Hindawi
Wireless Communications and Mobile Computing
Volume 2021, Article ID 6457099, 13 pages
https://doi.org/10.1155/2021/6457099

Research Article

Deep Reinforcement Learning for Collaborative Computation Offloading on Internet of Vehicles

Yureng Li,[1] **Shouzhi Xu**,[1] **and Dawei Li**[2]

[1]*College of Computer and Information Technology, China Three Gorges University, Yichang 443002, China*
[2]*Department of Computer Science, Montclair State University, Montclair, NJ, USA*

Correspondence should be addressed to Shouzhi Xu; xsz@ctgu.edu.cn

Received 16 August 2021; Accepted 25 October 2021; Published 30 November 2021

Academic Editor: Pengfei Wang

With the increase of Internet of vehicles (IoVs) traffic, the contradiction between a large number of computing tasks and limited computing resources has become increasingly prominent. Although many existing studies have been proposed to solve this problem, their main consideration is to achieve different optimization goals in the case of edge offloading in static scenarios. Since realistic scenarios are complicated and generally time-varying, these studies in static scenes are imperfect. In this paper, we consider a collaborative computation offloading in a time-varying edge-cloud network, and we formulate an optimization problem with considering both delay constraints and resource constraints, aiming to minimize the long-term system cost. Since the set of feasible solutions to the problem is nonconvex, and the complexity of the problem is very large, we propose a Q-learning-based approach to solve the optimization problem. In addition, due to the dimensional catastrophes, we further propose a DQN-based approach to solve the optimization problem. Finally, by comparing our two proposed algorithms with typical algorithms, the simulation results show that our proposed approaches achieve better performance. Under the same conditions, by comparing our two proposed algorithms with typical algorithms, the simulation results show that our proposed Q-learning-based method reduces the system cost by about 49% and 42% compared to typical algorithms. And in the same case, compared with the classical two schemes, our proposed DQN-based scheme reduces the system cost by 62% and 57%.

1. Introduction

With the rapid development of the Internet of Vehicles (IoVs), vehicles have generated an increasing number of intensive computation tasks, such as online interactive applications, route planning, and traffic flow prediction. However, since these applications require a certain amount of computing resources and have Quality of Service (QoS) requirements, basic equipment in IoVs cannot meet the needs of vehicles suffering from hardware and other limitations [1].

To relieve the pressure of tight resources in IoV, Mobile Cloud Computing (MCC) is often used as a promising solution in such cases. In MCC networks, cloud servers (CS) have large volume of computation and communication resources to provide offloading services to multiple users at the same time [2]. However, since CS is usually deployed far away from vehicles, MCC can lead to significant transmission delays and system costs [3].

Mobile edge computing (MEC) is believed to serve as a reliable paradigm due to its ability to improve the QoS of vehicles in a way that reduces latency and energy costs [4]. To address latency-sensitive tasks while satisfying their demand for resources, MEC sinks computing services to the network edge, while network resources are also fully utilized [5, 6]. In IoVs, MEC servers are deployed on roadside units (RSUs), and vehicles within their coverage area are able to receive offloading services. The computing resources in MEC are still limited, which may result in some of the computation tasks failing when the amount of tasks in the network is too large [7]. Most of the previous studies have considered the optimal allocation of computing resources and model selection separately. Moreover, in terms of computation offloading, some studies offload computing tasks from vehicles to edge servers or CS without considering the optimization of resources or the combination of both. Therefore, it is necessary to put forward a combined solution when solving practical problems.

Moreover, with the development of machine learning, reinforcement learning (RL) is considered as an effective method for finding optimal computation offloading strategies in time-varying scenarios. Compared with other optimization methods, agents in RL can find an optimal policy by observing the current environment to select actions as well as obtaining future rewards by constantly interacting with the environment [8]. Therefore, it is of great interest to design an efficient RL-based computation offloading and resource allocation scheme.

In this paper, we propose an efficient offloading scheme in the edge-cloud network by jointly optimizing offloading decision and resource allocation and ultimately achieving the optimization goal of minimizing system costs. In the concerned scenario, CS makes offloading decisions for each vehicle based on the current system state. Distinct from existing works, we consider both cooperation between CS and RSUs, as well as cooperation between RSUs. We formulate the optimization problem as a mixed-integer nonlinear programming (MINLP) problem, and to solve the optimization problem, we first propose a Q-learning-based approach. However, since Q-learning method may lead to dimensional catastrophe when the state and action space become too large [9], we further propose a DQN-based approach to compensate for this drawback. The main contributions of this article are as follows:

(i) We construct an edge-cloud network for time-varying IoVs, in which CS together with RSUs can process computing tasks for vehicles cooperatively

(ii) We study the cooperative offloading problem in proposed model and formulate the optimization problem as a MINLP problem, aiming to minimize the system cost

(iii) After defining the state space, action space, and reward function, we approximate the optimisation process as a Markov decision process (MDP). Based on the MDP, we propose Q-learning and DQN-based methods to solve the optimisation problem, respectively

(iv) Numerical results demonstrate that our proposed schemes significantly reduce system cost compared with other typical algorithms

The remainder of this paper is organized as follows. Section 2 introduces the related work of this article. Section 3 describes the system model in detail, including the computational model and the communication model. In Section 4, we describe the optimization problem and formulate the optimization problem as an MINLP problem. In Section 5, a Q-based method and a DQN-based method are proposed for the optimization problem, respectively. Simulation results and analysis are given in Section 6. Finally, Section 7 gives a summary of this paper.

2. Related Work

In recent years, research on computational offloading in MEC and MCC scenarios has become increasingly popular due to the need for practical scenarios. Specifically, in [10], Mao et al. perform a joint optimization for power and computational offloading in a MEC scenario using NOMA in order to achieve the optimization goal of minimizing system energy consumption. In [11], Ning et al. consider a MEC heuristic offloading scheme based on partial offloading with the optimization goal of reducing the system delay. In [12], Kuang et al. use a hierarchical approach to obtain suboptimal solutions to optimize the offloading pattern and power allocation in the MEC scenario. In [13], authors offload tasks to nearby vehicles as well as edge devices, and in this way solve the problem of computational offloading and probabilistic caching. In [14], Bi et al. consider the problem of service delivery and deployment in a single-user MEC system with the optimization goal of minimizing the overall system latency.

Considering the different characteristics of MEC and MCC, some studies have also considered the option of combining both. In [15, 16], the authors studied the system architecture of an edge-cloud system. In [17], Lin et al. proposed a directional charging scheme and improved energy transfer model in the MEC system. In [18], Wang et al. consider the server allocation problem for edge computing system deployment where each edge cloud is modeled as an M/M/c queue. In [19, 20], authors study the computation, communication, and the storage resources problems in both IoV and MEC networks. Wang et al. in [21] consider using D2D technology in MEC system to collect larger and better quality sensing data.

