electronics

MDPI

Article

Deep Reinforcement Learning-Based Task Offloading and Load Balancing for Vehicular Edge Computing

Zhoupeng Wu, Zongpu Jia, Xiaoyan Pang and Shan Zhao *

School of Computer Science and Technology, Henan Polytechnic University, Jiaozuo 454000, China; 212109020047@home.hpu.edu.cn (Z.W.)
* Correspondence: 212109020045@home.hpu.edu.cn

Abstract: Vehicular edge computing (VEC) effectively reduces the computational burden on vehicles by offloading tasks from resource-constrained vehicles to edge nodes. However, non-uniformly distributed vehicles offloading a large number of tasks cause load imbalance problems among edge nodes, resulting in performance degradation. In this paper, we propose a deep reinforcement learning-based decision scheme for task offloading and load balancing with the optimization objective of minimizing the system cost considering the split offloading of tasks and the load dynamics of edge nodes. First, we model the mutual interaction between mobile vehicles and Mobile Edge Computing (MEC) servers using a Markov decision process. Second, the optimal task-offloading and resource allocation decision is obtained by utilizing the twin delayed deep deterministic policy gradient algorithm (TD3), and server load balancing is achieved through edge collaboration using a server selection algorithm based on the technique for order preference by similarity to the ideal solution (TOPSIS). Finally, we have conducted extensive simulation experiments and compared the results with several other baseline schemes. The proposed scheme can more effectively reduce the system cost and increase the system resource utilization.

Keywords: task offloading; resource allocation; edge collaboration; load balancing; deep reinforcement learning

Citation: Wu, Z.; Jia, Z.; Pang, X.; Zhao, S. Deep Reinforcement Learning-Based Task Offloading and Load Balancing for Vehicular Edge Computing. *Electronics* **2024**, *13*, 1511. https://doi.org/10.3390/electronics13081511

Academic Editor: Carlo Mastroianni

Received: 22 March 2024
Revised: 9 April 2024
Accepted: 12 April 2024
Published: 16 April 2024

1. Introduction

The Internet of Vehicles (IoV) is a convergence of vehicular ad hoc networks (VANETs) and the Internet of Things (IoTs) that will enhance transportation efficiency and vehicle safety [1]. Vehicles fitted with a wide range of advanced equipment generate a variety of computationally intensive applications, such as collision warning and driverless driving, which have stringent latency requirements. However, local processing in the vehicle can be affected by insufficient resources, resulting in performance degradation [2]. How to ensure low-latency requirements for vehicles with limited resources is therefore a key challenge for IoV.

VEC has a promising application in intelligent vehicle applications as an extended application of edge computing in IoV [3]. VEC improves vehicle computational performance by deploying MEC servers with high computational power in road side units (RSUs) to provide computational services. Specifically, computation-intensive application tasks can be offloaded from the vehicle over the wireless network and computed by MEC servers with more resources [4]. MEC servers have greater computing power than vehicles, enabling the faster processing of application tasks and shorter application response times. For this reason, a growing number of researchers have started to focus on task offloading in VEC [5].

Deep reinforcement learning (DRL) incorporates the perceptual and decision-making capabilities of deep learning and is a subfield of artificial intelligence. DRL is capable of solving problems that traditional reinforcement learning cannot in high-dimensional state and action spaces, and edge nodes utilizing DRL's cognitive and analytical capabilities

can interact directly with dynamic vehicular networks to reduce backhaul bandwidth and cached content delivery latency and enhance computational efficiency [6]. Up to now, the effectiveness of DRL-based task-offloading optimization in VEC scenarios has been validated in several studies [7–9].

However, the following issues still need to be explored. First, due to the high communication and storage costs of MEC servers, particularly when a high number of vehicles gather within their communication coverage, unloading vehicle tasks to MEC servers may have the opposite effect [10]. In addition, the mobility of vehicles in a VEC network and the disparity in regional infrastructure deployments can lead to load imbalances between MEC servers. Appropriate offloading decisions for vehicle-generated tasks can effectively balance the load between edge servers, thus effectively improving the resource utilization [11].

A Markov decision process (MDP) is a mathematical model for describing a decision-maker's choice of an optimal policy in uncertain environments and is well suited for the complex and varied offloading environments of Telematics tasks. TD3 is an efficient and stable deterministic policy reinforcement learning algorithm and is easy to implement and suitable for high-dimensional continuous action spaces, and thus, it can be used to solve MDP problems. TOPSIS is a common and effective method in multi-objective decision analysis that accurately responds to gaps between evaluation options. On the basis of the above considerations, we design a task-offloading and load-balancing decision scheme based on DRL called TOLB. Our main contributions can be summarized as follows:

1. For the multi-vehicle and multi-server scenario in vehicular networking, a dynamic computational offloading problem is constructed as an MDP. The decision problem is then transformed into an optimization problem to minimize the system cost while guaranteeing the load balancing of the MEC servers.
2. In this study, we designed a novel jointly optimized task-offloading and load-balancing scheme, TOLB. TOLB designs a TD3-based task-offloading and resource allocation algorithm to obtain the best decision for task offloading, and it uses a TOPSIS-based server selection algorithm to select low-load MEC servers to which high-load MEC servers can migrate the tasks and achieve load balancing through edge collaboration.
3. We have carried out some comparative experiments to evaluate how the proposed scheme performs. Compared to the benchmark scheme, the findings reveal that the proposed scheme can better optimize the processing power of the MEC server and lower the system cost.

The remainder of this paper is structured as follows: Section 2 analyzes related work, Section 3 discusses the system network architecture and optimization issues, Section 4 details the specifics of the TOLB scheme, Section 5 evaluates the experimental performance, and Section 6 concludes the paper.

2. Related Work

The transfer and computation of computationally intensive tasks between the vehicle and the MEC server not only consume energy but also generate time delays [12,13]. Optimization schemes for the delay and energy consumption incurred by tasks during execution have become a crucial research direction in task-offloading studies [14]. Numerous task-offloading methods have been proposed for offloading studies of latency-sensitive and high-complexity tasks in VEC.

Ning et al. [15] considered edge collaboration in an intersection scenario by using Lyapunov optimization to minimize the entire task computation latency of vehicles under the RSU's persistent energy consumption. Zeng et al. [16] analyzed the dynamic between an MEC server and a vehicle according to the Stackelberg game, proving the existence of an optimal offloading policy between the two entities; they also designed a rapid search algorithm using a genetic algorithm to find the optimal MEC server pricing strategy. Liu et al. [17] presented a distributed algorithm using the Stackelberg game with multiple leaders and followers to improve the utility of mobile vehicles and MEC servers under

deadline constraints. Lin et al. [18] explored a heterogeneous VEC network, leveraging task popularity among vehicles for dynamic clustering, and proposed an online vehicle task-offloading solution based on bandit context clustering. Luo et al. [19] designed a self-learning-based distributed computing offloading algorithm to solve the distributed offloading decision game and minimize the computational cost of performing the task. However, traditional optimization algorithms require several iterations to reach a relatively optimal solution, which can cause unacceptable delays in realistic application scenarios [20].

