# A DQN-Based Task Offloading Strategy for Autonomous Driving: Postprint

**Authors:** Wang Jin, Zhang Xinyou

**Date:** 2022-05-10T11:22:58Z

## Abstract

Due to their limited battery life and computing power, autonomous vehicles struggle to meet the processing demands of latency-sensitive or compute-intensive tasks while ensuring driving range. To address this issue, this paper proposes an autonomous vehicle task offloading strategy based on deep Q-network (DQN) in the context of mobile edge computing (MEC). First, a task priority-based collaborative task offloading model for "vehicle-edge-cloud" is defined, which requires joint optimization of vehicle computing capacity and task offloading strategy to minimize system latency and energy consumption. Since this problem is a mixed-integer nonlinear programming problem, it is solved in two steps: an analytical solution for optimal vehicle computing capacity is derived through mathematical derivation, and subsequently, with this value fixed, the optimal task offloading strategy is obtained based on the DQN algorithm. Finally, a simulation model is established by integrating tools such as SUMO, PyTorch, and Python, comparing the performance of the DQN algorithm with three other algorithms under varying task loads, MEC server computing capacities, and energy consumption weight coefficients. The experimental results verify the feasibility and superiority of the proposed strategy.

## Full Text

## Preamble

**Vol. 39 No. 9**
**Application Research of Computers**
**ChinaXiv Partner Journal**

# A DQN-Based Driverless Task Offloading Strategy

**Wang Jin, Zhang Xinyou†**
(School of Computing & Artificial Intelligence, Southwest Jiaotong University, Chengdu 611756, China)

**Abstract:** Due to limited battery life and computing power, driverless cars struggle to meet the processing demands of delay-sensitive or computationally intensive tasks while ensuring adequate driving range. To address this challenge, this paper proposes a driverless task offloading strategy based on Deep Q-Network (DQN) in the context of Mobile Edge Computing (MEC). First, we define a task priority-based "vehicle-edge-cloud" collaborative offloading model that jointly optimizes vehicle computing capability and task offloading strategy to minimize system latency and energy consumption. Since this problem constitutes a mixed-integer nonlinear programming problem, we solve it in two steps: (1) derive an analytical solution for optimal vehicle computing capability through mathematical derivation, and (2) obtain the optimal task offloading strategy using DQN under fixed vehicle computing capability. Finally, we establish a simulation model integrating SUMO, PyTorch, and Python tools to compare DQN against three alternative algorithms under varying task loads, MEC server computing capabilities, and energy consumption weight coefficients. Experimental results validate the feasibility and superiority of the proposed strategy.

**Keywords:** driverless cars; mobile edge computing; task offloading; deep Q-network; mobility

## 0 Introduction

In recent years, driverless vehicles have attracted widespread attention from both industry and academia due to their convenience and driving efficiency [**?**, **?**]. By deploying intelligent infrastructure such as Road Side Units (RSUs), vehicular users can access various services including route planning, navigation, and in-vehicle entertainment [**?**, **?**]. However, onboard equipment in driverless vehicles suffers from limited computing capability and battery capacity. Consequently, most vehicles currently offload computational tasks to cloud servers for processing [**?**]. While cloud servers can provide substantial computing resources, high transmission latency degrades service quality for delay-sensitive applications and may even lead to traffic accidents.

Mobile Edge Computing (MEC) offers a practical and effective solution for driverless technology development. By deploying MEC servers alongside RSUs, computing, storage, and communication resources are distributed to the network edge near users [**?**], thereby reducing transmission latency. Nevertheless, offloading all tasks to MEC servers risks overloading them. Therefore, local

vehicle processing and cloud servers remain necessary for handling partial computational tasks.

Through task offloading, computationally intensive tasks can be migrated to MEC servers, alleviating resource constraints in driverless vehicles, reducing their energy consumption, and enhancing driving range [**?**], while simultaneously improving task processing efficiency and reducing task completion latency. This paper introduces a three-layer network architecture—vehicle layer, edge layer, and cloud layer—to provide diverse offloading options for resource-constrained vehicles. Driverless vehicles can choose to process tasks locally or offload them to MEC servers or cloud servers based on actual conditions.

The task offloading problem in the vehicle-edge-cloud architecture can be formulated as a joint optimization problem that balances task loads across vehicles, MEC servers, and cloud servers under resource constraints to minimize latency and energy consumption. Considering algorithm stability and reliability, this paper employs a Deep Q-Network (DQN)-based algorithm to solve this optimization problem. The main contributions are threefold:

a) We propose a task priority-based "vehicle-edge-cloud" collaborative offloading model that jointly optimizes vehicle computing capability and task offloading strategy to minimize system latency and energy consumption.

b) The problem is a mixed-integer nonlinear programming problem that is NP-hard. To reduce algorithmic complexity, we solve it in two steps. First, we derive an analytical solution for optimal vehicle computing capability through mathematical derivation. Then, with this value fixed, we obtain the optimal task offloading strategy using DQN.

c) We conduct comprehensive simulation experiments. Results demonstrate the feasibility and superiority of the proposed algorithm.