For the problems in MEC and MCC scenarios, some extant studies have used RL or DRL as solutions. In [22], Wang et al. transform the edge caching problem as a Markov decision process and propose a distributed cache replacement strategy based on Q-learning to address the optimization problem. In [23], Su et al. propose a Q-learning-based spectrum access scheme to optimize spectrum and maximize the transmission rate. In [24], Dinh et al. propose a Q-learning-based scheme to solve the optimization problem in a multiuser multiedge-node computation offloading scenario. He et al. use a dueling DQN approach in [25] to solve joint optimization problem in connected vehicle networks, considering not only network gains but also caching and computation gain in the proposed framework. In [26], Wang et al. investigate the best strategy for resource allocation in ICWNs by maximizing spectrum efficiency and system capacity across the network and propose a DQN-based task offloading scheme for MEC networks in urban cities. In [27], Zhou et al. use a DDQN-based approach to solve the energy minimization problem and simultaneously efficiently approximate the value function.

Unlike these existing studies, the content of this paper mainly considers the problem of MEC and MCC collaboration in an IoV environment. Through the collaboration of edge-side and cloud-side servers, the paper is aimed at minimizing the energy consumption of the system. In order to solve the optimization problem of offloading decision and resource allocation, we propose two algorithms based on Q-learning method and DQN method, respectively.

3. System Model

In this section, we first present an edge-cloud network including mobile vehicles, RSUs equipped with MEC servers, and cloud servers (CS). Next, we give precise definitions of the model components.

3.1. Model Architecture. The edge-cloud network we consider is shown in Figure 1. The set of vehicles is denoted by $N = \{1, 2, \cdots, n\}$, and the set of MEC servers deployed at roadside units (RSUs) is denoted by $M = \{1, 2, \cdots, m\}$. In particular, we set that in this system time is divided into time slots of $t \in \{0, 1, 2, \cdots, T\}$, where T is the finite time horizon. And the computing task on a vehicle i $(i \in N)$ in time slot t is defined as $\Lambda_i(t) = \{S_i(t), C_i(t), D_i^{\max}(t)\}$, where $C_i(t)$ represents the total number of CPU cycles required to process the task, $S_i(t)$ represents the size of the computing task, and $D_i^{\max}(t)$ denotes the maximum delay tolerant of the task. Typically, MEC servers are deployed at the edge of the network, consisting of cellular networks to provide services to vehicles. The CS, on the other hand, is deployed away from vehicles and provides computing services through the core network. In order to guarantee the reliability of data transmission and provide offloading service for vehicles, RSUs and CS are connected via core networks. In general, the amount of computing resources and bandwidth is much higher on CS than on MEC servers, but the offload service is more costly on CS. The descriptions of the main symbols in this paper can be found in Table 1.

In the relevant scenario, there are multiple vehicles driving on the road within the coverage of CS and RSUs. Here, we denote $L_{i,j}(t)$ as the link between vehicle i and RSU j, where $L_{i,j}(t) = 1$ means that in time slot t vehicle i is in the coverage area of RSU j and vehicle i is associated with RSU j. In time slot t, vehicle i needs task offloading service. After obtaining its motion information and task information, the vehicle's offloading request is sent to its associated RSU. If its associated RSU does not have enough resource and cannot fulfill its requirement, its task information will be further sent to other RSUs in this area for cooperative offloading. If all the RSUs in this area fail to meet the task's demand for resources, the computing task will be offloaded to CS. We define the offloading decision of vehicle i as the integer variable $x_{i,j}(t) \in \{0, 1\}$, where $x_{i,j}(t) = 1$ means that the computing task of vehicle i is offloaded to RSUs in time slot t, and $x_{i,j}(t) = 0$ means that the computing task is offloaded to CS. Based on the current resource status of the RSUs, the offloading decision of the vehicle and the resource allocation are made dynamically by the control center in the CS. The task offloading process described above is shown in Figure 2.

3.2. Communication Model. Each vehicle can only be connected to one RSU at a time within the RSU's coverage range. We assign the bandwidth of $b_{i,j}$ to the link between vehicle i and RSU j. As shown in eq. (1), we calculate the data transmission rate according to Shannon's formula

$$R_{i,j}(t) = b_{i,j}(t) \log_2\left(1 + \frac{P_{i,j}^{tr} h_{i,j}}{\bar{w}_0}\right), \tag{1}$$

where $h_{i,j}$ denotes the channel gain, $P_{i,j}^{tr}$ denotes the transmission power, and $\bar{w}_0$ denotes the power level of white noise.

3.3. Computation Model

3.3.1. Computing Model for CS. For $x_{i,j}(t) = 0$, vehicle i decides to offload the computation task to CS. Although cloud servers have a huge volume of computing and communication resources, the amount of resources available in each current time slot is still limited considering the maintenance cost of the resources and other factors. The offloading process in this case is divided into the following parts: (i) task transmission between vehicle and its associated RSU, (ii) the process of uploading tasks to CS, and (iii) task processing on CS. According to eq. (1), we can obtain the data transmission rate between vehicle and its associated RSU as $R_{i,j}$. The data transmission rate between CS and RSUs can be obtained as $R_{i,o}$. To sum up, we can obtain the transmission times for the first two processes as

$$T_{i,j}^{tr} = \frac{S_i(t)}{R_{i,j}(t)}, \tag{2}$$

$$T_{i,o}^{tr} = \frac{S_i(t)}{R_{i,o}(t)}, \tag{3}$$

where $S_i(t)$ denotes the task size of the vehicle. We define $f_{i,o}(t)$ as the assigned computational capacity from CS to vehicle. Thus, we can obtain the computation time of this process as

$$T_{i,o}^{cp} = \frac{C_i(t)}{f_{i,o}(t)}, \tag{4}$$

where $C_i(t)$ denotes the required CPU cycles of task. Then, we define the total execution time to offload tasks to CS as $T_{i,j}$. Thus, we have

$$T_{i,j} = T_{i,j}^{tr} + T_{i,o}^{tr} + T_{i,o}^{cp}. \tag{5}$$

Based on the above discussion, the system cost when tasks are offloaded to CS is formulated as

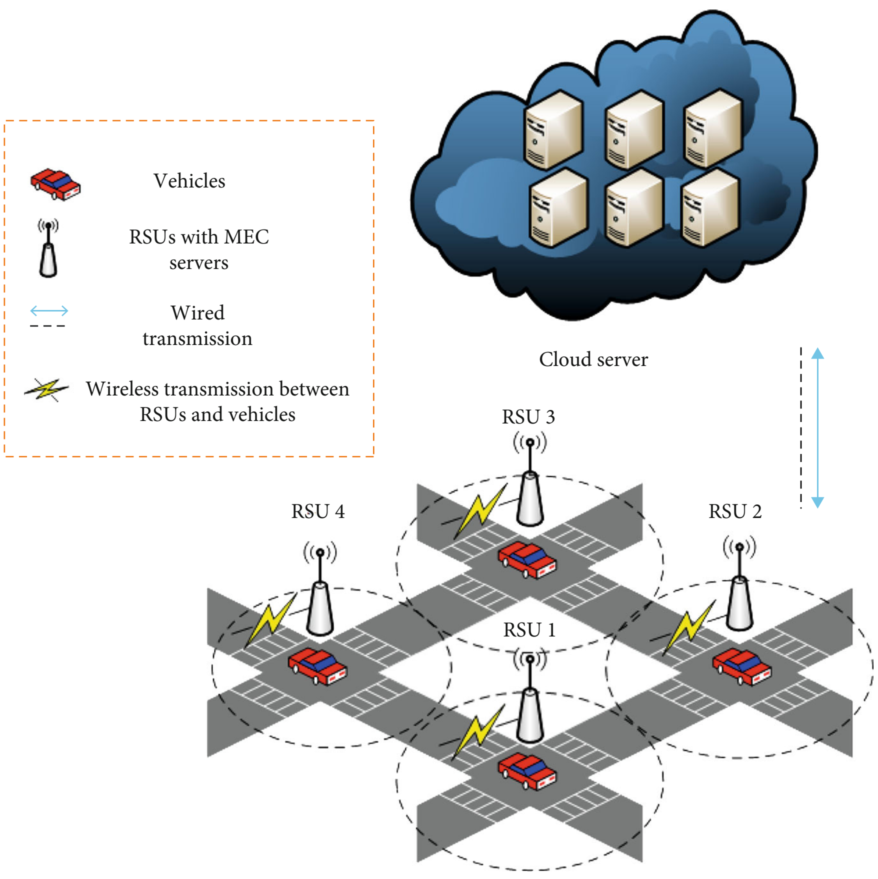

Figure 1: Network description.

