DQN-Based Applications Offloading with Multiple Interdependent Tasks in Mobile Edge Computing

Jiaxue Tu[1], Dongge Zhu[2], Yunni Xia[1](✉), Yin Li[3], Yong Ma[4], Fan Li[5], and Qinglan Peng[6]

[1] College of Computer Science, Chongqing University, Chongqing, China
xiayunni@hotmail.com
[2] Electric Power Research Institute of State Grid Ningxia Electric Power Co., Ltd., Yinchuan, Ningxia, China
[3] Guangzhou Institute of Software Application Technology, Guangzhou, China
[4] School of Computer and Information Engineering, Jiangxi Normal University, Nanchang, Jiangxi, China
[5] Sichuan University, Chengdu, China
[6] School of Artificial Intelligence, Henan University, Zhengzhou, China

Abstract. Recently, Vehicular Edge Computing (VEC) is evolving as a solution for offloading computationally intensive tasks in in-vehicle environments. However, when the number of vehicles and users is large, pure edge resources may be insufficient and limited, most existing work focuses on minimizing system latency by designing some offloading strategies. Therefore, hybrid multilayer edge structures are in dire require of mission deployment strategies that can synthesize cost and mission latency. In this paper, we argue that each application can be decomposed into multiple interdependent subtasks, and that the different subtasks can be deployed separately into different edge layers in a hybrid three-tier edge computing infrastructure for execution. We develop an improved DQN task deployment algorithm based on Lyapunov optimization to jointly optimize the average workflow latency and cost under a long-term cost constraint, and simulation results clearly show that, comparing with the traditional approach, our proposed method effectively reduces the cost consumption by 92.8% while sacrificing only some latency.

Keywords: VEC · Computation Offloading · Latency · Lyapunov Optimization · DQN

1 Introduction

Mobile Internet computation is rapidly growing, many new services, such as cloud computing and edge computing [1], continue to emerge, AR/VR. Meanwhile vehicles are becoming more and more important in providing computing power and access to the Internet, where mobile vehicles can collect data and

H. Gao et al. (Eds.): CollaborateCom 2023, LNICST 561, pp. 77–92, 2024.
https://doi.org/10.1007/978-3-031-54521-4_5

perform a number of intensive computational tasks to make intelligent decisions [2–4]. However, these compute-intensive applications often result in overloaded and sluggish mobile services because mobile devices typically have less capacity and resources than traditional centralized cloud centers. In this case, the user may have a poorer quality of experience when in-vehicle services are available [5,6], which may even lead to some serious traffic accidents [2,7].

To solve the problem of mismatch between vehicle resource demand and limited resources, the model called mobile cloud computing, has been proposed. Mobile cloud computing combines cloud computing with the mobile environment. In this model, when the local vehicle's resources are overloaded, the vehicle offloads the application to a traditional cloud server over a wireless network [8]. While utilizing remote computing resources on cloud servers can be effective in reducing computational latency, communication latency can increase [9,10].

Mobile Edge Computing (MEC) is coming out to face these challenges. That is, pushing services to the edge of the network. By placing services closer to the vehicle, MEC perfectly complies with the low transmission latency requirements of applications and effectively eases the demand for computing resources in the vehicle [11]. Vehicular Edge Computing (VEC) has received more and more attention in recent years, for VEC growing as an important branch of MEC [12], because it improved cloud computing capabilities to the vehicle. As shown in Fig. 1, in order to provide a high quality of service to the users, lots of cloud services are placed on the units of roadside, which can be achieved by the vehicles through network. In this solution, compute-intensive and latency-sensitive applications can execute on the RSUs.

Recently, extensive research efforts were paid on VEC-based task offloading and scheduling. A dynamic task allocation scheme is be proposed by yang et al. [13], it ensure quality of service. He et al. consider use game theoretic methods to solve task allocation problems and task allocation as a gaming system [14], Chiara Caiazzade et al. [15] studied the LTE nodes' energy consumption in different communication scheme of Request-Response. An efficient predictive combinatorial model to reduce offloading cost is proposed in Zhang et al. [19]. Most of the previous studies have assumed applications composed of independent tasks, and few studies have considered the dependency of task, Zhou et al. [16] obtain a deployment solution, prioritizing applications that meet deadline constraints and tasks that meet dependency constraints. Shabidani et al. [18] considered load balancing under task scheduling in the edge-fog-cloud architecture. By the way, the application of deep learning has become more and more hot, and the research of using deep learning to solve related problems has gradually increased. To achieve the a low latency and energy consumption, Wu et al. [20] uses a deep Q-network based offloading strategy to offload tasks. Zhang et al. [21] proposed the use of a convex optimized DQN model to reduce task latency in mobile edge environments.

Most of the existing work focuses on minimizing system latency by designing some offloading strategies, but these strategies do not consider server pricing. To optimize the balance between system latency and operational cost for multiple on-board applications, it is important to take server pricing into account when designing task scheduling policies.