The application of DRL to address the challenges of task offloading has gained attention in recent years. Wang et al. [21] aimed to minimize the delay of each task and developed a deep learning-based resource allocation method to adapt to changing MEC environments and handle high-dimensional inputs. Pang et al. [22] considered multitasking offloading and designed a time-optimized Dueling Double Deep Q Network (D3QN)-based multitasking offloading algorithm to decrease the latency and energy consumption of the system. Zheng et al. [23] developed an asynchronous dominant participant–critic-based decision-making algorithm in a digital twin network framework aiming at fast convergence and reduced system cost. Shi et al. [10] proposed an offloading algorithm based on a dual-depth Q network to solve the problem of offloading subtasks between vehicles and subtasks between vehicles and edge nodes with the aim of reducing the subtask packet loss rate, the average task delay, and the total energy consumption. Liu et al. [24] accounted for dependencies between subtasks by modeling these dependencies with a directed acyclic graph and proposed a task-offloading algorithm based on a deep deterministic policy gradient (DDPG). Peng et al. [14] allowed different vehicles to share the results of similar tasks and designed a shared offloading strategy based on DRL. Shi et al. [25] developed a smart-contract-based vehicular task allocation scheme within a lightweight blockchain-based VEC framework and utilized DRL to determine the resources required to perform a task, and dynamic pricing was used to incentivize vehicles to make idle resources available. Long et al. [26] designed a power allocation scheme based on decentralized DRL in a non-orthogonal multiple-access communication scenario with multiple inputs and outputs. These DRL-based schemes focus primarily on offloading decisions.

To model system utility, Dai et al. [27] formulated the optimization problem as a mixed-integer nonlinear programming problem and decoupled it into two subproblems: load-balancing and offloading decisions. Fan et al. [2] used an exact potential game model to simulate the task-offloading contest process among RSUs by decomposing the optimization problem into partial task-offloading and channel allocation subproblems and a server-load-balancing subproblem. Lu et al. [28] developed a multi-RSU workload-balancing scheme that adjusts to variable task popularity in dynamic environments to avoid wasting resources by offloading duplicate tasks. Gao et al. [29] constructed task-offloading, task-scheduling, and Central Processing Unit (CPU) frequency allocation problems as a hybrid nonlinear optimization problem and adopted an iterative optimization algorithm based on a Deep Q Network (DQN) and gradient descent to obtain the optimal decision. Marios et al. [30] designed a two-stage reinforcement learning-based computational offloading scheme, where the first stage designed a stochastic learning automata-based task-offloading decision, and the second stage designed a DQN-based cooperative offloading mechanism for edge sites to achieve load balancing. Wu et al. [31] proposed a bionic algorithm based on Invasive Tumor Growth optimization, which achieves multiple goals, such as load balancing and a reduction in energy consumption through collaboration between edge servers.

The existing research mainly focuses on the task-offloading problem in IoV, with relatively little research on the load-balancing problem of the system, while the task-offloading research on joint load balancing mostly considers the complete offloading of tasks, which reduces the utilization of system resources. Therefore, this study proposes a DRL-based task-offloading and load-balancing scheme. It splits the task into two parts according to a certain ratio, executes one on the vehicle server and the other on the MEC server, and allocates the computational power for executing the task to the vehicle and MEC servers. In addition, it considers the load state of the edge servers and migrates the tasks from

high-load MEC servers to low-load MEC servers for execution, which minimizes the system cost, effectively improves the system resource utilization, and achieves load balancing.

3. System Model

This section first outlines the offloading framework of the VEC system. Subsequently, a computational model is presented, encompassing local computation, task transfer, edge computation, and result return. This model aims to calculate task processing latency and energy consumption. The final section outlines the optimization objectives and constraints of this study.

3.1. System Framework

The architecture of the VEC system is illustrated in Figure 1. The system is designed around a unidirectional straight road, with RSUs positioned sequentially along the roadside. RSUs are equally spaced along the road, and the wireless communication coverage of the RSU is represented by L. Each RSU houses an MEC server, where the ensemble of MEC servers is indicated by $R = \{1, 2, \ldots, M\}$. The resource margin of the MEC server m is denoted by F_m. MEC servers with ample resources are identified as low-load servers, whereas those with limited resources are classified as high-load servers.

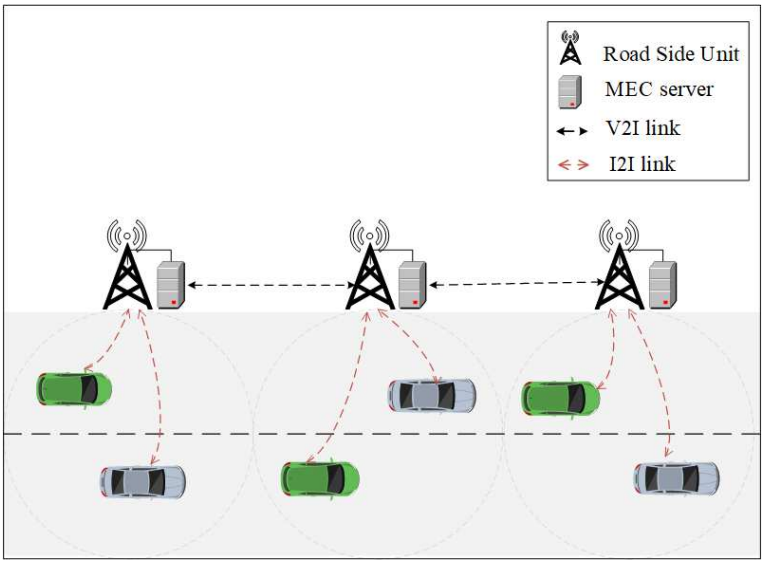

Figure 1. Vehicular edge computing network model.

The system contains N vehicles traveling at a constant speed, with the vehicle set represented by $V = \{1, 2, \ldots, N\}$. We segment the travel time of vehicles within the current communication coverage of the road into individual time slots, denoted by $T = \{0, 1, \ldots, T-1\}$. In each time slot, a vehicle generates a task for processing. The task generated by vehicle $n (n \in V)$ is indicated by $T_n = \{d_n, c_n, t_n^{\max}\}$, where d_n denotes the total data volume of the task in bits, c_n indicates the computational density of the task in CPU cycles per bit, and $t_n^{\max}$ is the upper limit of the delay that the task can tolerate.

In this paper, it is assumed that each task can be split and that the vehicle can offload part of the task to the MEC server in any proportion. λ_{nm} is the offloading ratio of task T_n. The amount of task data processed by MEC server m is $\lambda_{nm} d_n$, and the amount of data processed by vehicle n is $(1 - \lambda_{nm}) d_n$.

3.2. Communication Model

3.2.1. V2I Communication

In the system network model studied in this research, the unloading vehicle needs to transmit the task to the MEC server for execution through the wireless channel, which

must take into account the data transfer rate of the task in the channel. In addition, there exists an edge collaboration model for the MEC servers in the system proposed in this section, where the high-load servers can migrate the tasks to the low-load servers for task processing. Therefore, V2I communication and I2I communication must be considered for this system.

This study assumes that the wireless network connection state of the vehicle remains static during data upload. The data transmission rate between vehicle n and the MEC server is given by

$$r_{n,m} = B_{n,m} \log_2 \left(1 + \frac{\rho_n h_{n,m}}{\sum_{i' \in N'} \rho_{i'} h_{i',m} + N_0} \right) \tag{1}$$

where $B_{n,m}$ is the uplink channel bandwidth between vehicle n and MEC server m, ρ_n is the transmission power of vehicle n, $h_{n,m}$ is the channel gain, N_0 denotes the Gaussian white noise power, and $\sum_{i' \in N'} \rho_{i'} h_{i',m}$ represents the radio interference emitted by other vehicles within the communication range of MEC server m.

3.2.2. I2I Communication

When the load of MEC server m is too high, MEC server m must migrate the tasks to a low-load MEC server m'. Since the data transfer between MEC servers is performed via I2I, the data transfer rate is extremely large compared to V2I; thus, this study neglects the latency and energy consumption during task migration [32].

3.3. Computing Model

The total latency of the execution of the processing task consists of the transmission latency and the computation latency, the total energy consumption of the processing task consists of the transmission energy and the computation energy, and the computing model for executing the task is varied for different locations. According to the different execution modes of tasks, the computation model in the research scenario of this paper can be obtained.