Table 1

Notation	Definition
M	A set of RSUs
N	A set of vehicles
$P^{tr}_{i,j}$	The transmission power between vehicle and RSUs
$P^{tr}_{i,o}$	The transmission power between RSUs and CS
P_{CS}	The execution power of CS
P_{RSU}	The execution power of RSUs
$x_{i,j}(t)$	The offloading decision of vehicle i
$C_n(t)$	The CPU cycles of computing task for vehicle i
$S_n(t)$	The task size of computing task for vehicle i
$F_j(t)$	The total computational capability of RSU j
$B_j(t)$	The total radio capability of RSU j
$f_{i,j}(t)$	The computational resources of RSU j allocated to vehicle i
$b_{i,j}(t)$	The radio resources of RSU j allocated to vehicle i

$$E_1 = \sum_{i\in N}\left[\left(1-x_{i,j}(t)\right)\left(P^{tr}_{i,j}T^{tr}_{i,j}+P^{tr}_{i,o}T^{tr}_{i,o}+P_{CS}T^{cp}_{i,o}\right)\right], \qquad (6)$$

where $P^{tr}_{i,j}$ denotes the transmission power between vehicle and RSUs, $P^{tr}_{i,o}$ denotes the transmission power between RSUs and CS, and P_{CS} denotes the execution power of CS.

3.3.2. Computation Model for Associated Offloading. For $x_{i,j}(t)=1$ and $L_{i,j}(t)=1$, the vehicle's associated RSU has

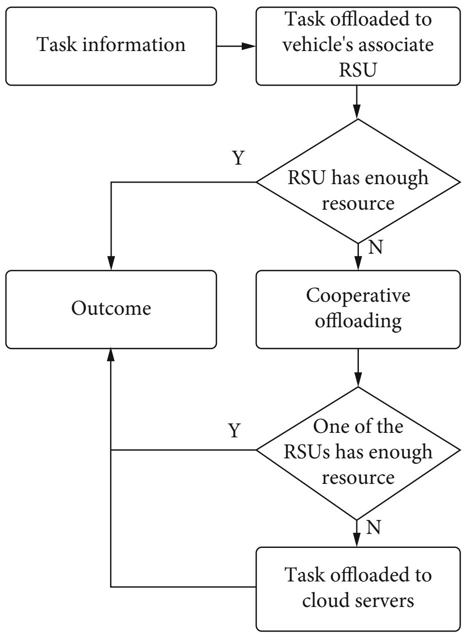
Task information
Task offloaded to vehicle's associate RSU
RSU has enough resource
Y
N
Outcome
Cooperative offloading
One of the RSUs has enough resource
Y
N
Task offloaded to cloud servers

Figure 2: Offload process.

enough resource to process its computing task, then vehicle i decides to offload the computation task to its associated RSU. Thus, the task processing in this case can be divided into two parts: (i) the transmission process from vehicle to its associated RSU and (ii) task processing on the associated RSU. Similar to the above, we have

$$T_{i,j}^{tr} = \frac{S_i(t)}{R_{i,j}(t)}, \tag{7}$$

$$T_{i,j}^{cp} = \frac{C_i(t)}{f_{i,j}(t)}, \tag{8}$$

where $f_{i,j}(t)$ denotes the assigned computational capacity. The total execution time can be obtained as

$$T_{i,j} = T_{i,j}^{tr} + T_{i,j}^{cp}, \tag{9}$$

Based on the above discussion, the system cost for associated offloading is formulated as

$$E_2 = \sum_{i\in N}\sum_{j\in M}\left[x_{i,j}(t)L_{i,j}(t)\left(P_{\mathrm{RSU}}T_{i,j}^{cp} + P_{i,j}^{tr}T_{i,j}^{tr}\right)\right], \tag{10}$$

where P_{RSU} denotes the execution power of RSUs.

3.4. Computation Model for Cooperative RSUs. For $x_{i,j}(t) = 1$ and $L_{i,j}(t) = 0$, since the associated RSU cannot meet its requirements in terms of resources, vehicle i decides to offload the computation task to cooperative RSU j. The task processing at this point consists of three processes: (i) the transmission process from vehicle to its associated RSU, (ii) the transmission process of the task between RSUs, and (iii) task processing on the target RSU j. We define the computational capacity assigned from target RSU to vehicle as $f_{i,j}(t)$, thus, we have

$$T_{i,j}^{tr} = \frac{S_i(t)}{R_{i,j}(t)}, \tag{11}$$

$$T_{r,r}^{tr} = \frac{S_i(t)}{R_{r,r}(t)}, \tag{12}$$

$$T_{i,j}^{cp} = \frac{C_i(t)}{f_{i,j}(t)}, \tag{13}$$

where $R_{r,r}$ denotes the transmission rate between RSUs.

The total task execution time in this case is expressed as

$$T_{i,j} = T_{i,j}^{tr} + T_{r,r}^{tr} + T_{i,j}^{cp}. \tag{14}$$

Combining the above discussion, the system cost when tasks are further offloaded to cooperative RSUs can be formulated as

$$E_3 = \sum_{i\in N}\sum_{j\in M}\left[x_{i,j}(t)\left(1 - L_{i,j}(t)\right)\left(P_{RSU}T_{i,j}^{cp} + P_{i,j}^{tr}T_{i,j}^{tr} + P_{r,r}^{tr}T_{r,r}^{tr}\right], \tag{15}$$

where $P_{r,r}^{tr}$ denotes the transmission power between RSUs.

4. Problem Formulation

In this section, we first calculate the total system cost based on the previous section. Then, we formulate an optimization problem, aiming to minimize the long-term system cost.