To address this problem, we incorporate cost constraints into the design of VEC-based time-delay systems and we propose a deep learning method using Lyapunov optimization. And the experimental results show that our proposed scheme is significantly better than the traditional one under the same constraints. Specifically, the proposed algorithm can effectively reduce the system cost by up to 92.8% compared to the traditional scheme, although at the slight expense of the system delay. The major innovations of this work are as follows:

– We develop a DQN-based data offloading algorithm based on Lyapunov optimization and demonstrate that it outperforms both the traditional scheme and the basic DQN algorithm, for minimizing the total delay under a long-term cost constraint.
– We use the improved DQN as the key subroutine to solve the offloading problem and achieve joint optimization under system delay, cost consumption, and long-term cost optimization. Simulation results show that our algorithm maintains a low cost based on a lower level of user latency, although it sacrifices some user delays by deploying a reasonable task offloading policy.

This paper is organized as: in the Sect. 2, we will discuss the VEC based edge-cloud hybrid network architecture and then presents the problem. Section 4 gives the optimization problem solving method based on DQN and Lyapunov optimization algorithm. Numerical analysis is presented in Sect. 5.

2 System Model

2.1 Network Model

The environment is made up of a three-layer infrastructure, i.e., the cloud layer, the local layer, and the VEC layer. Figure 1 shows the hybrid three-layer architecture for Dependent Aware Vehicle task deployment. The VEC layer comprise R RSUs, each of which are with a location of $[RL_x(j), RL_y(j)]$. The j^{th} RSU covers a circular area with a radius of r_j and the j^{th} RSU is equipped with s_j edge servers $(0<j \leq R)$.

Each vehicle is placed in a different location on a two-dimensional plane as well as on different roads, it's position can be described as $[VL_x(i), VL_y(i)]$. The conditions that need to be met for the rsu $j \in R$ to be accessible to the car i when it is running to a certain location are

$$r_j \geq \sqrt{(RL_x(j) - VL_x(i))^2 + (RL_y(j) - VL_y(i))^2} \tag{1}$$

The number of vehicles is M and the i^{th} car's speed is $v_i (0<i \leq M)$. Each vehicle has one application to be executed. Each application consists of interdependent tasks, and the i^{th} application's size of data is d_i.

The vehicles on the different roads can visit the j^{th} RSU through wireless channel, and every vehicle can access the cloud server at any time from any

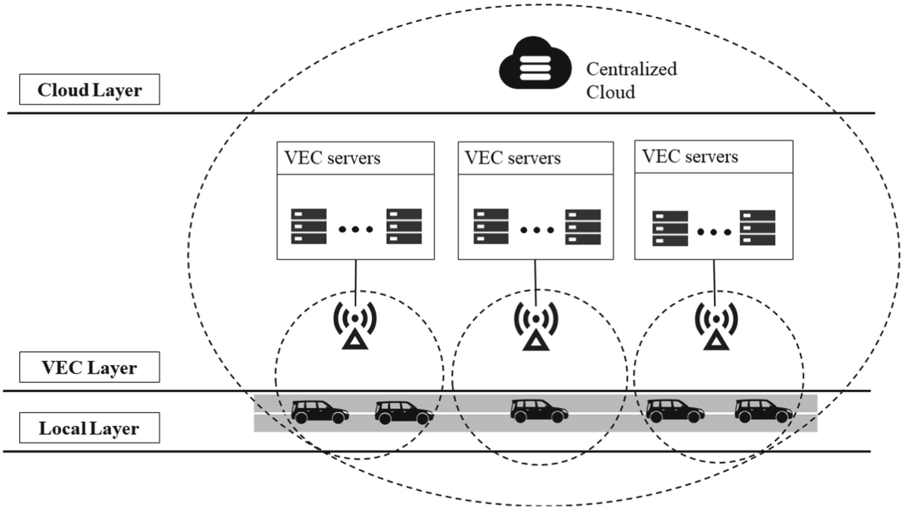

Fig. 1. The system model.

location. We assume centralized cloud has a limited number of servers and infinite resources. Each vehicle can make a choice to place their tasks to the centralized cloud servers or the VEC servers.

2.2 Performance Model

This part describes the performance evaluation model of the system. We assume applications are structured rather than monolithic applications, so each task can be pushed to different layers and the structure of each application is defined based on a directed acyclic graph of scientific workflows. The i^{th} application is assumed to be on the i^{th} vehicle, and it contains $K(i)$ tasks, $T_{i,k}$ is denoted as the k^{th} task $(0<k \leq K(i))$ of the i^{th} application.

We also consider Each computation unit, including centralized cloud, RSU and local computer, has its own cost consumption p_a. To minimize the cost of in-vehicle applications and users on edge servers while reducing average task latency, we try to find some of the offloading strategies at a macro level that reduces both the average elapsed time of all applications and the cost consumption of the server. The transfer time of the calculation results is negligible because the data size of an application's calculation results is much smaller than the size of the input data. We denote $MakeSpan_i$ as the total latency of application A_i and $cost_a$ as the total cost of local computing, MEC or centralized cloud. We will describe $cost_a$ in the next section. Time of task $T_{i,k}$ can be apart into two parts in our makespan model, named uplink delay and Application Makespan.

Uplink Delay. When task $T_{i,k}$ is choosed to be performed on VEC servers or the centralized cloud, the uplink delay needs to be taken into account due to the distance or limited bandwidth between them.

$$Rate_i^j = BV2E \cdot \log_2(1 + \frac{P_i \cdot g_j^i}{N_0}) \tag{2}$$

$BV2E$ denotes the bandwidth among vehicles and VEC servers, N_0 indicates the noise power, and P_i represents transmission power of the vehicle i, and g_j^i denotes the bandwidth among vehicle i and RSU j.