3.3.1. Local Computing Model

For the locally computed section of the task, $f_n^l \left(0 < f_n^l \leq F_n^l \right)$ denotes the computing power allocated by the system to tasks executed locally, where F_n^l is the highest computing power of vehicle n. The local delay of the processing task is given by

$$t_n^l = \frac{c_n (1 - \lambda_{nm}) d_n}{f_n^l} \tag{2}$$

The local energy consumption of vehicle n's running task is expressed by

$$E_n^l = k_l c_n (1 - \lambda_{nm}) d_n \left(f_n^l \right)^2 \tag{3}$$

where c_n denotes the computational density of the task, $(1 - \lambda_{nm}) d_n$ indicates the amount of task data processed locally, and k_l is the energy dissipation factor for vehicle n [33].

3.3.2. Edge Computing Model

When vehicle n offloads tasks to MEC server m, the amount of transmitted task data is $\lambda_{nm} d_n$. The task transmission delay is given by

$$t_n^{trans} = \frac{\lambda_{nm} d_n}{r_{n,m}} \tag{4}$$

where $r_{n,m}$ is the rate at which vehicle n communicates with MEC server m.

The energy consumption during task transfer is given by

$$E_n^{trans} = \rho_n t_n^{trans} \tag{5}$$

where ρ_n is the transmission power of vehicle n.

$f_n^{MEC}\left(0 < f_n^{MEC} \leq F_m^{MEC}\right)$ denotes the computational capacity (in CPU cycles/second) allocated by the system for tasks executed on MEC server m, where F_m^{MEC} denotes the maximum computational capacity of MEC server m. The computing latency of the offloaded task is expressed by

$$t_n^{MEC} = \frac{c_n \lambda_{nm} d_n}{f_n^{MEC}} \tag{6}$$

The energy consumption by MEC server m to execute tasks is defined as follows:

$$E_n^{MEC} = k_e c_n \lambda_{nm} d_n \left(f_n^{MEC}\right)^2 \tag{7}$$

where c_n equals the computational density of the task, $\lambda_{nm} d_n$ denotes the amount of offloaded data for the task, and k_e is the energy dissipation factor of MEC server m [33].

As the calculation results are very small compared to the amount of input data, this paper ignores the latency and energy consumption of returning the computational results.

3.4. The Formulation of the Problem

As previously mentioned, for task T_n, the computing latency includes both the local and offloaded parts. The total computing latency of task T_n can be described as

$$t_n = \max\left\{t_n^l, \left(t_n^{trans} + t_n^{MEC}\right)\right\} \tag{8}$$

The overall energy consumption can be calculated as

$$E_n = E_n^l + E_n^{trans} + E_n^{MEC} \tag{9}$$

Hence, the system cost of task T_n's execution can be given by

$$U(t) = u^t t_n + u^e E_n \tag{10}$$

where $u^t \in [0,1]$ and $u^e \in [0,1]$ are the weight factors for the delay and energy consumption to indicate how much the user values delay and energy, respectively, satisfying the constraint $u^t + u^e = 1$. Depending on the vehicle user's requirements, the values of these weighting factors can be adjusted for decision-making.

The aim of optimizing this system is to keep system costs to an absolute minimum while ensuring the load balancing of the MEC servers. The optimization problem is formulated as follows:

$$\min\left(\frac{1}{N}\sum_{n=1}^{N} U_n\right) \tag{11}$$

$$s.t.\ t_n \leq t_n^{\max}, \forall n \in V \tag{12}$$

$$0 \leq \lambda_{nm} \leq 1, \forall n \in V, \forall m \in R \tag{13}$$

$$0 < f_n^{MEC} \leq F_m^{MEC},\ 0 < f_n^l \leq F_n^l \tag{14}$$

$$\sum_{n=1}^{N} \lambda_{nm} d_n \leq \sum_{m=1}^{M} F_m \tag{15}$$

Constraint (12) ensures that the task completion latency is never greater than the maximum latency constraint tolerated by the task. Constraint (13) sets a limitation on the offloading decision variable, ensuring that the rate of task offloading is between 0 and 1.

Constraint (14) ensures that the computational power allocated by the system to the tasks is positive. Finally, constraint (15) ensures that the volume of task data transferred from the system to the MEC servers does not exceed the maximum load capacity of the system.

4. A Task-Offloading and Load-Balancing Decision Scheme Based on DRL

Owing to the high agility of vehicles, real-time offloading decisions need to be made based on the current network environment [34]. The dynamically changing environment of Telematics may lead to the uncertainty of system inputs and conditions, and the task-offloading and resource allocation algorithms should take the time-varying environment state into full consideration in order to make better decisions. Deep reinforcement learning can continuously optimize its own model according to feedback from the environment to adjust to the demands of multiple scenarios and tasks, and this algorithm is especially suitable for scenarios where the environment state is constantly changing; deep reinforcement learning is able to adapt to such changes through real-time exploration to obtain the best decision. Therefore, TOLB uses the TD3 algorithm to learn the environment model and the task-offloading and resource allocation strategies and then applies the TOPSIS-based server selection algorithm to select the low-load MEC servers to which the high-load MEC servers can migrate the tasks. TOLB is able to allocate resources in real-time changing environments, boosting the performance and robustness of the system, and TOLB also monitors the load state of the MEC servers, which significantly optimizes the system in terms of reduced latency, lower power consumption, and higher resource utilization.

4.1. MDP Model

An MDP is a sequential decision-making framework characterized by time-dependent and state-dependent properties. This study introduces three key elements into the MDP model.

4.1.1. State

In VEC, the state space contains information about several key vehicle and MEC server characteristics. The process of neural network training is significantly influenced by the variability and uncertainty of the value range of different task types, impacting the stability and convergence of the system. Therefore, we normalize the state values, defining the state s_t of time slot t as follows:

$$s_t = \left\{ \frac{X(t)}{L}, \frac{T(t)}{D_{\max}}, \frac{F(t)}{F} \right\} \tag{16}$$

where $X(t) = \{x_1(t), x_2(t), \ldots, x_N(t)\}$ denotes the position of each vehicle within time slot t, and L is the communication coverage of the RSU. $T(t) = \{T_1(t), T_2(t), \ldots, T_N(t)\}$ indicates the task produced by each vehicle during time slot t, and $D_{\max}$ denotes the maximum value of the task data volume. $F(t) = \{F_1(t), F_2(t), \ldots, F_M(t)\}$ denotes the resource margin of each MEC server in time slot t, and F represents the maximum resource capacity of the MEC server.

4.1.2. Action

The purpose of the agent is to have the state space mapped to the action space. To maximize the immediate payoff, the intelligent entity selects an action depending on the system state s_t during time slot t, determining the offload ratio of the task and the computational power assigned to the offloaded subtask. The action a_t is denoted by

$$a_t = \left\{ \lambda(t), f^l(t), f^{MEC}(t) \right\} \tag{17}$$

where $\lambda(t) = \{\lambda_1(t), \ldots, \lambda_n(t), \ldots, \lambda_N(t)\}$ denotes the task offload rate for each task during time slot t, $f^l(t) = \left\{ f_1^l(t), \ldots, f_n^l(t), \ldots, f_N^l(t) \right\}$ denotes the computing power allocated by the system to the task to execute locally during time slot t, and

$f^{MEC}(t) = \{f_1^{MEC}(t), \ldots, f_n^{MEC}(t), \ldots, f_N^{MEC}(t)\}$ represents the computing power assigned by the system to run the task locally and on the MEC server during time slot t.

4.1.3. Reward

On the basis of the current action a_t, the agent obtains a reward via the environment. The optimization goal of this system is to minimize the cost of the system; thus, the lower the weighted sum of the execution delay and energy consumption of the execution task, the better the action decision. The reward function is represented by

$$r = -\sum_{t=0}^{T-1} U(t) \tag{18}$$

4.2. TD3-Based Task-Offloading and Resource Allocation Algorithm

Considering that the IoV environment changes all the time, in an effort to find the optimal task-offloading and resource allocation decision in the complex and changing environment, TOLB has designed a task-offloading and resource allocation algorithm based on TD3, the architecture of which is displayed in Figure 2.