Based on the previous section, we define total system cost as follows

$$U(t) = E_1 + E_2 + E_3. \tag{16}$$

By jointly optimizing computational offloading and resource allocation in the proposed system, we formulate the optimization problem with minimizing long-term system cost as the optimization objective, which can be indicated as follows:

$$2(\mathscr{P}_1): \min_{x(t),f(t),b(t)} \bar{U} = \lim_{T\longrightarrow\infty}\frac{1}{T}\sum_{t=0}^{T-1}U(t), \tag{17}$$

$$s.t. x_{i,j}(t) \in \{0, 1\}, \tag{18}$$

$$T_{i,j} \le D_i^{\max}, \tag{19}$$

$$\sum_{n\in N} f_{i,j}(t) \le F_j(t), \tag{20}$$

$$\sum_{n\in N} b_{i,j}(t) \le B_j(t). \tag{21}$$

The meanings of the constraints are explained as follows:

(i) Constraint (18) indicates that the decision variable is a Boolean value

(ii) Constraint (19) guarantees that tasks need to be completed within the maximum time delay

(iii) Constraint (20) guarantees that the computation resources allocated by each RSU do not exceed its current available computation resources

(iv) Constraint (21) guarantees that the bandwidth allocated by each RSU does not exceed its current available bandwidth

According to the previous discussion, $x_{i,j}(t)$ represents the Boolean variable for the offloading decision. Meanwhile, $f_{i,j}(t)$ and $b_{i,j}(t)$ represent the computational resources and the bandwidth allocated for the task, respectively. Also, there are nonlinear conditions in the optimization problem. Therefore, optimization problem P_1 is a typical mixed integer nonlinear programming (MINLP) problem [28], which is an NP problem and cannot be solved in polynomial time.

5. Problem Transformation and Solution

In this section, we describe the optimization problem as a Markov decision process (MDP). Next, to solve the optimization problem based on Q-learning method and DQN method, we define state space, action space, and reward function of this problem.

5.1. State, Action, and Reward Definitions

(i) *State Space*. The state space indicates the current state of the environment in the system. In the concerned scenario, the system state of the available resources at current time slot is determined by $B_j(t) - \sum_{n\in N} b_{i,j}(t)$, and $F_j(t) - \sum_{n\in N} f_{i,j}(t)$, which, respectively, represent the available bandwidth and the available computation resources. Moreover, in order to compare among the states to determine if the system has reached the optimal state, we need to obtain the system cost $U(t)$ in each time slot. Hence, the state vector can be obtained as $z_t = (U(t), F_j(t) - \sum_{n\in N} f_{i,j}(t), B_j(t) - \sum_{n\in N} b_{i,j}(t))$

(ii) *Action Space*. In our concerned scenario, agents need to perform multiple actions including developing offloading decisions and deciding how much resource to allocate at each time slot. Therefore, the action vector consists of the offloading decision vector, the computation resource allocation vector, and the bandwidth resource allocation vector. Hence, the action vector in current time slot can be obtained as $d_t = (x_{i,j}(t), f_{i,j}(t), b_{i,j}(t))$

(iii) *Reward Function*. The optimization objective in this paper is to minimize the system cost, which is the opposite of the meaning of the system reward value. Therefore, we define the reward that agents can obtain at state z_t when performing action d_t as $r(z_t, d_t) = -U(z_t, d_t)$, where $U(z_t, d_t) = U(t)$ is the system cost of the current state

5.2. Markov Decision Process. In this step, we transform the optimization problem into a MDP problem where agents perform adaptive learning and decision making through iterative interactions with the unknown environment. The specific steps are as follows: first, an agent observes the current system state z_t. This intelligence performs action d_t based on the current policy π for each time slot. As a mapping from the current system state to action, policy π can be obtained as $\pi : \mathscr{Z} \longrightarrow p(\mathscr{D} = d \mid \mathscr{Z})$, where $\mathscr{D}$ denotes the set of actions and $\mathscr{Z}$ denotes the set of states. The probability of an agent moving to the next state z_{t+1} is $p(z_{t+1} \mid z_t, d_t)$, and the reward can be obtained as $r_t = r(z_t, d_t)$.

To summarize what was discussed above, a state value function $V^\pi(z_t)$ can be defined to indicate the long-term effect of the current state. Hence, the state value function $V^\pi(z_t)$ under the policy $\pi(z_t)$ can be expressed as

$$V^\pi(z_t) = \mathbb{E}_\pi[R_t \mid z_0 = z_t] = \mathbb{E}_\pi\left[\left(\sum_{t=0}^{\infty} \varphi^t r_t\right) \mid z_0 = z_t\right], \tag{22}$$

where z_0 denotes the initial state, and φ denotes the discounting factor indicating the importance of future rewards.

Finally, combined with the optimization objective, the agents need to obtain the optimal strategy in the current state to maximize the cumulative reward. Therefore, the optimization problem can be translated into an optimal state value function as

$$V^{\pi^*}(z_t) = \max_{\pi}\left[r(z_t, d_t) + \varphi \sum_{z_{t+1}} p(z_{t+1} \mid z_t, d_t) V^{\pi^*}(z_{t+1})\right]. \\ \text{s.t.constraintsin}(19) - (22) \tag{23}$$

Thus, the optimal action for state z_t can be obtained as

$$d_t^* = \underset{d_t}{\operatorname{argmax}}\, V^\pi(z_t, d_t). \tag{24}$$

5.3. Q-Learning-Based Solution. As an efficient value-based model-free iterative learning algorithm, the Q-learning approach enables agents to continuously approximate the optimal Q-value by learning the optimal action in the corresponding environment at each time slot. Specifically, agents of Q-learning method need to obtain the results of the state-value function for each policy and update the two-dimensional Q-table with the corresponding Q-values. Thus, agents can get the optimal strategy for each state based on the magnitude of the Q-values.

Specifically for the content of this paper, we use a Q-learning method to solve the optimization problem. The optimal action values can be defined as $Q(z, d)$, and the optimal state value function can be obtained as

$$V^{\pi^*}(z_t) = \max_{d_t} Q^\pi(z_t, d_t). \tag{25}$$

```
Input: state space $\mathscr{Z}$, action space $\mathscr{D}$, learning rate $\varepsilon$, discount factor $\varphi$
Output: the Q-values for every state-action pair
1:  Initialize $Q(z, d)$ arbitrarily for $\forall z \in \mathscr{Z}$, $d \in \mathscr{D}$
2:  for each episode do
3:    for each step of episode do
4:      In the current state $z_t$ choose an action with a random probability $\phi$
5:      If $\phi < \varepsilon$ then
6:        randomly select an action $d_t$
7:      else
8:        select $d_t = \underset{d}{\operatorname{argmax}} Q(z_t, d_t)$
9:      end if
10:     Execute action $d_t$, observe the reward $r_t$ and the next state $z_{t+1}$
11:     Update $Q(z_t, d_t)$ according to eq.(29)
12:     Update state $z_t \longleftarrow z_{t+1}$
13:   end for
14: end for
```

Algorithm 1: Q-learning-based joint computation offloading and resource allocation algorithm.

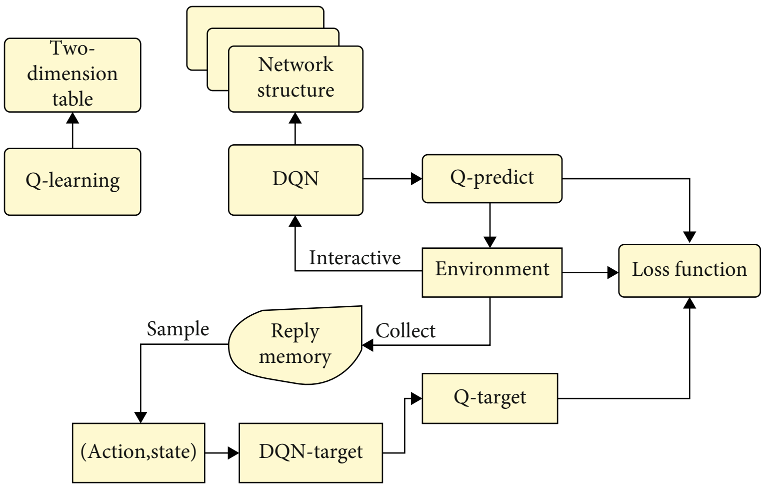

Figure 3: The network structure of DQN.