For VEC computation, the uplink delay of k^{th} task with data size $d_{i,k}$ of application A_i to VEC servers can be given as

$$t_{i,k}^{uplinkE} = \frac{d_{i,k}}{Rate_i^j} \tag{3}$$

Then, we define $BV2C$ as the bandwidth among vehicles and the remote cloud server. The transmission rate of the remote cloud server and the uplink delay from the vehicle to the centralized cloud server can be expressed as follows:

$$Rate_c^j = BV2C \cdot \log_2(1 + \frac{P_i \cdot g_i^c}{N_0}) \tag{4}$$

$$t_{i,k}^{uplinkC} = \frac{d_{i,k}}{Rate_i^c} \tag{5}$$

Application Makespan. The servers that choosed by the vehicle i^{th} can be described as Des_i, $Des_i = \{1, 2, ..., S + 2\}$, the number of servers in the choosed RSU is S. And 1 represents compute task in local, that is, the task is computed on the vehicle's local system, deploying locally will have very low cost but high latency; $S + 2$ indicates deploy task on the centralized cloud server where it is very fast but also expensive to execute the task; 2 to $S + 1$ means execute it on VEC servers and tasks executed on vehicle edge servers will enjoy some speed over local blocks and will cost less than in the cloud.

A task $T_{i,k}$, it's readiness time is the latest time at which all of its predecessors have been done, then the ready time $RT_{i,k}$ can be be expressed as follows:

$$RT_{i,k} = \max_{T_{i,h} \in pre(T_{i,k})} RFT_{i,h} \tag{6}$$

$T_{i,h} \in pre(T_{i,k})$ means the pre task must be completed before task $T_{i,k}$ is ready and $pre(T_{i,k})$ indicates all parent tasks of the current task $T_{i,k}$. When all the pre tasks have been completed, the present task $T_{i,h}$ is ready to transmit.

Then we denoted $AT_{i,k,s}$ as the earliest time that one of the server $s \in Des_i$ is accessible for task $T_{i,k}$. The starting time of one task is duration between its ready time and the available time of the choosed VEC server. Consequently, the starting time $ST_{i,k,s}$ of task $T_{i,k}$ over server s is:

$$ST_{i,k,s} = \max\left(RT_{i,k}, AT_{i,k,s}\right) \tag{7}$$

Only when the task is ready and the server is available, the task can be deployed on the server. Then we denote $ET_{i,k,s}$ as the execution time that task $T_{i,k}$ over server s, $s \in Des_i$, $ET_{i,k,s}$ is thus:

$$ET_{i,k,s} = \frac{d_{i,k}}{f_s} \tag{8}$$

where $d_{i,k}$ is the data size of the task $T_{i,k}$, and f_s means the computation rate of server s, $s \in Des_i$.

The finishing time $FT_{i,k,s}$ of task $T_{i,k}$ over server s, $s \in Des_i$ are tied to which layer the mission is ultimately deployed to, thus, the finish time can be given as

$$FT_{i,k,s} = \begin{cases} ST_{i,k,s} + ET_{i,k,s}, & if \ s = 1 \\ ST_{i,k,s} + t_{i,k}^{uplinkE} + ET_{i,k,s}, & if \ s \in \{2, ..., S+1\} \\ ST_{i,k,s} + t_{i,k}^{uplinkC} + ET_{i,k,s}, & otherwise \end{cases} \tag{9}$$

When all the small tasks in the application A_i are deployed, obviously, the final completion time of the terminated tasks in the application is the completion time of the entire application. $t_{i,K}$ denotes terminated task of application A_i, and the makespan of application A_i is thus:

$$MakeSpan_i = RFT_{i,K} \tag{10}$$

And the averaged makespan of all M applications over the vehicles is

$$AveSpan = \frac{1}{i} \cdot \sum_{i=1}^{M} MakeSpan_i \tag{11}$$

And One of our optimization metrics is the Maximum completion time for all computational tasks MakeSpan

$$MakeSpan = \max_{1 \le i \le M} \max_{1 \le k \le K} \sum_{s=1}^{S+2} (ET_{i,k,s} \cdot x_{i,k,s}) \tag{12}$$

where $x_{i,k,s}$ represents whether the k^{th} task of the i^{th} application is scheduled on server s, $x_{i,k,s} = \{0, 1\}$ represents the task $t_{i,k}$ is scheduled or not.

2.3 Cost Model

We will present the cost model of the system in this section. We assume that each computation unit, including centralized cloud, RSU and local computer, has its own cost consumption p_s. For the cost reduction purpose, we consider $cost_s$ as the total cost of local severs, MEC servers or centralized cloud, $cost_s(t)$ as the total cost up to time t

$$cost_s = \sum_{i=1}^{M} \sum_{k=1}^{K} ET_{i,k,s} \cdot x_{i,k,s} \cdot p_s \tag{13}$$

where $ET_{i,k,s}$ represents the execution duration of the k^{th} task of the i^{th} application on server s, $x_{i,k,s} = \{0,1\}$ represent the task $t_{i,k}$ is schedule in server s or not.

$AvgCost$ denotes the average cost of each application, it also represents the average cost of solving each user's application needs.

$$AvgCost = \frac{\sum_{s=1}^{S+2} cost_s}{M} \tag{14}$$

where $ET_{i,k,s}$ represents the execution time that the k^{th} task of the i^{th} application applied in server s. One prerequisite before achieving our optimization goal is that the Average user cost is not allowed to exceed a threshold, i.e., there is an upper limit to the cost of our design requirement.

