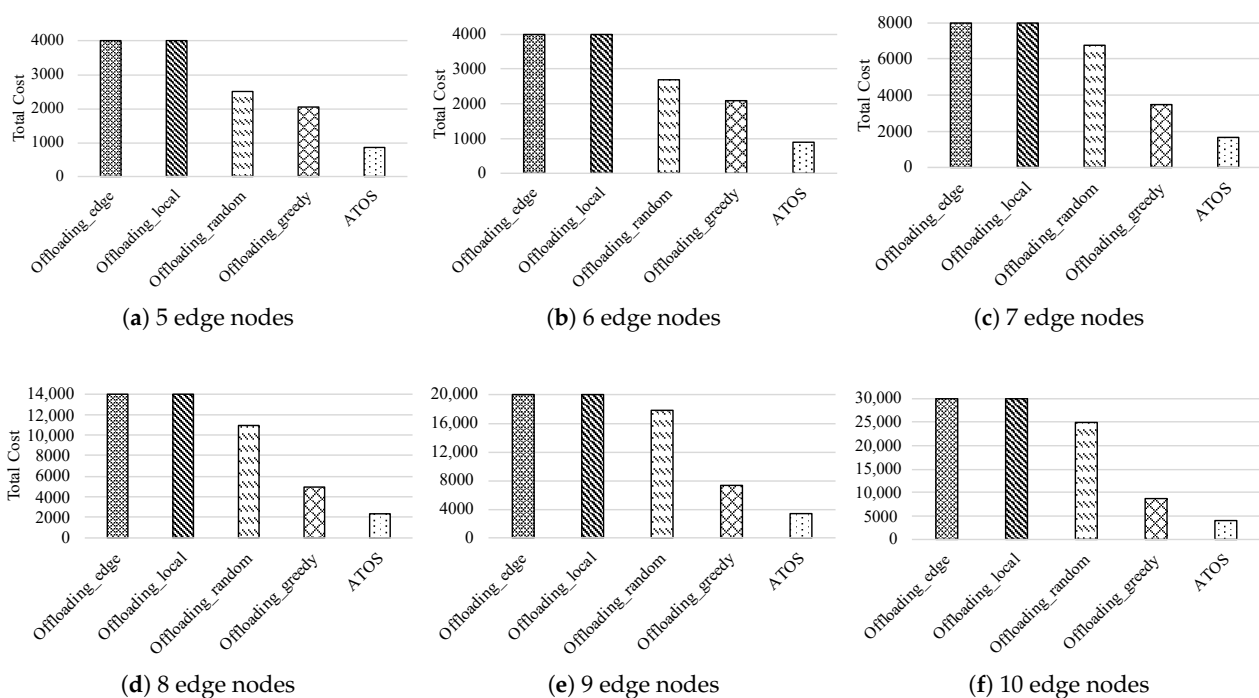


Figure 4. The total cost under different offloading strategies with edge nodes ranging from 5 to 10.

5.3. Evaluations on the Convergence

In this subsection, we analyze the convergence of ATOS. We choose 6 groups of topologies with different edge nodes (5, 6, 7, and 8), and the number of sub-tasks in each application ranges from 12 to 21. In order to facilitate the analysis of the results, the number of iterations of each group is 500. According to the results, we obtain the following observations: (i) The total cost under each group will close to convergence after 500 iterations. As shown in Figure 5, the total cost within 0 to 100 iterations is decreasing sharply. In groups 5, 6, 7, 9, and 10, the total cost fluctuates strongly at about 30 iterations. In group 8, although the fluctuation is not violent, the abnormal values appears frequently, at about 100 to 380 iterations. For different topologies, the ranges of convergence values are different due to the various sizes of sub-tasks in the applications. (ii) The learning ranges of ATOS increase with the expansion of different topologies. With the increasing number of edge nodes, more actions can be selected in the process of learning and training, so the range of total cost becomes larger. When the number of nodes in the edge layer reaches 10, the total cost will be close to 5×10^6 . Due to the differences in the applications deployed by users, some groups will fluctuate during the convergence process. For example, in

Figure 5c, the value is fluctuating from 100 to 200 iterations. In summary, we can see that ATOS basically reaches convergence and maintains stability quickly.