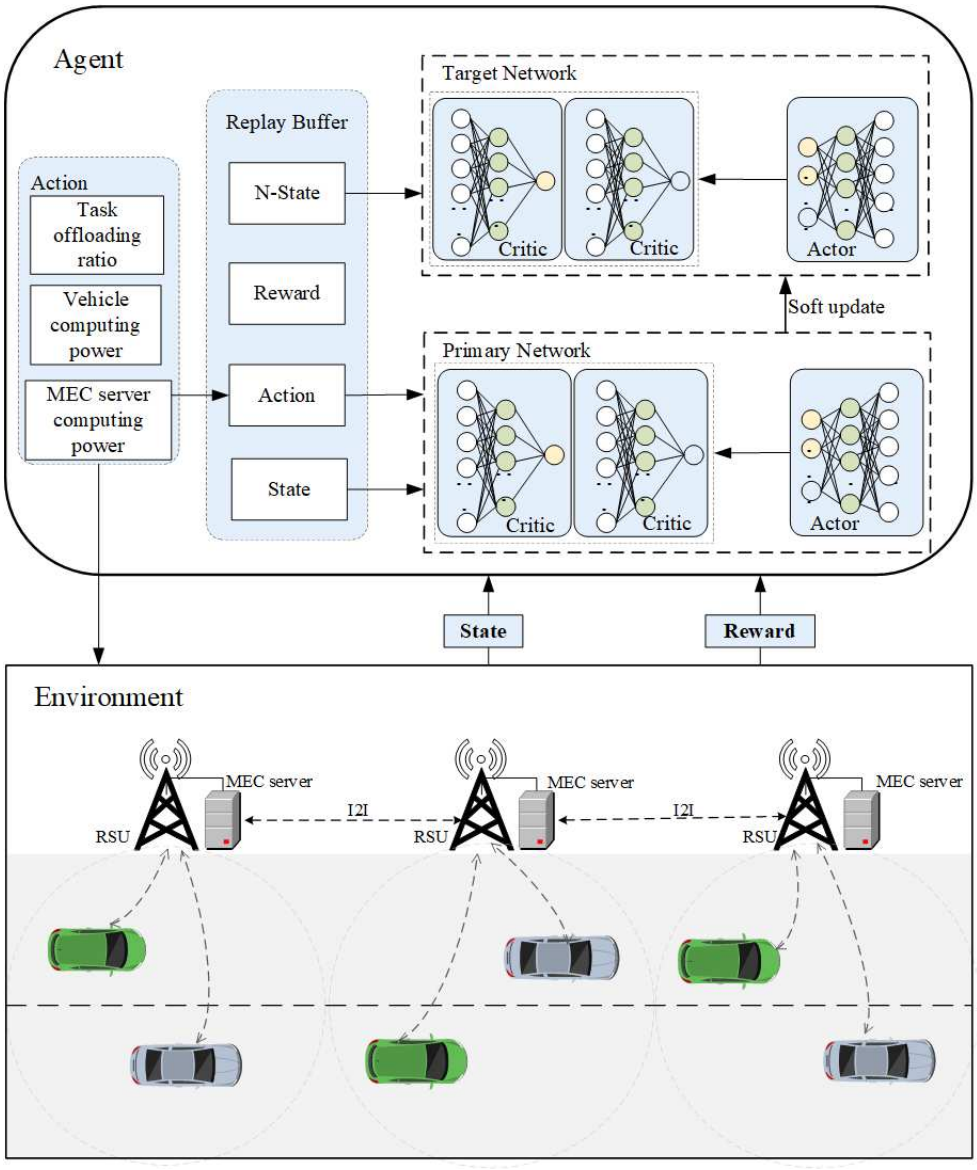

Figure 2. TD3-based task-offloading and resource allocation algorithm network model.

The TD3 algorithm has three components, including the primary network, the target network, and the experience pool. The primary network is built from two critic networks and one actor network. The TD3 agent in the VEC system maps the state space, which consists of information about the main features of the vehicle and the MEC server, to the action space. The actor network generates exploration strategies, and the two critic networks evaluate the strategies. The primary network's inputs are the initial state of the system, the state of the training process, and a summary of the training actions stored in the replay buffer. The outputs are actions consisting of the offload ratio of the task and the computational power to perform the task. The target network facilitates the training process of the primary network and calculates the target value. The replay buffer, on the other hand, records the states, actions, and rewards experienced by the TD3 agent during the learning process for the actor network to be trained. The TD3 agent is able to obtain action strategies with increasingly better reward values through the trial and error of continuous interaction between the agent and the system environment and to attain the optimization goal of minimizing the average cost of the system. The running process of the TD3-based task-offloading and resource allocation algorithm is presented below.

First, three networks, the critic network Q_{θ_1}, the critic network Q_{θ_2}, and the actor network π_ϕ, are initialized with randomized parameters θ_1, θ_2, and ϕ. Next, the three target networks are initialized accordingly for the above three networks. Subsequently, the parameters of the target network are taken from the primary network such that $\theta_1' = \theta_1$, $\theta_2' = \theta_2$, and $\phi' = \phi$, while the replay buffer B, which stores the experience of various actions, is initialized.

Each state s_t in the set selects an action a_t on the basis of the current policy and noise. To execute the action, the vehicle offloads a partial task to the MEC server, then the vehicle computes the partial task that is processed locally, and the MEC server computes the offloaded partial task. The TD3 agent observes the next state s_{t+1} and receives an instant reward r_t. (s_t, a_t, r_t, s_{t+1}) is then placed in B for training. Once the number of training experiences in B reaches a certain threshold, a batch of data of size H will be randomly selected from B. a_{t+1} is computed at state s_{t+1} according to Equation (19), and the Q-target value is obtained according to Equation (20). The formula is denoted by

$$a' = clip\Big(\pi(s') + clip(\varepsilon, -c, c), a_{low}, a_{high}\Big) \tag{19}$$

$$Q_{target} = r + \gamma \min_{i=1,2} Q_{\phi_i}(s', a') \tag{20}$$

where $clip(\varepsilon, -c, c)$ restricts ε to the range between $-c$ and c, and π represents the target actor network.

Simultaneously, the Q-values are evaluated. The critic network update operation is described as follows:

$$\theta_i = arg\min \frac{1}{N} \sum \Big(Q_{target} - Q_{\theta_i(s_t, a_t)}\Big)^2 \tag{21}$$

The strategy for the TD3 actor network utilizes a delayed update approach, whereby the actor network is updated after step k, and ϕ is updated through a deterministic policy gradient, as expressed below:

$$\nabla_\phi J(\phi) = \frac{1}{N} \sum \nabla_{a_t} Q_{\theta_1}(s_t, s_a)\Big|_{a=\pi_\phi(s)} \nabla_\phi \pi_\phi(s) \tag{22}$$

Eventually, the target network is updated through a soft update method described as

$$\theta_i' = \zeta\theta_i + (1-\zeta)\theta_i' \tag{23}$$

$$\phi_i' = \zeta\phi_i + (1-\zeta)\phi_i' \tag{24}$$

where the soft update factor is denoted by ζ.

Algorithm 1 outlines our proposed TD3-based task-offloading and resource allocation algorithm.