Total cost is an important metric for our optimization goal. So we should keep the operating cost of the rented edge servers as low as possible.

$$Cost = \sum_{s=1}^{S+2} cost_s \tag{15}$$

3 Problem Formulation

In this paper, we set a total of two optimization metrics. We want the scheduling algorithm to make the maximum completion time of all computational tasks as short as possible, while the operating cost of the rented edge servers is low. The problem is formulated as follows:

$$\min f_1 = \max_{1 \le i \le M} \max_{1 \le k \le K} \sum_{s=1}^{S+2} (ST_{i,k,s} + t_{i,k}^{uplink} + ET_{i,k,s}) \cdot x_{i,k,s} \tag{16}$$

$$\min f_2 = \sum_{i=1}^{M} \sum_{k=1}^{K} \sum_{s=1}^{S+2} ET_{i,k,s} \cdot x_{i,k,s} \cdot p_s \tag{17}$$

s.t.

$$\lim_{T \to \infty} \frac{1}{T} \sum_{t=1}^{T} \sum_{s=1}^{S+2} cost_a(t) \le Q_m \tag{18}$$

$$i \in [1, M], k \in [1, K], s \in [1, S+2], x_{i,k,s} = \{0,1\} \tag{19}$$

$$ST_{i,k,s} \ge 0, t_{i,k}^{uplink} \ge 0, ET_{i,k,s} \ge 0, p_s \ge 0 \tag{20}$$

The constraint (18) is the constraint of the long-term cost for the mobile users, it requires the average user cost not to exceed the limit of Q_m. And ST represents start time of task scheduling, ET denotes the time taken by the task to execute on the server, $t_{i,k}^{uplink}$ denotes the different uplink delays, $t_{i,k}^{uplink} = 0$ when the task chooses to be deployed locally. Their corresponding subscripts i, k, s denote scheduling the k^{th} task of application i^{th} to server s. In this case $x_{i,k,s} = 1$, otherwise $x_{i,k,s} = 0$. p_s stands for the cost per unit required to run the server s. Traditional optimization methods are difficult to solve the problem of multi-objective optimization under resource-poor conditions proposed in this paper. Therefore, we propose an optimization algorithm based on DQN and Lyapunov to solve this problem.

4 Proposed Algorithm and Analysis

In this section, we will develop an algorithm using the Lyapunov optimization framework and then evaluate the capabilities of the algorithm in terms of minimizing latency and smoothing out cost consumption.

4.1 Lyapunov Optimization Algorithm

Solving (16) (17) directly is difficult for the conflicting cost constraints for mobile users and the task offloading decisions are also different in each time slot. Thus, we intend to use Lyapunov optimization to build a cost-insufficient queue that guides the data task offloading strategies to follow the cost constraints.

Due to the stochastic and dynamic identity of the data flow task, it seems difficult to find out the generation of the next time slot task, therefore, we should allocate the existing resources in the current time slot to make sure that the system queue can be stabilized for a long period of time. Then, the queue length $q(t)$ at the beginning of time slot t follows the equation as

$$q(t+1) = q(t) + \sum_{s=1}^{S+2} cost_s^{new}(t) - \sum_{s=1}^{S+2} cost_s^{finish}(t) \tag{21}$$

where $q(0) = 0$, and $q(t+1)$ is the queue backlog from the cost constraint in the time slot t, which indicating the deviation of the current energy consumption, $cost_s^{(new)}$ represents new unscheduled servers' cost added at time t and $cost_s^{(finish)}$ indicates the cost of the servers that all tasks scheduled are completed and no tasks are scheduled at time t.

In order to characterize the total queue length of the system, we denote $q(t)$ is the set of current system load states, and defines the Lyapunov equation as

$$L(t) = \frac{1}{2} \cdot q^2(t) \tag{22}$$

Equation (21) represents the level of queue backlog in the system. Within each time slot, the scheduler gives a decision such that the Lyapunov equation always remains under a bound, then the system is always stable. Therefore, we make the Lyapunov equation always remain in a region with low congestion by reducing the Lyapunov-drift.

$$\Delta(q(t)) = L(t+1) - L(t) \tag{23}$$

The Lyapunov drift $\Delta(q(t))$ represents the degree of change in the total queue length in the Lyapunov equation after one time slot, The extent of this change depends on the current choice of strategy, and an effective strategy can result in a small change in $\Delta(q(t))$. Therefore, adding queue stability to makespan and cost consumption optimization, that is adding Lyapunov drift to the reward function of DQN, we will explain it in the next section.

4.2 DQN-Based Multi-objective and Multi-workflow Offloading

We propose a Deep Reinforcement Learning-based Multi-Objective Multi-Workflow Scheduling (DRLMOMWS) algorithm. Under edge computing, this scheduling algorithm program will be configured to the edge server, when there is a computing task deployed to the edge server, the program will take the working state of the edge server cluster and currently pending tasks, make their information as inputs to the model, and after the algorithmic model, the corresponding scheduling scheme will be obtained. We use DQN as a reinforcement learning algorithm and employed multiple intelligences.