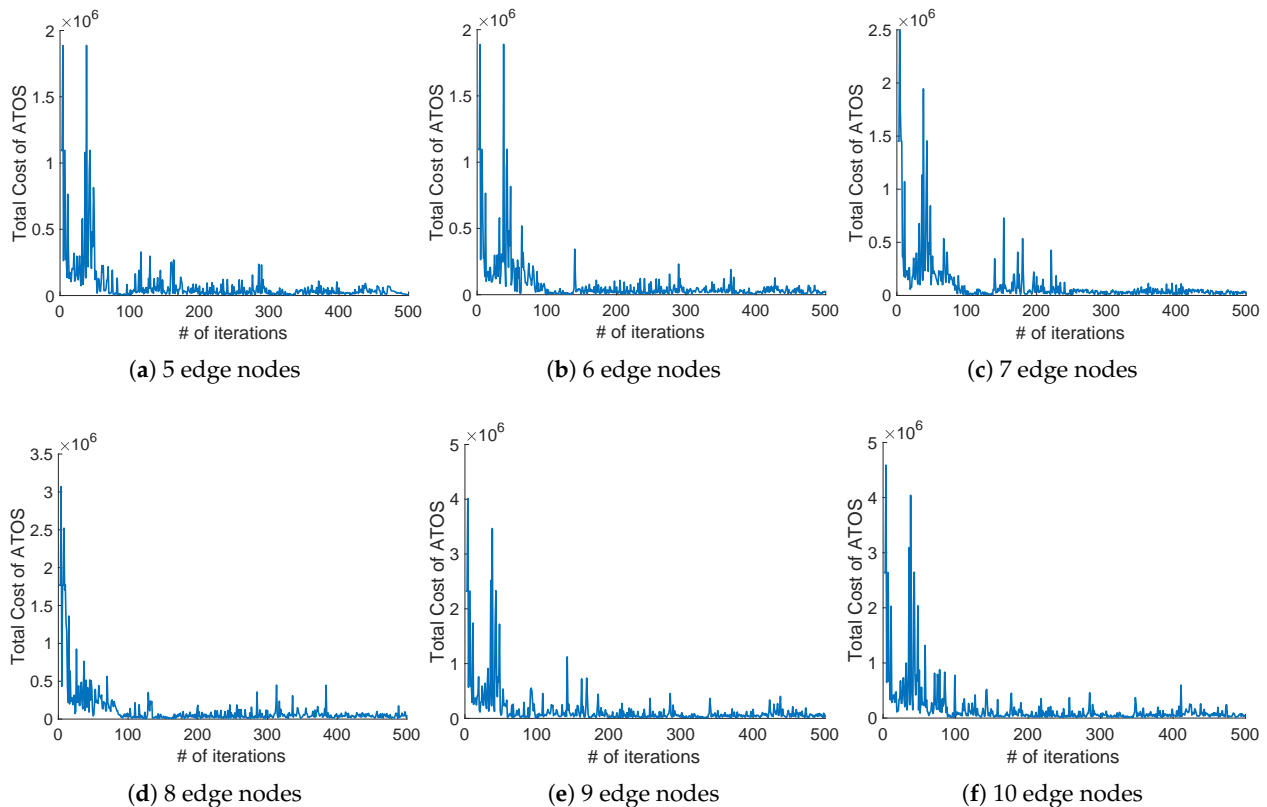


Figure 5. The convergence of total cost under ATOS with the number of edge nodes ranging from 5 to 10.