Therefore, the cumulative reward after performing action d_t can be obtained as

$$Q^{\pi}(z_t, d_t) = r(z_t, d_t) + \varphi \sum_{z_{t+1}} p(z_{t+1} \mid z_t, d_t) V^{\pi^*}(z_{t+1}). \quad (26)$$

Summarizing the two formulas above, the expected reward can be obtained as

$$Q^{\pi}(z_t, d_t) = r(z_t, d_t) + \varphi \sum_{z_{t+1}} p(z_{t+1} \mid z_t, d_t) \max_{d_{t+1}} Q^{\pi}(z_{t+1}, d_{t+1}). \quad (27)$$

The iterative formula for the optimal Q-value can be obtained by updating the state-action function as

$$Q(z_t, d_t) = Q(z_t, d_t) + \varepsilon \left[r(z_t, d_t) + \varphi \max_{d_{t+1}} Q(z_{t+1}, d_{t+1}) - Q(z_t, d_t) \right], \quad (28)$$

where $\varepsilon \in (0, 1)$ denotes the learning rate parameter.

Combining the above discussion, our proposed algorithm is shown in Algorithm 1. In order to make a trade-off between the exploration and the exploitation, we use ε-greedy strategy to choose actions [29].

5.4. DQN-Based Solution. In the time-varying scenarios, we consider the number of vehicles and the size of the tasks are stochastic in nature. This leaves the possibility of a huge action-state space, where although the above Q-learning-based solution can obtain the best policy by updating the Q-table, it may lead to a dimensional disaster in real scenarios. If we stick to the above Q-learning-based solution,

```
1:  **Initialize** replay memory set $D$
2:  **Initialize** action-value function with random weights $\theta$
3:  **Initialize** target action-value function with weights $\theta^- = \theta$
4:  **for** episode =1, M
5:    **Initialize** sequence $z_1 = \{r_1\}$ and preprocessed sequence $\tau_1 = \tau(z_1)$
6:    **for** t =1,2,...,T **do**
7:      With probability $\varepsilon$ select a random action $d_t$
8:      Otherwise select $d_t = \underset{d}{\text{argmax}} Q(z_t, d_t ; \theta)$
9:      Execute action $d_t$, observe the reward $r_t$ and the next state $z_{t+1}$
10:       Set $z_{t+1} = z_t, d_t, r_{t+1}$ and preprocess $\tau_{t+1} = \tau(z_{t+1})$
11:       Store experience $(\tau_t, d_t, r_t, \tau_{t+1})$ in $D$
12:       Sample random minibarch of experience $(\tau_i, d_i, r_i, \tau_{i+1})$ from $D$
13:       Set $y_i = r_i$ if episode terminates at step $i+1$
14:       Otherwise $y_i = r_{t+1} + \varphi \max_d Q(\tau_{i+1}, d_{i+1} ; \theta_i)$
15:       Perform a gradient descent step on $(y_t - Q(\tau_i, d_i ; \theta))^2$ with respect to the network parameters $\theta$
16:       Every $C$ step reset
17:     **end for**
18:  **end for**
```

Algorithm 2: DQN-based joint computation offloading and resource allocation algorithm.

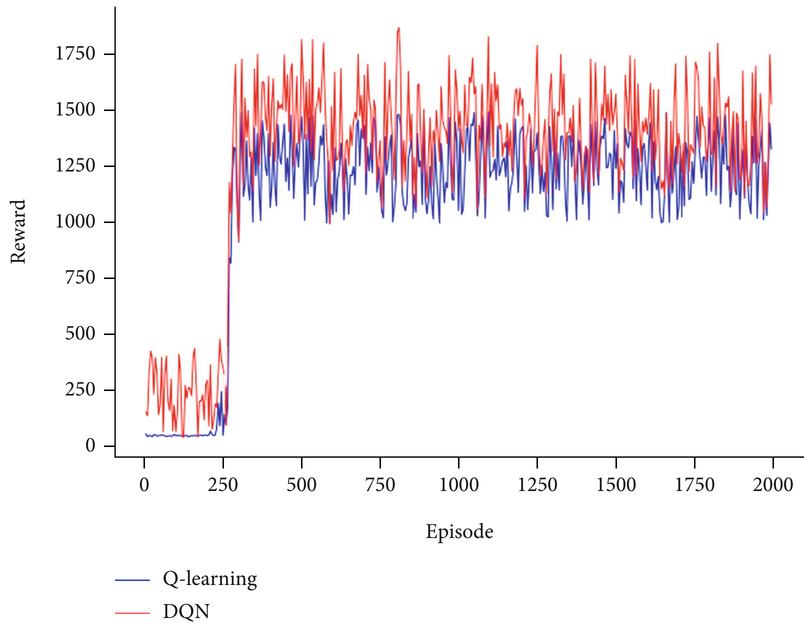

Figure 4: Convergence performance.

finding the corresponding Q-value in a huge Q table can be costly in time and memory.

To avoid this drawback of Q-learning method, we further use a DQN-based approach to solve the optimization problem. Compared to Q-learning, DQN is essentially an improvement method. As a value function approximation, in order to solve the problem of large state space, also known as dimensional disaster, DQN uses the architecture of deep neural network (DNN) to replace Q-table. As a nonlinear approximator of the optimization problem, the DNN in DQN can capture the complex interaction between states and actions [30]. After taking the states as the input to the DQN network, we can get the Q-value of the actions as the output. By doing so, we can estimate the Q-value as $Q(z_t, d_t) \approx Q(z_t, d_t ; \theta)$, where θ are the weights of the DQN. Therefore, as in eq. (25), the optimal action in this method can be obtained as

$$d_t^* = \underset{d_t}{\text{argmax}}\, V^\pi(z_t, d_t ; \theta). \tag{29}$$

In actual engineering application, DQN mainly needs to solve two obvious problems: low sample utilization rate and unstable value obtained by training. In order to deal with these two problems, DQN uses the following two key technologies

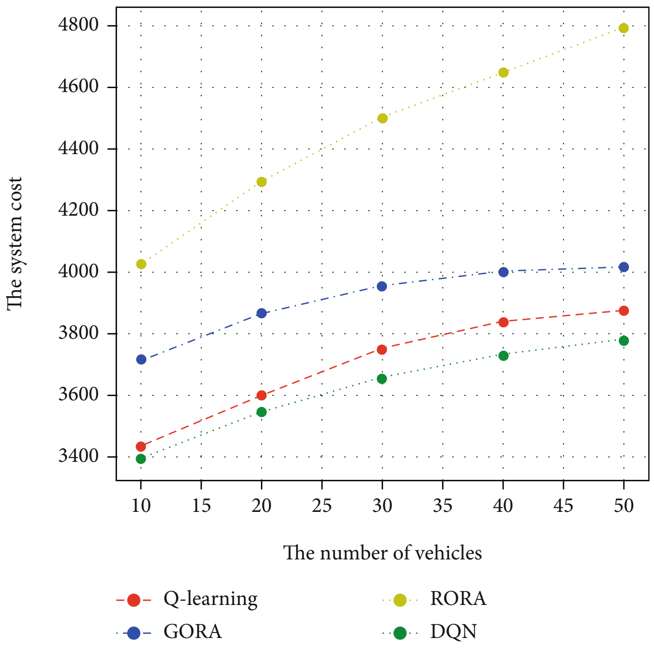

Figure 5: Effects of different number of vehicles on system cost.