The multi-workflow offloading problem in this paper can be modeled as a Markov decision model. At time slot t, the agent first gets an initial state $s_t \in S$ from the environment, after getting the initial state, the agent will get the next action a_t according to the given policy π. The agent then feeds this action a_t back to the environment, and the environment body executes the action which causing a change in state from s_t to s_{t+1}. At the same time, the agent receives r_t as a reward from the ambient body as feedback.

Specifically, we define state space as $S = \{L_i, \alpha\}$, where $L_i = \{locationx_i(t), locationy_i(t)\}$ denotes the 2D coordinate position of the vehicle where the app that needs to make a decision is currently located, $\alpha = \{\alpha_1(t), \alpha_2(t), ..., \alpha_{S+2}(t)\}$ denotes indicates the current readytime of the local server, MEC server, and cloud server. And the action space is $A = \{\rho_1, \rho_2 ... \rho_n\}$, where ρ_i is the server to which the decision-making app can be deployed, and the vehicle in which the app is located needs to be within the range of the server. Next, the intelligent body selects an action from the discrete set of actions $a_t \in A$ according to its strategy, and then receives a reward given by the environment. We argue that the benefits of the offloading decision problem are related to the latency of the applications in the system as well as the average user cost.

$$R_{makespan} = [\frac{ET_{i,k,s}(a) - (makespan_{new} - makespan_{old})}{ET_{i,k,s}(a)}]^3 \tag{24}$$

$$R_{cost} = [\frac{cost_{worst} - ET_{i,k,s}(a) \cdot p_s}{cost_{worst} - cost_{best}}]^3 + \Delta(q(t)) \tag{25}$$

Than the intelligence tries to obtain the maximized expected discounted return value by $R_t = \sum_{\tau=t}^{\infty} \gamma^{\tau-t} \cdot r_\tau$. Where $\gamma \in [0, 1]$ is the discount factor to weigh the importance of future and current returns.

For an intelligent that makes decisions according to a stochastic strategy π, the state-action pair (s, a) and the value definition of the state are respectively

$$Q^\pi(s,a) = E[R_t | s_t = s, a_t = a, \pi] \tag{26}$$

$$V^\pi(s) = E_{a-\pi(s)}[Q^\pi(s,a)] \tag{27}$$

With a defined strategy $a = \arg\max_{a' \in A} Q^*(s, a')$, The better Q function can be defined as $Q^*(s,a) = \max_\pi Q^\pi(s,a)$, it can satisfy $V^*(S) = \max_a Q^*(s,a)$. Intuitively, the value function V measures the importance of the ambient body being in a particular state s, while the Q function measures the importance of the change in reward value that would result from choosing each possible action while in that state.

The DQN algorithm uses a convolutional neural network to predict Q-values, thus solving the problem of Q-functions that are difficult to compute and store in large-scale problems. In addition to this, DQN introduces an empirical playback pool to store the used data, thus breaking the correlation between the data. During the learning process, the intelligence collects a set of empirical datasets through multiple iterations $D_t = \{e_1, e_2, ...e_t\}$, where $e_t = (s_t, a_t, r_t, s_{t+1})$. When we go to train the Q-network, we randomly use a portion of the sample data from the dataset D_t for training, rather than using the standard time-difference method to obtain the current experience.

The ultimate goal of the algorithm is to learn and select the optimal policy that achieves joint optimization of delay and cost for this model by minimizing the loss function.

$$y_i^{DQN} = R_t + \gamma \cdot \max_{a'} Q(s', a'; \theta^-) \tag{28}$$

$$L_i(\theta_i) = E_{(s,a,r,s')-\mu(D)}[(y_i^{DQN} - Q(s,a;\theta_i))^2] \tag{29}$$

where θ indicates the weights of target network.

For reinforcement learning algorithms for single intelligences, since there are only unique intelligences in the environment, the decisions made by the intelligences I are able to be executed and reacted to the environment. However, for multi-intelligents environments, there is a possibility that the decisions of multiple intelligences may conflict, therefore, we take measures to deal with situations of conflicting decision-making: maximize the rewards of all intelligences

$$\max_{\pi_s \in \delta(A(s)} \sum_{i \in I} \sum_{a \in A(s)} \pi_s(a) \cdot Q_i(s,a) \tag{30}$$

Since DQN is used as the reinforcement learning algorithm, the training of the model will take a lot of time since it involves a deep neural network, so we use two scheduling programs with the same initial weights of the neural network at the same time, one of them is responsible for training and refining the network model, and the other is only responsible for making decisions. Since there is no need to train the model, the program that makes the decision does not involve the back propagation process, and the computation time is fast enough to satisfy real-time.

Based on the above description, we can summarize in Algorithm 1 the basic flow of our online algorithm for giving Lyapunov-based DQNs.

Algorithm 1 Deep Q Learning algorithm

Initialize function Q with random weights
Initialize memory D with capacity C
for $episode = 1, 2, 3, ...$ **do**
  Initialize sequence $s_l = \{x_l\}$
  Initialize preprocessed sequenced $\phi = \phi(s_i)$
  for $t = 1$ to T **do**
    With probabilty ϵ select random action $a_t \in A$
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta$
    Execute action a_t in environment and observe reward r_t and state s_t
    Store transition$(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    Sample random minibatch of tansitions $(\phi_t, a_t, r_t, \phi_{t+1})$ from D
    Set $y_j = r_j + \gamma \cdot \max_a Q(\phi_{j+1}, a'; \theta)$
    Perform a gradient descent step on $(y_j - Q(\phi_t, a_j l\theta)^2)$
  end for
end for

5 Simulation and Discussions

In this section, we will evaluate the performance and nausea due to the Lyapunov optimization-based DQN algorithm introduced in this paper in terms of both the average latency of the in-vehicle application execution and the average cost of the user application.