Algorithm 1 TD3-based task-offloading and resource allocation algorithm

Input:
Output:

Initialize the network parameters $\theta_i (i = 1, 2)$ and ϕ randomly.
To initialize the parameters of target networks, press $\theta_i' \leftarrow \theta_i (i = 1, 2)$ and $\phi' \leftarrow \phi$
To initialize the replay buffer B
for each episode **do**
  for each time slot **do**
    Define the current round's noise N_t and initialize the first state s_t.
    Select an action a_t on the basis of the current policy and action noise ε.
    Perform the action a_t and gain reward r_t and the next state s_{t+1}.
    Save (s_t, a_t, r_t, s_{t+1}) in the replay buffer B.
    Batches of data are randomly selected from the replay buffer B.
    According to Equation (19), we obtain a_{t+1}.
    According to Equation (20), we obtain Q_{target}.
    Updating critics $\theta_i (i = 1, 2)$ through Equation (21).
    If the delayed update condition is satisfied.
      Update ϕ through Equation (22);
      Update the target network through Equations (23) and (24).
  end for
end for

4.3. TOPSIS-Based Server Selection Algorithm

When the offload task arrives at MEC server m, if MEC server m has sufficient resources, the task can be executed immediately. Otherwise, the system will select appropriate low-load servers for task migration, and the tasks will be handled by the low-load MEC servers in the way of edge collaboration, with the computation output returned to MEC server m to fulfill the load balancing of the MEC servers.

TOLB uses a TOPSIS-based server selection algorithm to pick low-load servers for task migration processing. This selection process involves a comprehensive multi-indicator evaluation, where the TOLB evaluation model first considers the server's available resource margin and distance as evaluation indicators. It then applies AHP to assign different weights based on the importance of each evaluation indicator in the system. Finally, the TOPSIS technique is adopted to calculate the comprehensive evaluation scores of each MEC server, and the MEC server with the highest comprehensive evaluation score is the best selection. The TOPSIS-based server selection algorithm comprises the following steps.

When MEC server m is under a high load, each of the other servers in the system becomes an option, with the distance between these servers and the available resource margin used as evaluation metrics, with their matrix defined by $B = \left\{b_{ij}\right\}_{2\times(M-1)}$. A shorter distance between other servers and m and a larger available resource margin of the other servers are indicators of a better option. The bigger and better and smaller and better indicators can be normalized using Equations (25) and (26) to obtain the normalization matrix $C = \left\{c_{ij}\right\}_{2\times(M-1)}$.

$$c_{ij} = \frac{b_{ij} - \left(b_{ij}\right)_{\min}}{\left(b_{ij}\right)_{\max} - \left(b_{ij}\right)_{\min}} \tag{25}$$

$$c_{ij} = \frac{\left(b_{ij}\right)_{\max} - b_{ij}}{\left(b_{ij}\right)_{\max} - \left(b_{ij}\right)_{\min}} \tag{26}$$

Considering the task's latency sensitivity and the MEC server's load, the available resources are prioritized among the evaluation metrics, and their importance is increased. The metric weight vector $W = (\omega_1, \omega_2)^T$ is computed using AHP.

The weights $W = (\omega_1, \omega_2)^T$ obtained via AHP are then multiplied by the normalization matrix C to derive a weighted normalization matrix.

$$D = \{d_{ij}\} = (\omega_i c_{ij})_{2\times(M-1)} \tag{27}$$

The optimal and worst samples are determined according to the following formulas:

$$D^+ = \{\max d_{ij}\} = [d_1^+, d_2^+] \tag{28}$$

$$D^- = \{\min d_{ij}\} = [d_1^-, d_2^-] \tag{29}$$

The Euclidean distances between every sample and the highest and lowest samples are calculated using Equations (30) and (31).

$$E_j^+ = \sqrt{\sum_{i=1}^{2} (d_{ij} - d_i^+)^2} \tag{30}$$

$$E_j^- = \sqrt{\sum_{i=1}^{2} (d_{ij} - d_i^-)^2} \tag{31}$$

Equation (32) is utilized to obtain the ratings for each evaluator.

$$S_i = \frac{E_i^-}{E_i^- + E_i^+} i = 1, 2, \ldots, M-1 \tag{32}$$

Based on the magnitude of S_i, each evaluation object is ranked, with a larger value indicating closer proximity to the ideal condition. Ultimately, the server with the highest value is taken as the migration object. Algorithm 2 demonstrates the implementation of the TOPSIS-based server selection algorithm.

Algorithm 2 TOPSIS-based server selection algorithm

Input: MEC server location, total resources, computing power
Output: MEC servers with the highest overall evaluation value

1: Form evaluation objects and indicators into a $2 \times (M-1)$ matrix.
2: Harmonize the types of evaluation indicators and the positive orientation of indicators.
3: Normalize matrices that have been normalized using Equations (25) and (26).
4: Determine indicator weights using hierarchical analysis.
5: Multiply the normalization matrix by the resulting weights via Equation (27) to obtain the weighted normalization matrix.
6: Calculate the optimal and worst samples according to Equations (28) and (29).
7: By using Equations (30) and (31), calculate the Euclidean distances from each sample to the best and worst samples.
8: Calculate the score of each evaluation object according to Equation (32).
9: Select the highest-rated evaluator.

5. Performance Simulation and Result Analysis

This section begins with a brief overview of the simulation experimental environment of this paper, followed by a brief description of the parameter settings of the experiment, and concludes with an experimental comparison and performance evaluation of the TOLB scheme and other baseline methods proposed in this section.

5.1. Experimental Environment and Parameter Settings

The described experiment utilized Python 3.9.7 and TensorFlow 2.6.0 to simulate a VEC environment with multiple vehicles and servers. The simulation experiments were conducted on a server that is equipped as follows: an Intel Core i5-8300H processor, 8 GB RAM, and an NVIDIA GTX1050. Table 1 details the simulation parameters applied in the experiments, some of which reference the environmental parameter settings in [34].

Table 1. System environment and related parameters of TOLB.

Explanation	Parameter	Quantity
RSU coverage	L	100 m
Speed of vehicles	v	20 Km/h
Wireless channel bandwidth	$B_{n,m}$	20 MHz
Channel gain	$h_{n,m}$	2
Gaussian white noise power	N_0	−174 dBm
Vehicle transmission power	ρ_n	1 w
Maximum computing power of vehicle	F_n^l	0.5 GHz
Maximum computing power of VEC server	F_m^{MEC}	5 GHz
Task size	d_n	[100, 200]
Latency weight	u^t	0.5
Energy consumption weighting	u^e	0.5

5.2. Analysis of Results

This study evaluated the performance of TOLB by testing the following different offloading schemes against TOLB.

1. TD3-based task offloading and resource allocation (TD3-TR). In the TD3-TR scheme, edge collaboration is not considered, and the task-offloading and computational resource allocation policies are derived from the continuous interaction between TD3 intelligence and the environment.
2. DDPG-based task-offloading and resource allocation (DDPG-TR) scheme. In the DDPG-TR scheme, the task-offloading and computational resource allocation policy is decided by DDPG intelligence.
3. All Edge Offloading (AEO). In the AEO scheme, all tasks generated by vehicle users are offloaded to the MEC server for handling.
4. All Random Offloading (ARO) scheme. In the ARO scheme, the tasks generated by the vehicle users are partially offloaded to the MEC server for handling, and the task-offloading ratio is randomly generated within a given range.

Figure 3 demonstrates the effect of varying learning rates on the TOLB system's average cost. In DRL, the learning rate critically influences convergence; very high rates can destabilize the algorithm and hinder convergence, whereas very low rates slow convergence, potentially leading to suboptimal solutions. Typically, the actor and critic networks have identical learning rates. At a learning rate of 1×10^{-4}, the system's average cost stabilizes over iterations but converges more gradually, which is attributed to the slow neural network updates necessitated by the lower learning rate. A learning rate of 1×10^{-3} achieves faster convergence and attains the global optimum. However, a rate of 1×10^{-2}, while leading to rapid convergence, fails to reach the optimum achieved at 1×10^{-3}, suggesting that very high rates may bypass the global optimum, thus compromising algorithm performance. Therefore, a learning rate of L = 1×10^{-3} was used for all the subsequent experiments in this paper.