6. Conclusions

In this paper, we study the application-driven task offloading in edge computing by considering the strong dependencies of sub-tasks. We first formulate the task offloading as a joint optimization problem that considers the total delay and energy consumption. Based on that, we propose a novel task offloading strategy ATOS based on DRL by adding a preliminary sorting mechanism. We analyze the characteristics of application-driven tasks and propose a heuristic algorithm PSM to determine the processing order of the parallelism sub-tasks. Finally, we study the convergence and performance of ATOS through extensive experiments. The results show that ATOS can obtain a reasonable offloading strategy and reduce the total cost of users.

In future work, we will consider the mobility of users under the cooperation of edge nodes. In addition, we will further consider the application-driven tasks with strong dependencies that combine with actual scenarios.

Author Contributions: Conceptualization, M.S. and T.B.; methodology, M.S.; software, M.S.; validation, M.S., T.B., and D.X.; formal analysis, M.S.; investigation, M.S.; resources, H.L.; data curation, H.L. and G.S.; writing—original draft preparation, M.S.; writing—review and editing, D.X. and H.L.; visualization, M.S.; supervision, T.B. and G.S.; project administration, T.B.; funding acquisition, T.B. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Acknowledgments: The authors would like to thank the reviewers for their efforts and for providing helpful suggestions that have led to several important improvements in our work. We would also like to thank all teachers and students in our laboratory for helpful discussions.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Cisco, U. Cisco Annual Internet Report (2018–2023) White Paper. Available online: <https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/whitepaper-c11-741490.html> (accessed on 26 March 2021).
2. Mao, Y.; Zhang, J.; Letaief, K.B. Dynamic Computation Offloading for Mobile-Edge Computing With Energy Harvesting Devices. *IEEE J. Sel. Areas Commun.* **2016**, *34*, 3590–3605. [[CrossRef](#)]
3. Ulukus, S.; Yener, A.; Erkip, E.; Simeone, O.; Zorzi, M.; Grover, P.; Huang, K. Energy Harvesting Wireless Communications: A Review of Recent Advances. *IEEE J. Sel. Areas Commun.* **2015**, *33*, 360–381. [[CrossRef](#)]
4. Chen, M.; Hao, Y. Task offloading for mobile edge computing in software defined ultra-dense network. *IEEE J. Sel. Areas Commun.* **2018**, *36*, 587–597. [[CrossRef](#)]
5. Yu, R.; Xue, G.; Zhang, X. Application Provisioning in Fog Computing-enabled Internet-of-Things: A Network Perspective. In Proceedings of the IEEE INFOCOM 2018-IEEE Conference on Computer Communications, Honolulu, HI, USA, 16–19 April 2018; pp. 783–791
6. Spatharakis, D.; Dimolitsas, I.; Dechouniotis, D.; Papatthanail, G.; Fotoglou, I.; Papadimitriou, P.; Papavassiliou, S. A scalable edge computing architecture enabling smart offloading for location based services. *Pervasive Mob. Comput.* **2020**, *67*, 101–217. [[CrossRef](#)]
7. Xu, C.; Lei, J.; Li, W.; Fu, X. Efficient multi-user computation offloading for mobile-edge cloud computing. *IEEE/ACM Trans. Netw.* **2016**, *24*, 2795–2808.
8. Meng, H.; Chao, D.; Guo, Q.; Li, X. Delay-sensitive task scheduling with deep reinforcement learning in mobile-edge computing systems. *J. Physics Conf. Ser.* **2019**, *1229*, 22–30. [[CrossRef](#)]
9. Li, Z.; Zhu, Q. Algorithm-Based Optimization of Offloading and Resource Allocation in Mobile-Edge Computing. *Information* **2020**, *11*, 83. [[CrossRef](#)]