5.1 Simulation Setup

For the simulation experiment, we assume that 5 RSUs are contained in MEC system and each RSU contains 5 VEC servers. One centrailized cloud is contained in environment and this cloud contains two cloud servers. The processing speed of local vehicles fall between 6×10^6 to 1×10^7 kB/s and the execution rate of VEC servers are 2.4×10^7 to 4.7×10^7 kB/s, the execution rate of cloud server is about 2 times as much as edge servers. The channel bandwidth among vehicles and RSUs is 3.7 MHz, the channel gain G_n among vehicles and RSUs is 20^{-4}. The transmission power of each vehicle is 120 mW. There are 30 mobile vehicles randomly distributed within the rsu coverage and move with random trajectories and each vehicle is distributed from 1 to 4 m/s. The datasize of tasks are between 2×10^7 to 5×10^7 KB. The noise power N_0 is 10^{-10} mW. Local servers are inexpensive, while cloud servers cost is about 50 times as much as edge servers. The DAG for some of the applications is shown in Fig. 2.

In the simulation, we compare the Lyapunov optimization scheme proposed in this chapter with the original scheme without Lyapunov optimization as well as with three traditional schemes for two aspects of application latency and average user cost under changing unique conditions.

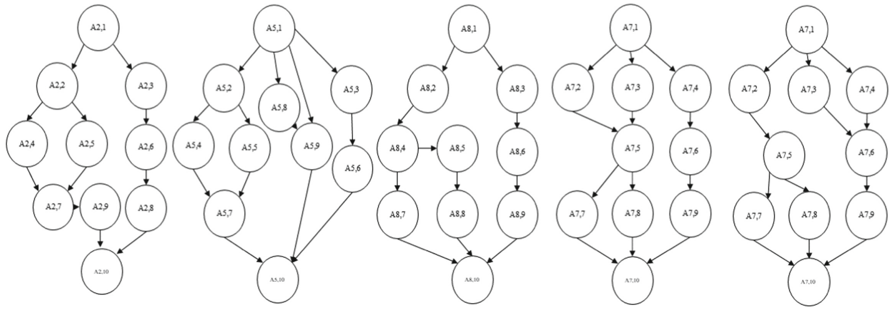

Fig. 2. The workflows of 5 vehicles in the experiment.

- Random Selection (RS): Computing the TPQ, the TPQ is the task list that sorted by the last start time of all tasks, select tasks from TPQ, and then deploy tasks $A_{i,k}$ to a random VEC server $r \in Des_i$.
- MAMTS: Computing the TPQ, Select tasks from TPQ, and the TPQ is the task list that sorted by the last start time of all tasks, then system will deploy task $A_{i,k}$ to the server $r \in Des_i$ choosed by MAMTS algorithm [8].
- Greedy Selection (GS): Computing the TPQ, the TPQ is the task list that sorted by the last start time of all tasks, select tasks from TPQ, and then greedily deploy the task to VEC server $r \in Des_i$, because it could provide the task for the least execution time [17].

5.2 Performance Comparison

Figure 3 shows the cost consumption per unit time slot of our proposed DQN-based multi-objective multi-intelligence task deployment algorithm before and after using Lyapunov optimization. The results show that Lyapunov-opt further reduces the task latency from the original algorithm while strictly adhering to the long-term energy constraint. Moreover, after Lyapunov-opt, the cost of task execution becomes smoother and more uniformly distributed in a macroscopic sense.

In Fig. 3, we can also see that in the first about one thousand time segments, the energy consumption with Lyapunov optimization will be slightly higher than without Lyapunov optimization, this phenomenon occurs because the Liapunov optimization is to average the overall cost consumption to the various time periods, and try to maintain the balance of the cost queue and slow down the peak of the peak period of the peak period of time, Therefore, in the early stage, the cost of the optimized model is slightly higher than the non-optimized model.

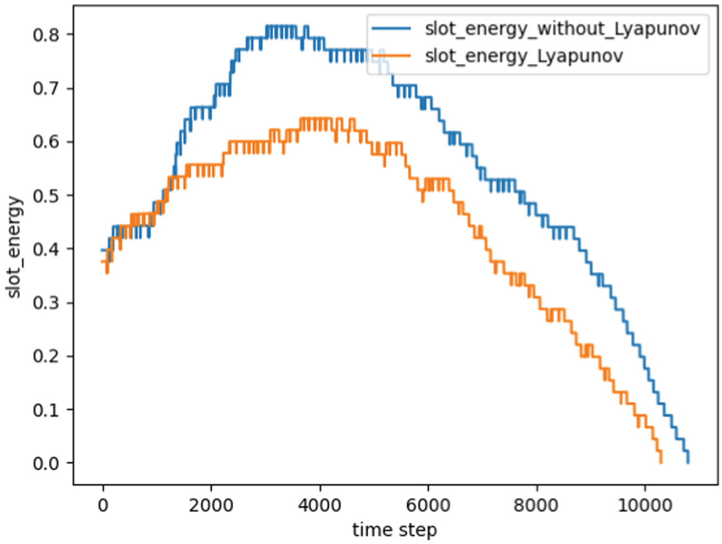

Fig. 3. Comparisons of Slot cost with Lyapunov or not.