(i) *Experience Replay*. An experience pool is constructed to remove data correlations which is a dataset consisting of the recent experiences of the intelligences

(ii) *Freezing Q-Target Networks*. The parameters in the goal are fixed for a time period (or for a fixed number of steps) to stabilize the learning goal

Next, we describe the specific execution steps of DQN. Figure 3 shows the network structure of DQN and the difference between DQN and Q-learning. The network first outputs a prediction Q-value $Q(z_t, d_t)$, then selects the next action based on this Q-value and passes it into the environment for interaction, then obtains a new state value and continues to feed it into the training. At the same time, the results of each interaction with the environment are stored in a fixed-length experience pool. A target Q-network with the same structure and parameters is copied from the Q-network at certain steps to stabilize the output target, and the target Q-network samples the data from the experience pool to output a stable target value y_t. And y_t can be obtained as

$$y_t \equiv r_{t+1} + \varphi \max_d Q(z_{t+1}, d_{t+1}; \theta_t). \tag{30}$$

And the update of the value function of Q-value can be obtained as

$$Q(z_t, d_t) \longleftarrow Q(z_t, d_t) + \varepsilon[y_t - Q(z_t, d_t)]. \tag{31}$$

DQN approximates the value function using a deep convolutional neural network. The value function here corresponds to a set of parameters, which in a neural network are the weights of each layer of the network, denoted by θ. At this point, updating the value function is actually updating the parameter θ. When the neural network is determined, θ indicates the value function. And the update method of θ is the gradient descent, which can be expressed as

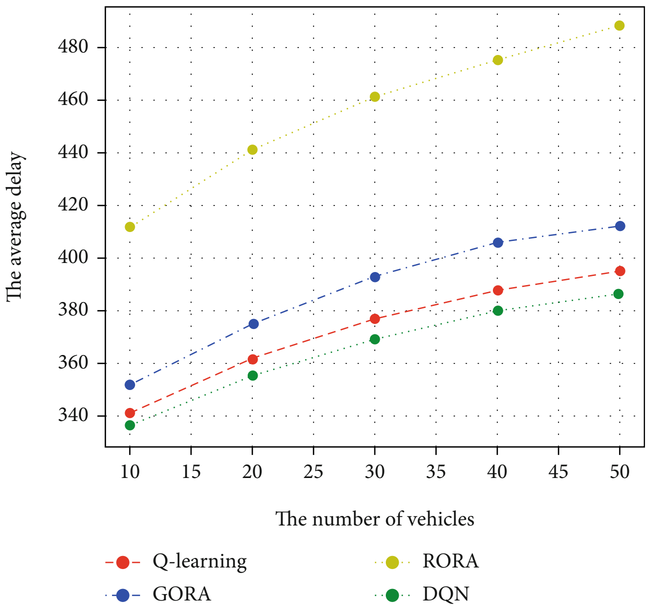

Figure 6: Effects of different number of vehicles on average delay.

$$\theta_{t+1} = \theta_t + \varepsilon[y_t - Q(z_t, d_t; \theta_t)]\nabla_{\theta_t} Q(z_t, d_t; \theta_t). \tag{32}$$

In each training iteration, we train the DQN network by minimizing the loss function. In the previous Q-learning based method, we updated the Q-table by iterating through it using the rewards and the current Q-table at each step. Then, we can use this calculated Q-value as

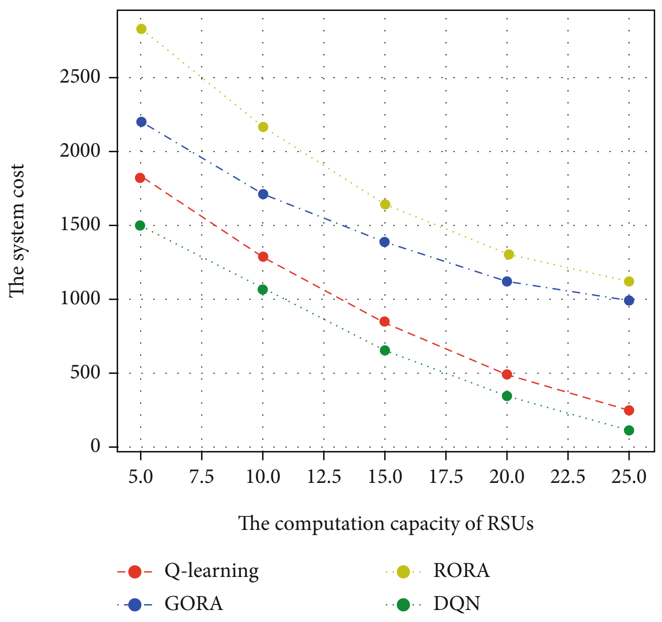

Figure 7: Effects of different computation capacity of RSUs on system cost.

the label to design the loss function, and we use the mean squared difference between the approximate and true values to represent the loss function, which can be obtained as

$$Loss(t) = \mathbb{E}_{\pi}\left[\left(Q(z_t, d_t) - Q(z_t, d_t\,; \theta_t)\right)^2\right]. \qquad (33)$$

To summarize the above about DQN, the specific algorithm steps are shown in Algorithm 2. Same as the Q-learning-based method, we use ε-greedy strategy in the selection of actions.

6. Performance Evaluation

In this section, we evaluate numerical results of the proposed joint computation offloading and resource allocation algorithm in a dynamic edge-cloud network and compare it with other typical schemes.

6.1. Simulation Settings. In the simulation experiments, we consider a dynamic scenario in which there are several vehicles driving in this area and several RSUs distributed on the roadside. Similar to the experimental in [31], for each vehicle, the required CPU cycles of the computing tasks are randomly selected in the range of [0.4, 0.6, 0.7, 0.8, 0.3, 0.2, 0.8, 0.9, 0.4, 0.5, 0.2, 0.3, 0.8, 0.9, 0.4] Gcycles.

6.2. Simulation Results. In the next simulation experiments, in general, our proposed schemes are compared with the random offloading and resource allocation (RORA) scheme and greedy offloading and resource allocation (GORA) scheme [32].