Figure 4 depicts the changes in the average cost of the system across different schemes. The figure shows that the average system costs of AEO and ARO remain constant as the number of iterations increases due to the unchanging task-offloading pattern. TD3-TR and TOLB outperform DDPG-TR in terms of convergence. This is because TD3-TR and TOLB employ the TD3 algorithm, which is an Upgraded version of the DDPG algorithm, to generate task-offloading and resource allocation decisions. TOLB considers edge server

loads and uses edge collaboration to execute tasks on the MEC server. TOLB uses edge collaboration to execute tasks, and in the case of overloaded MEC servers, tasks can be moved to other MEC servers for handling, reducing the task delay and hence the average cost of the system, while TD3-TR cannot avoid the delay caused by the high load on MEC servers. TD3-TR cannot avoid the large waiting latency due to the high load on MEC servers. The experiments confirm the effectiveness of the TOLB system, where the average cost of the system with the TOLB scheme is reduced by 7.2%, 11.4%, 52.7%, and 61.1% compared to TD3-TR, DDPG-TR, AEO, and ARO, respectively.

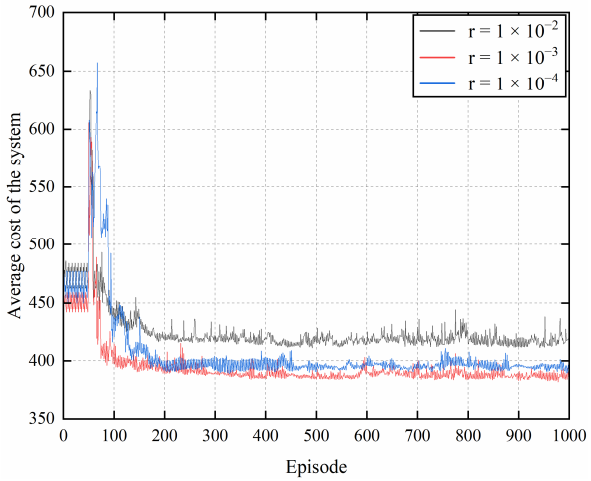

Figure 3. The average cost of the system with different learning rates.

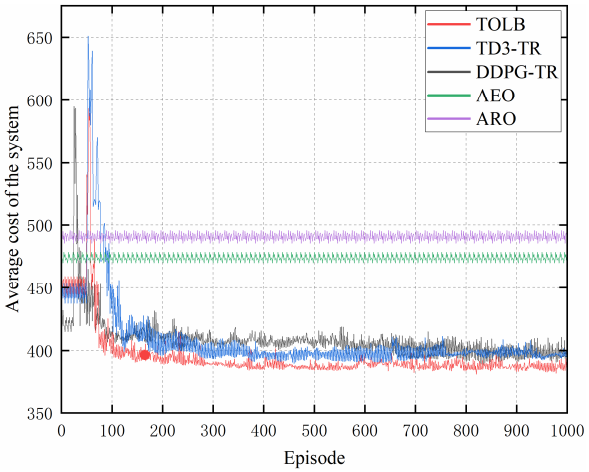

Figure 4. The average cost of the system with different schemes.

Figure 5 illustrates how the average cost of the system changes as the number of vehicles increases in different schemes. The figure shows that the average system cost increases as the number of vehicles increases in all five scenarios. At the beginning of the rise in vehicle number, due to the lack of good offloading strategies, the average cost of the system with AEO and ARO is higher than that of other strategies, and the average cost of the system with AEO is lower than that of ARO. With increasing numbers of vehicles, there are more and more tasks that need to be offloaded from the system, and the resources of the MEC servers become increasingly strained. TOLB can solve the problem of server resource constraints through edge collaboration, which solves the problem of insufficient server resources, so the upward trajectory of the average cost of the system in TOLB is slower than that of TD3-TR and DDPG-TR, and the average cost of the system in TOLB is the smallest.

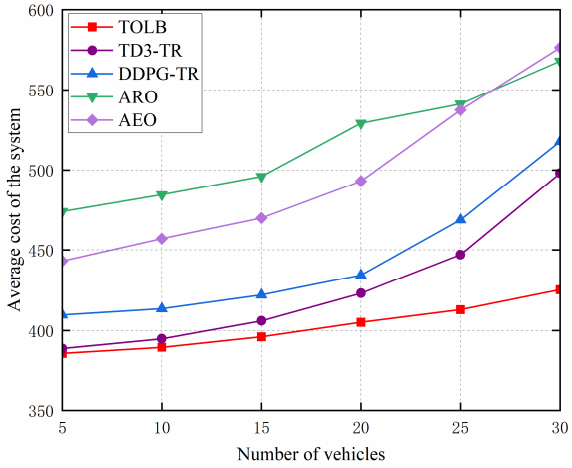

Figure 5. The average cost of the system with different numbers of vehicles.

Figure 6 illustrates how the average cost of the system evolves with increasing task data volume under various schemes. As the amount of data grows, the average system cost of the five schemes also increases. This is because larger data sizes demand more computing resources and are more difficult to handle, leading to greater increases in latency and energy consumption. Among all the schemes, TOLB exhibits the smallest average cost of the system because it allocates system resources by ensuring the optimal offloading of the running tasks. In contrast, the average cost of the system with TD3-TR and DDPG-TR becomes worse as the task volume becomes larger because they do not consider factors such as resource constraints and load balancing, which leads to irrational resource allocation. AEO and ARO do not consider task offloading and resource allocation, and therefore, the average cost of the system with AEO and ARO is consistently higher than that of the other schemes, and it increases as the task volume changes, rising sharply. The figure clearly shows that the average cost of the system with TOLB is better than with TD3-TR, DDPG-TR, AEO, and ARO under different average task size scales.

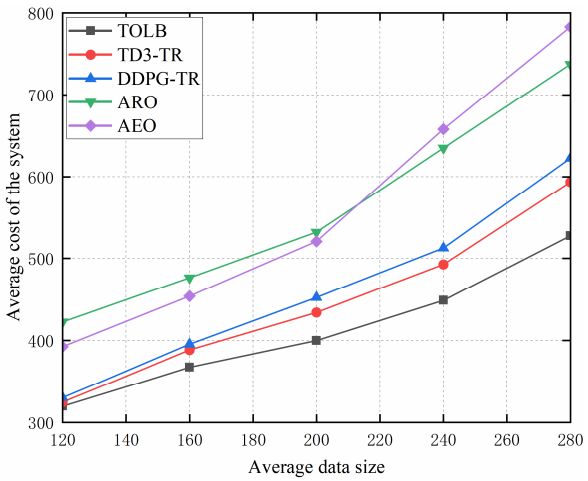

Figure 6. The average cost of the system with different average task data sizes.

Figure 7 presents a comparison of the average cost of the system for different numbers of tasks. Among the five scenarios, with a consistent increase in the average system cost, TOLB shows the best performance. When there are few tasks, there are sufficient resources in the system, so the average cost of the system between TOLB, TD3-TR, and DDPG-TR is not significant. As the number of tasks gradually grows, the communication and computation resources within the VEC system are subsequently strained, and TD3-TR, DDPG-TR, AEO, and ARO are unable to resolve the competition for communication and

computation resources due to the rising number of tasks, resulting in an increase in the task waiting delay and an increase in the average cost of the system.