Figure 4 shows the comparison of the average task latency and average cost consumption of the different approaches at different processing speeds. As can be seen from the figure, although the average latency of our algorithm is almost three times that of the other algorithms, the gap between the latencies gradually decreases as the processing speed of the server increases. On the other hand, the total cost consumption of our method can be reduced by 60% or more compared to other methods. This is because conventional methods do not consider energy consumption, which consumes a significant amount of cost while achieving the minimum average interval latency. In contrast, our scheme employs Lyapunov optimization, which greatly satisfies the cost constraints despite a slight sacrifice in delay performance.

Analyzing Fig. 4 in more detail, we can see from the left sub-figure that since the traditional method does not consider the cost, the change in the average cost consumption will be disorderly, while our method maintains a low cost consumption in the case that the server speed becomes gradually faster; in the right sub-figure we can see that the average task latency of our method decreases gradually with the increase of the server processing speed, the reason that there is a rebound in the middle segment is that our method is considering the joint optimization of latency and cost. The reason that there is a rebound in the middle segment is that our method is a joint optimization considering delay and cost, and combined with the left figure we can see that the cost is decreasing during this period, then the corresponding delay will have a rise. But we can

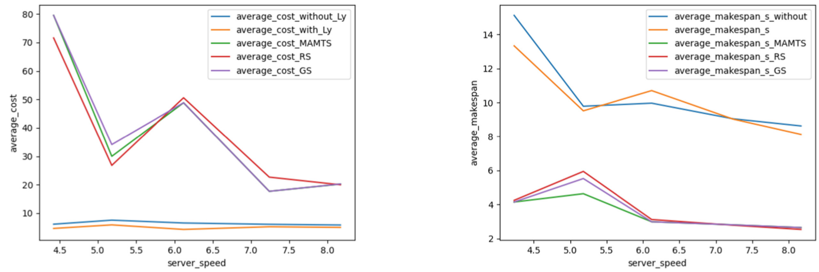

Fig. 4. Comparison of average cost of applications and average makespan delay with different processing speed

also observe that the method optimized by Lyapunov is smoother in terms of both cost and delay changes.

Figure 5 shows the performance comparison of five different schemes when the number of in-vehicle applications varies. As can be seen from the figure, the average effective delay and cost of all these schemes increase when the number of applications increases. Compared with the other strategies, our proposed strategy is two to three times higher than the traditional methods in terms of average delay, but the gap between the average delay decreases as the number of edge users increases; also, our proposed strategy increases the average cost very slowly and the average cost consumption is much smaller than the other algorithms.

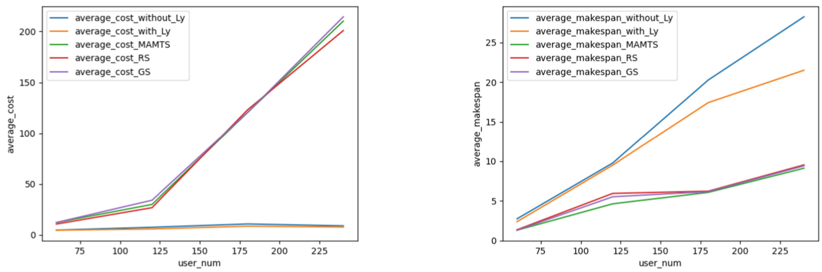

Fig. 5. Comparison of average cost of applications and average makespan delay with different number of applications

It can also be seen from Fig. 5 that the increase in edge users implies a high-pressure environment at the edge. Facing high-pressure environments, it can be seen from the left subfigure that our approach consistently keeps the average cost low, although there is a relatively small increase in the average cost. Meanwhile, it can be seen from the right subfigure that the optimized approach using Lyapunov is able to further reduce the average latency and further optimize the task offloading strategy in more high-pressure environments, but in contrast to the traditional scheme, the reason for our higher average latency may be that our proposed scheme focuses on task offloading, while the comparison scheme considers both task scheduling and task offloading.

6 Conclusions

In this paper, we propose a DQN algorithm based on Lyapunov optimization that acts mainly on vehicle-mounted applications with interdependent tasks. These tasks can be deployed to different layers in a hybrid three-tier edge computing environment. We assume that each application is represented by a DAG. Our optimization problem is defined as minimizing the average cost of all mobile users and the average latency of all applications subject to long-term cost constraints. We then propose an algorithm based on Lyapunov optimization and DQN with multi-objective multi-intelligentsia for generating task offloading solutions. Simulation results clearly show that our proposed approach outperforms other algorithms in terms of both application latency and combined user cost tradeoffs. For the method proposed in this paper, there are some limitations, for example, this paper only considers task offloading without considering task scheduling, which can lead to some time-sensitive task timeouts. For future improvement, the scheduling of tasks can be considered along with task offloading and the deadline of tasks can be considered as one of the rewards of DQN.

Acknowledgement. This work was supported in part by the Key Research and Development Project of Henan Province under Grant No. 231111211900, in part by the Henan Province Science and Technology Project under Grant No. 232102210024.