Figure 4 reveals the convergence performance of our two proposed algorithms. In terms of general trends, both curves tend to increase in reward value and then converge after a period of time. However, the number of training episodes

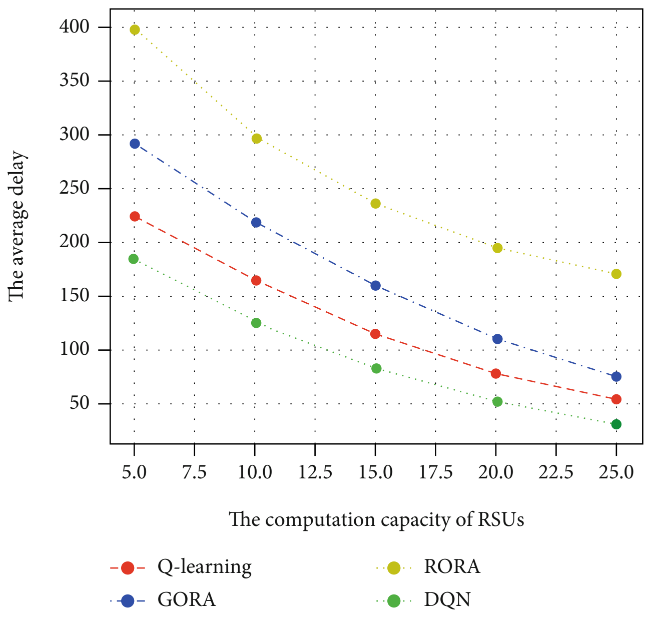

Figure 8: Effects of different computation capacity of RSUs on average delay.

to reach convergence and the final convergence to the reward value differ due to the difference between these two methods. What can be seen from the figure is that the Q-learning method reaches convergence after about 280 training episodes, while the curve of the DQN method reaches convergence after about 250 training episodes. In addition, the DQN-based approach allows for higher reward values at the time of final convergence.

Figures 5 and 6 show how the total system cost and average delay is affected by changes in the number of vehicles, respectively. We intercepted the curve with independent variables changing in the range of 10-50. According to the

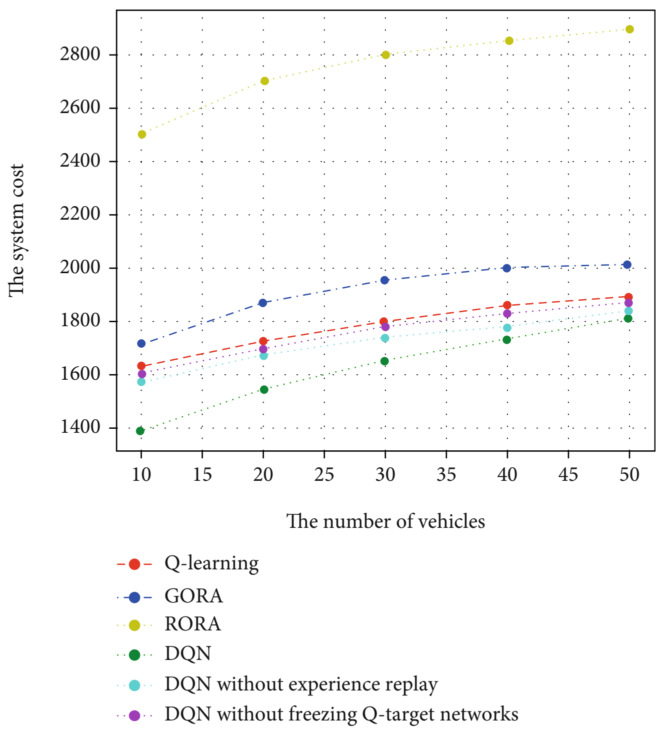

Figure 9: The impact of these two key technologies in evaluation.

optimization objective equation, the number of vehicles has direct effect on the system cost and average delay, so the general trend of all curves is that the system cost and average delay are positively related to the number of vehicles. As joint optimization schemes of computation offloading, the system cost of our proposed Q-learning based algorithm and DQN-based method keeps increasing, and they always work better compared to the other two schemes in the figure. This is because our proposed schemes consider the offloading decision making of RSUs and CS cooperatively, by which the utilization efficiency of network resources can be increased and system cost has been greatly reduced. For example, when the number of vehicles equal to 30, our proposed Q-learning-based method reduces the cost of the system by about 16% and 5% compared to the classical solutions RORA and GORA, respectively. And in the same case, compared with the classical two schemes, our proposed DQN-based scheme reduces the system cost by 18% and 7.5%. When we take the average delay as the dependent variable, the performance improvement of our proposed Q-learning-based method relative to RORA and GORA can reach 18% and 4%. And likewise, the performance improvement of our proposed DQN-based method relative to RORA and GORA can reach 19.9% and 9%.

Figures 7 and 8 show how the total system cost and the average delay are affected by changes in the computation capacity of RSUs, respectively. Compared to the two graphs above, the curves in Figures 7 and 8 show more dramatic changes. In general, the system cost and average latency are reduced with the increase of the computation capacity of RSUs. However, the performance varies due to the different schemes. Obviously, our two proposed solutions have some performance advantages. Specifically, for computational resources, when the computation capacity of RSUs is equal to 30 (in GHz), when compared to RORA and GORA, the performance improvement of our proposed Q-learning-based method can reach 49% and 42%. And the performance improvement of our proposed DQN-based method relative to RORA and GORA can reach 62% and 57%.

In addition, in order to reflect the impact of two key techniques, experience replay and freezing Q-target networks, on DQN method, we tried to add two experiments without one key technology under the same experimental conditions, and the effect is shown in Figure 9. Since the impact of the above two techniques on DQN is mainly in eliminating data correlation and speeding up convergence, the effect of the scheme after removing these two techniques, respectively, is similar to Q-learning method. From the evaluations, these two key technologies have a relatively obvious performance additive effect on the DQN method. Without these two key techniques, DQN method's performance is close to Q-learning method.

In summary, we have compared our proposed scheme with RORA and GORA. From the experimental results, it is clear that the proposed scheme minimizes the system cost and has some advantage over the other two schemes. For the proposed two schemes, compared with the Q-learning-based method, the DQN-based method has a better performance due to the advantages of using deep neural networks.

7. Conclusion

In this paper, we propose a joint computation offloading and resource allocation scheme for the edge-cloud network. The optimization problem aims to minimize the system cost, including the computation cost and the radio cost. Meanwhile, we also consider the capacity constraints and the latency constraints. Then, we transform the original optimization problem into a Mixed Integer Nonlinear Programming problem. Eventually, a Q-learning-based method for computation offloading and resource allocation is developed to enable tractable analysis. To avoid the dimensionality catastrophe due to the two-dimensional table structure of Q-learning, we further propose a DQN-based algorithm to solve the optimization problem. Through a series of comparative experiments, it is clear that the proposed schemes have good performances in system cost minimization.

Data Availability

The processed data required to reproduce these findings cannot be shared at this time as the data also forms part of an ongoing study.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China (Grant no. U1703261). The corresponding author is Shouzhi Xu.

References

[1] J. Zhao, S. Ni, L. Yang, Z. Zhang, Y. Gong, and X. You, "Multiband cooperation for 5G HetNets: a promising network paradigm," *IEEE Vehicular Technology Magazine*, vol. 14, no. 4, pp. 85–93, 2019.

[2] P. Wang, Z. Zheng, B. Di, and L. Song, "HetMEC: latency-optimal task assignment and resource allocation for heterogeneous multi-layer mobile edge computing," *IEEE Transactions on Wireless Communications*, vol. 18, no. 10, pp. 4942–4956, 2019.