10. Zhang, H.; Guo, J.; Yang, L.; Li, X.; Ji, H. Computation offloading considering fronthaul and backhaul in small-cell networks integrated with MEC. In Proceedings of the IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), Atlanta, GA, USA, 1–4 May 2017; pp. 115–120.
11. Xu, J.; Li, X.; Ding, R.; Liu, X. Energy efficient multi-resource computation offloading strategy in mobile edge computing. *Comput. Integr. Manuf. Syst.* **2019**, *25*, 954–961.
12. Wei, F.; Chen, S.; Zou, W. A greedy algorithm for task offloading in mobile edge computing system. *China Commun.* **2018**, *15*, 149–157. [[CrossRef](#)]
13. Lu, S.; Wu, J.; Duan, Y.; Wang, N.; Fang, J. Towards cost-efficient resource provisioning with multiple mobile users in fog computing. *J. Parallel Distrib. Comput.* **2020**, *146*, 96–106. [[CrossRef](#)]
14. Yu, B.; Pu, L.; Xie, Y.; Jian, Z. Joint task offloading and base station association in mobile edge computing. *J. Comput. Res. Dev.* **2018**, *55*, 537–550.
15. Zhao, X.; Peng, J.; You, W. A privacy-aware computation offloading method based on lyapunov optimization. *J. Electron. Inf. Technol.* **2020**, *42*, 704–711.
16. Liu, L.; Liu, X.; Zeng, S.; Wang, T.; Pang, R. Research on virtual machines migration strategy based on mobile user mobility in mobile edge computing. *J. Chongqing Univ. Posts Telecommun. Ence Ed.* **2019**, *31*, 158–165.
17. Zhang, H.; Jing, K.; Liu, K. An offloading mechanism based on software defined network and mobile edge computing in vehicular networks. *J. Electron. Inf. Technol.* **2020**, *42*, 645–652.
18. Zhang, H.; Wu, W.; Wang, C.; Li, M.; Yang, R. Deep Reinforcement Learning-Based Offloading Decision Optimization in Mobile Edge Computing. In Proceedings of the 2019 IEEE Wireless Communications and Networking Conference (WCNC), Marrakech, Morocco, 15–18 April 2019; pp. 1–7
19. Song, S.; Fang, Z.; Zhang, Z.; Chen, C.L.; Sun, H. Semi-Online Computational Offloading by Dueling Deep-Q Network for User Behavior Prediction. *IEEE Access* **2020**, *8*, 118192–118204. [[CrossRef](#)]
20. Liu, Y.; Cui, Q.; Zhang, J.; Chen, Y.; Hou, Y. An Actor-Critic Deep Reinforcement Learning Based Computation Offloading for Three-Tier Mobile Computing Networks. In Proceedings of the 2019 11th International Conference on Wireless Communications and Signal Processing (WCSP), Xi'an, China, 23–25 October 2019; pp. 1–6
21. Mnih, V.; Badia, A.P.; Mirza, M.; Graves, A.; Lillicrap, T.; Harley, T.; Silver, D.; Kavukcuoglu, K. Asynchronous methods for deep reinforcement learning. In Proceedings of the 33rd International Conference on International Conference on Machine Learning, New York, NY, USA, 20–22 June 2016; pp. 1928–1937
22. Zhang, W.; Luo, C.; Wang, J.; Wang, C.; Zhu, Q. Deep Reinforcement Learning-Based Offloading Scheduling for Vehicular Edge Computing. *IEEE Internet Things J.* **2020**, *7*, 5449–5465. [[CrossRef](#)]
23. Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; Klimov, O. Proximal policy optimization algorithms. *arXiv* **2017**, arXiv:1707.06347.

24. Li, Q.; Zhao, J.; Gong, Y. Computation offloading and resource management scheme in mobile edge computing. *Telecommun. Sci.* **2019**, *35*, 1–11.
25. Zhang, W.; Wen, Y.; Guan, K.; Kilper, D.; Luo, H.; Wu, D.O. Energy-Optimal Mobile Cloud Computing under Stochastic Wireless Channel. *IEEE Trans. Wirel. Commun.* **2013**, *12*, 4569–4581. [[CrossRef](#)]
26. Guo, F.; Zhang, H.; Hong, J.; Xi, L.; Leung, V. An Efficient Computation Offloading Management Scheme in the Densely Deployed Small Cell Networks With Mobile Edge Computing. *IEEE/ACM Trans. Netw.* **2018**, *26*, 2651–2664. [[CrossRef](#)]