References

1. Kumar, S., Bhagat, L., Jin, J.: Multi-neural network based tiled 360° video caching with mobile edge computing. J. Netw. Comput. Appl. **201**, 103342 (2022). https://doi.org/10.1016/j.jnca.2022.103342
2. Hu, X., Wang, J., Zhong, C.: Statistical CSI based design for intelligent reflecting surface assisted MISO systems. Sci. China Inf. Sci. **63**(12) (2020). https://doi.org/10.1007/s11432-020-3033-3
3. Lai, X., Fan, L., Lei, X., Deng, Y., Karagiannidis, G.K., Nallanathan, A.: Secure mobile edge computing networks in the presence of multiple eavesdroppers. IEEE Trans. Commun. **70**(1), 500–513 (2022). https://doi.org/10.1109/TCOMM.2021.3119075
4. Na, Z., et al.: UAV-based wide-area internet of things: an integrated deployment architecture. IEEE Network **35**(5), 122–128 (2021). https://doi.org/10.1109/MNET.001.2100128
5. Quan, W., Cheng, N., Qin, M., Zhang, H., Chan, H.A., Shen, X.: Adaptive transmission control for software defined vehicular networks. IEEE Wirel. Commun. Lett. **8**(3), 653–656 (2019). https://doi.org/10.1109/LWC.2018.2879514
6. Lee, E., Lee, E.K., Gerla, M., Oh, S.Y.: Vehicular cloud networking: architecture and design principles. IEEE Commun. Mag. **52**(2), 148–155 (2014). https://doi.org/10.1109/MCOM.2014.6736756

7. Li, T., Gao, C., Jiang, L., Pedrycz, W., Shen, J.: Publicly verifiable privacy-preserving aggregation and its application in IoT. J. Netw. Comput. Appl. **126**, 39–44 (2019). https://doi.org/10.1016/j.jnca.2018.09.018
8. Liu, Y., et al.: Dependency-aware task scheduling in vehicular edge computing. IEEE Internet Things J. **7**(6), 4961–4971 (2020). https://doi.org/10.1109/JIOT.2020.2972041
9. Lin, W., et al.: A hardware-aware CPU power measurement based on the power-exponent function model for cloud servers. Inf. Sci. **547**, 1045–1065 (2021). https://doi.org/10.1016/j.ins.2020.09.033
10. Hu, L., Yan, H., Li, L., Pan, Z., Liu, X., Zhang, Z.: MHAT: an efficient model-heterogenous aggregation training scheme for federated learning. Inf. Sci. **560**, 493–503 (2021). https://doi.org/10.1016/j.ins.2021.01.046
11. Mao, Y., You, C., Zhang, J., Huang, K., Letaief, K.B.: A survey on mobile edge computing: the communication perspective. IEEE Commun. Surv. Tutorials **19**(4), 2322–2358 (2017). https://doi.org/10.1109/COMST.2017.2745201
12. Hou, X., Li, Y., Chen, M., Wu, D., Jin, D., Chen, S.: Vehicular fog computing: a viewpoint of vehicles as the infrastructures. IEEE Trans. Veh. Technol. **65**(6), 3860–3873 (2016). https://doi.org/10.1109/TVT.2016.2532863
13. Zhou, Z., Liu, P., Chang, Z., Xu, C., Zhang, Y.: Energy-efficient workload offloading and power control in vehicular edge computing, pp. 191–196 (2018). https://doi.org/10.1109/WCNCW.2018.8368975
14. He, Q., et al.: A game-theoretical approach for mitigating edge DDoS attack. IEEE Trans. Dependable Secur. Comput. **19**(4), 2333–2348 (2022). https://doi.org/10.1109/TDSC.2021.3055559
15. Caiazza, C., Giordano, S., Luconi, V., Vecchio, A.: Edge computing vs centralized cloud: impact of communication latency on the energy consumption of LTE terminal nodes. Comput. Commun. **194**, 213–225 (2022). https://doi.org/10.1016/j.comcom.2022.07.026
16. Zhou, Y., et al.: A novel approach to applications deployment with multiple interdenpendent tasks in a hybrid three-layer vehicular computing environment, pp. 251–256 (2021). https://doi.org/10.1109/SMC52423.2021.9659035
17. Zhao, Z., Liu, S., Zhou, M., Guo, X., Xue, J.: Iterated greedy algorithm for solving a new single machine scheduling problem, pp. 430–435 (2019). https://doi.org/10.1109/ICNSC.2019.8743328
18. Shahidani, F., Ghasemi, A., Haghighat, A.: Task scheduling in edge-fog-cloud architecture: a multi-objective load balancing approach using reinforcement learning algorithm, pp. 1337–1359 (2023). https://doi.org/10.1007/s00607-022-01147-5
19. Zhang, K., Mao, Y., Leng, S., He, Y., Zhang, Y.: Mobile-edge computing for vehicular networks: a promising network paradigm with predictive off-loading. IEEE Veh. Technol. Mag. **12**(2), 36–44 (2017). https://doi.org/10.1109/MVT.2017.2668838
20. Wu, Y., Gao, C.: Intelligent task offloading for vehicular edge computing with imperfect CSI: a deep reinforcement approach **55**, 9 (2022). https://doi.org/10.1016/j.phycom.2022.101867
21. Zhang, L., Xia, J., Gao, C., Zhu, F., Fan, C., Ou, J.: DQN-based mobile edge computing for smart internet of vehicle, 45 (2022). https://doi.org/10.1186/s13634-022-00876-1