[3] P. Wang, R. Yu, N. Gao, C. Lin, and Y. Liu, "Task-driven data offloading for fog-enabled urban IoT services," *IEEE Internet of Things*, vol. 8, no. 9, pp. 7562–7574, 2021.

[4] X. Chen, W. Li, S. Lu, Z. Zhou, and X. Fu, "Efficient resource allocation for on-demand mobile-edge cloud computing," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 9, pp. 8769–8780, 2018.

[5] J. Du, L. Zhao, J. Feng, X. Chu, and F. R. Yu, "Economical revenue maximization in cache enhanced mobile edge computing," in *2018 IEEE International Conference on Communications (ICC)*, pp. 1–6, Kansas City, MO, USA, 2018.

[6] Z. Ning, X. Wang, and J. Huang, "Mobile edge computing-enabled 5G vehicular networks: toward the integration of communication and computing," *IEEE Vehicular Technology Magazine*, vol. 14, no. 1, pp. 54–61, 2019.

[7] H. Zhou, X. Chen, S. He, J. Chen, and J. Wu, "DRAIM: a novel delay-constraint and reverse auction-based incentive mechanism for WiFi offloading," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 4, pp. 711–722, 2020.

[8] S. Nath and J. Wu, "Deep reinforcement learning for dynamic computation offloading and resource allocation in cache-assisted mobile edge computing systems," *Intelligent and Converged Networks*, vol. 1, no. 2, pp. 181–198, 2020.

[9] C. Xu, W. Zhao, L. Li, Q. Chen, D. Kuang, and J. Zhou, "A Nash Q-learning based motion decision algorithm with considering interaction to traffic participants," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 11, pp. 12621–12634, 2020.

[10] S. Mao, S. Leng, and Y. Zhang, "Joint communication and computation resource optimization for NOMA-assisted mobile edge computing," in *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, pp. 1–6, Shanghai, China, 2019.

[11] Z. Ning, P. Dong, X. Kong, and F. Xia, "A cooperative partial computation offloading scheme for mobile edge computing enabled internet of things," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4804–4814, 2019.

[12] Z. Kuang, L. Li, J. Gao, L. Zhao, and A. Liu, "Partial offloading scheduling and power allocation for mobile edge computing systems," *IEEE Internet of Things Journal*, vol. 6, no. 4, pp. 6774–6785, 2019.

[13] L. T. Tan and R. Q. Hu, "Mobility-aware edge caching and computing in vehicle networks: a deep reinforcement learning," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 11, pp. 10190–10203, 2018.

[14] S. Bi, L. Huang, and Y. A. Zhang, "Joint optimization of service caching placement and computation offloading in mobile edge computing systems," *IEEE Transactions on Wireless Communications*, vol. 19, no. 7, pp. 4947–4963, 2020.

[15] H. Liu, F. Eldarrat, H. Alqahtani, A. Reznik, X. de Foy, and Y. Zhang, "Mobile edge cloud system: architectures, challenges, and approaches," *IEEE Systems Journal*, vol. 12, no. 3, pp. 2495–2508, 2018.

[16] N. Cheng, W. Xu, W. Shi et al., "Air-ground integrated mobile edge networks: architecture, challenges, and opportunities," *IEEE Communications Magazine*, vol. 56, no. 8, pp. 26–32, 2018.

[17] C. Lin, Z. Yang, H. Dai, L. Cui, L. Wang, and G. Wu, "Minimizing charging delay for directional charging in Wireless Rechargeable Sensor Networks," *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, pp. 1819–1827, 2019.

[18] E. Wang, D. Li, B. Dong, H. Zhou, and M. Zhu, "Flat and hierarchical system deployment for edge computing systems," *Future Generation Computer Systems*, vol. 105, no. 2020, pp. 308–317, 2020.

[19] K. Zhang, Y. Mao, S. Leng, Y. He, and Y. ZHANG, "Mobile-edge computing for vehicular networks: a promising network paradigm with predictive off-loading," *IEEE Vehicular Technology Magazine*, vol. 12, no. 2, pp. 36–44, 2017.

[20] C. Liang, Y. He, F. R. Yu, and N. Zhao, "Enhancing QoE-aware wireless edge caching with software-defined wireless networks," *IEEE Transactions on Wireless Communications*, vol. 16, no. 10, pp. 6912–6925, 2017.

[21] P. Wang, Z. Yu, C. Lin, L. Yang, Y. Hou, and Q. Zhang, "D2D-enabled reliable data collection for mobile crowd sensing," in *2020 IEEE 26th International Conference on Parallel and Distributed Systems (ICPADS)*, pp. 180–187, Hong Kong, 2020.

[22] W. Wang, R. Lan, J. Gu, A. Huang, H. Shan, and Z. Zhang, "Edge caching at base stations with device-to-device offloading," *IEEE Access*, vol. 5, pp. 6399–6410, 2017.

[23] Z. Su, M. Dai, Q. Xu, R. Li, and S. Fu, “Q-learning-based spectrum access for content delivery in mobile networks,” *IEEE Transactions on Cognitive Communications and Networking*, vol. 6, no. 1, pp. 35–47, 2020.

[24] T. Q. Dinh, Q. D. La, T. Q. S. Quek, and H. Shin, “Learning for computation offloading in mobile edge computing,” *IEEE Transactions on Communications*, vol. 66, no. 12, pp. 6353–6367, 2018.

[25] Y. He, N. Zhao, and H. Yin, “Integrated networking, caching, and computing for connected vehicles: a deep reinforcement learning approach,” *IEEE Transactions on Vehicular Technology*, vol. 67, no. 1, pp. 44–55, 2018.

[26] D. Wang, H. Qin, B. Song, X. Du, and M. Guizani, “Resource allocation in information-centric wireless networking with D2D-enabled MEC: a deep reinforcement learning approach,” *IEEE Access*, vol. 7, pp. 114935–114944, 2019.

[27] H. Zhou, K. Jiang, X. Liu, X. Li, and V. C. M. Leung, “Deep reinforcement learning for energy-efficient computation offloading in mobile edge computing,” *IEEE Internet of Things Journal*, 2021.

[28] A. Heidari, Z. Y. Dong, D. Zhang, P. Siano, and J. Aghaei, “Mixed-integer nonlinear programming formulation for distribution networks reliability optimization,” *IEEE Transactions on Industrial Informatics*, vol. 14, no. 5, pp. 1952–1961, 2018.

[29] Q. Chen, X. He, and W. Meng, “Air-ground cooperative access control algorithm based on Q-learning,” in *International Conference on Computing, Networking and Communications*, pp. 461–465, Big Island, HI, USA, 2020.

[30] H. Zhou, T. Wu, H. Zhang, and J. Wu, “Incentive-driven deep reinforcement learning for content caching and D2D offloading,” *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 8, pp. 2445–2460, 2021.

[31] Z. Ning, K. Zhang, X. Wang et al., “Joint computing and caching in 5G-envisioned internet of vehicles: a deep reinforcement learning-based traffic control system,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 8, pp. 5201–5212, 2021.

[32] H. Zhou, X. Chen, S. He, C. Zhu, and V. C. M. Leung, “Freshness-aware seed selection for offloading cellular traffic through opportunistic mobile networks,” *IEEE Transactions on Wireless Communications*, vol. 19, no. 4, pp. 2658–2669, 2020.