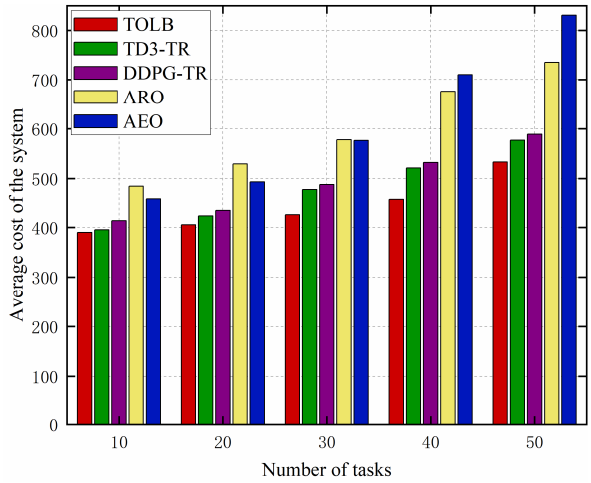

Figure 7. The average cost of the system with different numbers of tasks.

Figure 8 depicts how the average cost of the system changes with the number of MEC servers for the different scenarios. The figure shows that as the number of MEC servers increases, the average cost of the system decreases for all five scenarios. This is because, with the increase in the number of MEC servers, the vehicles are allocated more computational resources, so the difference in the average cost of the system between TOLB and TD3-TR and DDPG-TR is not much when the number of MEC servers becomes more. AEO and ARO also have a decrease in the average cost of the system due to the increase in computational resources. And since AEO offloads all the tasks to the MEC servers, the task execution latency of AEO is smaller than the task execution latency of ARO, which results in the average cost of the system with AEO being lower than the average cost of the system with ARO.

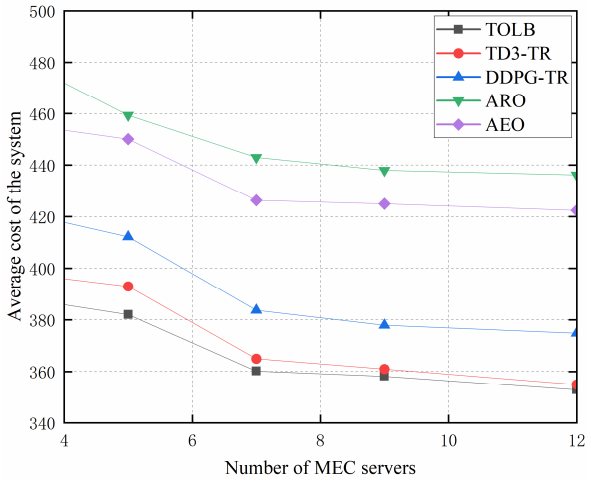

Figure 8. The average cost of the system with different numbers of MEC servers.

Figure 9 illustrates the comparison of the average cost of the system with the five schemes with different MEC server computing power. This figure shows that AEO leads to the largest average cost of the system and the slowest rate of average system cost reduction, which is due to the fact that task offloading using TOLB, TD3-TR, DDPG-TR, and ARO considers the partial offloading of tasks, and the vehicle and MEC servers process the tasks together, so the tasks will be executed with less latency, and the average cost of the system will be reduced. TOLB and TD3-TR use more advanced algorithms, so the average system

cost of TOLB and TD3-TR is lower than the average cost of the system with DDPG-TR. TOLB considers the load balancing of MEC servers, so TOLB performs better than TD3-TR when the MEC computational power is low. And when the MEC server computational power is high enough, TOLB and TD3-TR perform as well as TD3-TR, which shows that the load level of the edge servers in a VEC system has a strong influence on the performance of the system.

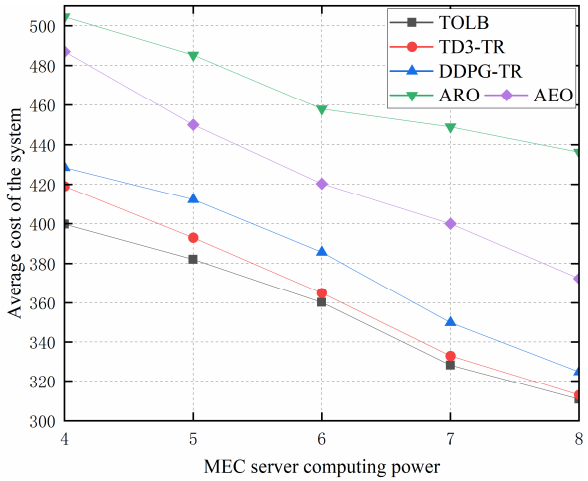

Figure 9. The average cost of the system with different MEC server computing power.

6. Summary

In this paper, we examine a multi-vehicle and multi-server environment under VEC and propose a deep reinforcement learning-based computational offloading and load-balancing decision-making scheme. This scheme enhances system stability by optimizing the offloading strategy of tasks while maintaining the server load balance. Specifically, this paper first models the dynamic interaction between the vehicle and the VEC server as an MDP and defines the weighted sum of the delay and energy consumption from executing the task as the system cost; then, the optimal policy is determined using the TD3-based task-offloading and resource allocation algorithm; and finally, the server load problem is considered, and for highly loaded servers that need to be processed for task migration, the optimal edge server collaboration object is selected using the TOPSIS-based server selection algorithm to achieve the minimization of the system cost under server load balancing. Simulation experiments demonstrate that the proposed TOLB scheme significantly reduces the task processing delay and energy consumption, decreases the likelihood of VEC server load imbalance, and enhances system performance. However, this scheme only considers task offloading while the vehicle is within the current RSU communication range and ignores the situation where the vehicle enters the next RSU communication range during task offloading and computation. Therefore, the situation where the vehicle enters the next RSU communication area during the data transfer and waits for the return of the computation results will be considered in future research.

Author Contributions: Conceptualization, Z.J., Z.W., X.P. and S.Z.; methodology, Z.J., Z.W., X.P. and S.Z.; software, Z.W. and Z.J.; validation, Z.W. and S.Z.; formal analysis, Z.W.; investigation, Z.J. and Z.W.; resources, Z.J. and Z.W.; writing—original draft preparation, Z.W. and Z.J.; writing—review and editing, Z.J. and Z.W.; visualization, Z.J.; supervision, Z.J. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the National Natural Science Foundation of China Youth Fund (62202145).

Data Availability Statement: Data is contained within the article.

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

References

1. Wang, L.-L.; Gui, J.-S.; Deng, X.-H.; Zeng, F.; Kuang, Z.-F. Routing algorithm based on vehicle position analysis for internet of vehicles. *IEEE Internet Things J.* **2020**, *7*, 11701–11712. [CrossRef]
2. Fan, W.; Hua, M.; Zhang, Y.; Su, Y.; Li, X.; Wu, F.; Liu, Y.a. Game-based task offloading and resource allocation for vehicular edge computing with edge-edge cooperation. *IEEE Trans. Veh. Technol.* **2023**, *72*, 7857–7870. [CrossRef]
3. Yan, R.; Gu, Y.; Zhang, Z.; Jiao, S. Vehicle Trajectory Prediction Method for Task Offloading in Vehicular Edge Computing. *Sensors* **2023**, *23*, 7954. [CrossRef]
4. Liu, Y.; Wang, S.; Zhao, Q.; Du, S.; Zhou, A.; Ma, X.; Yang, F. Dependency-aware task scheduling in vehicular edge computing. *IEEE Internet Things J.* **2020**, *7*, 4961–4971. [CrossRef]
5. Li, Y.; Yang, C.; Chen, X.; Liu, Y. Mobility and dependency-aware task offloading for intelligent assisted driving in vehicular edge computing networks. *Veh. Commun.* **2024**, *45*, 100720. [CrossRef]
6. Yao, L.; Xu, X.; Bilal, M.; Wang, H. Dynamic edge computation offloading for internet of vehicles with deep reinforcement learning. *IEEE Trans. Intell. Transp. Syst.* **2022**, *24*, 12991–12999. [CrossRef]
7. Moghaddasi, K.; Rajabi, S.; Soleimanian Gharehchopogh, F.; Hosseinzadeh, M. An Energy-Efficient Data Offloading Strategy for 5G-Enabled Vehicular Edge Computing Networks Using Double Deep Q-Network. *Wirel. Pers. Commun.* **2023**, *133*, 2019–2064. [CrossRef]
8. Fang, C.; Hu, Z.; Meng, X.; Tu, S.; Wang, Z.; Zeng, D.; Ni, W.; Guo, S.; Han, Z. Drl-driven joint task offloading and resource allocation for energy-efficient content delivery in cloud-edge cooperation networks. *IEEE Trans. Veh. Technol.* **2023**, *72*, 16195–16207. [CrossRef]
9. Sun, D.; Chen, Y.; Li, H. Intelligent Vehicle Computation Offloading in Vehicular Ad Hoc Networks: A Multi-Agent LSTM Approach with Deep Reinforcement Learning. *Mathematics* **2024**, *12*, 424. [CrossRef]
10. Shi, Y.; Chu, J.; Sun, X.; Ning, S. A computation offloading method with distributed double deep Q-network for connected vehicle platooning with vehicle-to-infrastructure communications. *IET Intell. Transp. Syst.* **2023**. *online version of record before inclusion in an issue*. [CrossRef]
11. Li, P.; Xie, W.; Yuan, Y.; Chen, C.; Wan, S. Deep reinforcement learning for load balancing of edge servers in iov. *Mob. Netw. Appl.* **2022**, *27*, 1461–1474. [CrossRef]
12. Wang, X.; Ning, Z.; Guo, S.; Wang, L. Imitation learning enabled task scheduling for online vehicular edge computing. *IEEE Trans. Mob. Comput.* **2020**, *21*, 598–611. [CrossRef]
13. Min, H.; Li, Y.; Wu, X.; Wang, W.; Chen, L.; Zhao, X. A measurement scheduling method for multi-vehicle cooperative localization considering state correlation. *Veh. Commun.* **2023**, *44*, 100682. [CrossRef]
14. Peng, X.; Han, Z.; Xie, W.; Yu, C.; Zhu, P.; Xiao, J.; Yang, J. Deep reinforcement learning for shared offloading strategy in vehicle edge computing. *IEEE Syst. J.* **2022**, *17*, 2089–2100. [CrossRef]
15. Ning, Z.; Zhang, K.; Wang, X.; Guo, L.; Hu, X.; Huang, J.; Hu, B.; Kwok, R.Y. Intelligent edge computing in internet of vehicles: A joint computation offloading and caching solution. *IEEE Trans. Intell. Transp. Syst.* **2020**, *22*, 2212–2225. [CrossRef]
16. Zeng, F.; Chen, Q.; Meng, L.; Wu, J. Volunteer assisted collaborative offloading and resource allocation in vehicular edge computing. *IEEE Trans. Intell. Transp. Syst.* **2020**, *22*, 3247–3257. [CrossRef]
17. Liu, S.; Tian, J.; Deng, X.; Zhi, Y.; Bian, J. Stackelberg game-based task offloading in vehicular edge computing networks. *Int. J. Commun. Syst.* **2021**, *34*, e4947. [CrossRef]
18. Lin, Y.; Zhang, Y.; Li, J.; Shu, F.; Li, C. Popularity-aware online task offloading for heterogeneous vehicular edge computing using contextual clustering of bandits. *IEEE Internet Things J.* **2021**, *9*, 5422–5433. [CrossRef]
19. Luo, Q.; Li, C.; Luan, T.H.; Shi, W.; Wu, W. Self-learning based computation offloading for internet of vehicles: Model and algorithm. *IEEE Trans. Wirel. Commun.* **2021**, *20*, 5913–5925. [CrossRef]
20. Ju, Y.; Cao, Z.; Chen, Y.; Liu, L.; Pei, Q.; Mumtaz, S.; Dong, M.; Guizani, M. NOMA-assisted secure offloading for vehicular edge computing networks with asynchronous deep reinforcement learning. *IEEE Trans. Intell. Transp. Syst.* **2023**, 1–14. [CrossRef]
21. Wang, G.; Xu, F. Regional intelligent resource allocation in mobile edge computing based vehicular network. *IEEE Access* **2020**, *8*, 7173–7182. [CrossRef]
22. Pang, H.; Wang, Z. Dueling Double Deep Q Network Strategy in MEC for Smart Internet of Vehicles Edge Computing Networks. *J. Grid Comput.* **2024**, *22*, 1–12. [CrossRef]
23. Zheng, J.; Zhang, Y.; Luan, T.H.; Mu, P.K.; Li, G.; Dong, M.; Wu, Y. Digital Twin Enabled Task Offloading for IoVs: A Learning-Based Approach. *IEEE Trans. Netw. Sci. Eng.* **2023**, *11*, 659–672. [CrossRef]
24. Liu, G.; Dai, F.; Huang, B.; Qiang, Z.; Wang, S.; Li, L. A collaborative computation and dependency-aware task offloading method for vehicular edge computing: A reinforcement learning approach. *J. Cloud Comput.* **2022**, *11*, 68. [CrossRef]
25. Shi, J.; Du, J.; Shen, Y.; Wang, J.; Yuan, J.; Han, Z. DRL-based V2V computation offloading for blockchain-enabled vehicular networks. *IEEE Trans. Mob. Comput.* **2022**, *22*, 3882–3897. [CrossRef]
26. Long, D.; Wu, Q.; Fan, Q.; Fan, P.; Li, Z.; Fan, J. A power allocation scheme for MIMO-NOMA and D2D vehicular edge computing based on decentralized DRL. *Sensors* **2023**, *23*, 3449. [CrossRef] [PubMed]
27. Dai, Y.; Xu, D.; Maharjan, S.; Zhang, Y. Joint load balancing and offloading in vehicular edge computing and networks. *IEEE Internet Things J.* **2018**, *6*, 4377–4387. [CrossRef]

28. Lu, Y.; Han, D.; Wang, X.; Gao, Q. Enhancing vehicular edge computing system through cooperative computation offloading. *Clust. Comput.* **2023**, *26*, 771–788. [CrossRef]
29. Gao, J.; Kuang, Z.; Gao, J.; Zhao, L. Joint offloading scheduling and resource allocation in vehicular edge computing: A two layer solution. *IEEE Trans. Veh. Technol.* **2022**, *72*, 3999–4009. [CrossRef]
30. Avgeris, M.; Mechennef, M.; Leivadeas, A.; Lambadaris, I. A two-stage cooperative reinforcement learning scheme for energy-aware computational offloading. In Proceedings of the 2023 IEEE 24th International Conference on High Performance Switching and Routing (HPSR), Albuquerque, NM, USA, 5–7 June 2023; pp. 179–184.
31. Wu, X.; Dong, S.; Hu, J.; Huang, Z. An efficient many-objective optimization algorithm for computation offloading in heterogeneous vehicular edge computing network. *Simul. Model. Pract. Theory* **2024**, *131*, 102870. [CrossRef]
32. Lv, W.; Yang, P.; Zheng, T.; Yi, B.; Ding, Y.; Wang, Q.; Deng, M. Energy consumption and qos-aware co-offloading for vehicular edge computing. *IEEE Internet Things J.* **2022**, *10*, 5214–5225. [CrossRef]
33. Bi, J.; Yuan, H.; Duanmu, S.; Zhou, M.; Abusorrah, A. Energy-optimized partial computation offloading in mobile-edge computing with genetic simulated-annealing-based particle swarm optimization. *IEEE Internet Things J.* **2020**, *8*, 3774–3785. [CrossRef]
34. Ju, Y.; Chen, Y.; Cao, Z.; Liu, L.; Pei, Q.; Xiao, M.; Ota, K.; Dong, M.; Leung, V.C. Joint secure offloading and resource allocation for vehicular edge computing network: A multi-agent deep reinforcement learning approach. *IEEE Trans. Intell. Transp. Syst.* **2023**, *24*, 5555–5569. [CrossRef]