## 1 Research Status

Task offloading technology addresses insufficient device computing resources by migrating computational tasks to edge nodes or cloud servers. Numerous researchers have investigated multi-user MEC task offloading problems in various scenarios. Based on different optimization objectives, prior work can be categorized into three classes:

a) **Latency optimization.** To address complex task offloading processes and long response times in multi-cloud environments, reference [**?**] proposed a weighted adaptive inertia weight particle swarm optimization algorithm based on multi-cloudlet collaboration. Experimental results show that this scheme effectively reduces task offloading time compared to non-collaborative approaches. To accelerate task processing efficiency in vehicular network environments, reference [**?**] proposed an AHP-DQN-based task offloading algorithm that uses Analytic Hierarchy Process (AHP) to reasonably partition vehicle tasks and deploys offloading decisions based

on real-time channel gains. Extensive simulations demonstrate the algorithm' s ability to reduce task processing latency. However, this scheme does not optimize vehicle computing capability. Reference [**?**] proposed a DQN-based end-edge-cloud offloading scheme for vehicular environments considering resource allocation at both MEC and cloud servers, proving its superiority through extensive experiments. Nevertheless, this scheme does not consider task classification or priority issues.

b) **Energy consumption optimization.** Reference [**?**] designed a low-power edge computing system for autonomous driving robots and vehicle services on cost-effective embedded systems, successfully supporting multiple autonomous driving services with low energy consumption. Considering vehicle heterogeneity and RSU impact, reference [**?**] established a network model with heterogeneous vehicles and RSUs, proposing an algorithm to find optimal resource allocation strategies that effectively reduce energy consumption.

c) **Joint latency and energy optimization.** Reference [**?**] analyzed time distributions for general task computation and server queuing delays, proposing an Integer Linear Programming (ILP)-based algorithm to optimize system latency and total server energy consumption, with experimental results validating its effectiveness. To address emerging applications' requirements for low latency, high complexity, and high reliability, reference [**?**] proposed a cloud-edge collaborative task offloading algorithm based on service orchestration that effectively reduces both latency and energy consumption. Reference [**?**] proposed a DQN-based multi-objective task offloading algorithm to minimize task completion latency and energy consumption for resource-constrained devices, considering device mobility and task dependencies. Multiple test scenarios validated the algorithm' s effectiveness. However, this scheme also does not consider task classification or priority issues.

The aforementioned works provide feasible solutions for task offloading problems with different optimization objectives in device or vehicular network contexts, with latency-energy models offering valuable insights. However, the limited battery capacity of driverless vehicles cannot be ignored—vehicle computing capability requires optimization to reduce energy consumption. Furthermore, the high mobility of driverless vehicles may cause them to move beyond MEC servers' V2I communication range during travel. Finally, driverless vehicles generate various task types during operation, such as driving-related control command tasks or entertainment-related tasks, which differ in urgency. Most device-related literature does not consider vehicle mobility, while most vehicular network literature neglects task priority classification or vehicle computing capability optimization. Consequently, existing offloading schemes are unsuitable for driverless scenarios, and research on task offloading in driverless contexts remains limited.

This paper addresses the driverless scenario under a three-layer network architecture, considering vehicle computing capability optimization, mobility issues,

and task priority classification to construct a novel task offloading model.

## 2 System Model

This section first introduces the network model, communication model, computation model, and priority model for task offloading in driverless scenarios. Based on these components, we define an optimization problem that jointly optimizes vehicle computing capability and task offloading strategy to minimize system latency and energy consumption.

### 2.1 Network Model

The system comprises a three-layer network architecture consisting of a vehicle layer, edge layer, and cloud layer, as shown in Figure 1. The vehicle layer includes all driverless vehicles traveling on roads. We assume each vehicle is equipped with dual radio interfaces: an 802.11p interface (supporting V2X communication) and a cellular network interface (e.g., LTE-A) [**?**]. Vehicles can communicate with RSUs via the 802.11p interface. The edge layer consists of edge nodes located alongside roads, where each edge node includes an RSU and a co-located MEC server. RSUs have certain communication coverage capabilities for collecting information about vehicles, network conditions, and tasks. The cloud layer contains cloud servers. Driverless vehicles communicate with base stations via cellular network interfaces, which connect to cloud servers through wired links to offload tasks.

The system involves multiple driverless vehicles with computational tasks requiring timely processing. We assume tasks are indivisible—each task must be executed either locally, on an MEC server, or on a cloud server. When a task is offloaded to an MEC server, due to vehicle mobility, if the vehicle moves out of the RSU' s wireless coverage range before task completion, the task must be offloaded to a cloud server for execution. For clarity, Table 1 lists key symbols and their meanings.

**Table 1. Key Symbols**

| Symbol | Meaning |
|---|---|
| $N$ | Set of driverless vehicles |
| $K$ | Set of RSUs |
| $D_n$ | Task data size for vehicle $n$ |
| $C_n$ | Required CPU cycles to complete task |
| $\tau_n$ | Maximum tolerable delay |
| $p_n$ | Vehicle transmission power |
| $B_m$ | Total uplink bandwidth of MEC server |
| $B_{n,c}$ | Uplink bandwidth between vehicle $n$ and cloud server |
| $v_{n,m}$ | Transmission rate from vehicle $n$ to MEC server |
| $v_{n,c}$ | Transmission rate from vehicle $n$ to cloud server |

| Symbol | Meaning |
|---|---|
| $g_{n,m}$ | Channel gain between vehicle $n$ and MEC server |
| $g_{n,c}$ | Channel gain between vehicle $n$ and cloud server |
| $f_n$ | Computing capability of vehicle $n$ |
| $f_m$ | Computing capability of MEC server |
| $f_c$ | Computing capability of cloud server |
| $P_n$ | Priority level of vehicle task |

### 2.2 Communication Model

We first introduce the communication model. Driverless vehicles transmit tasks to MEC servers via wireless channels. According to Shannon' s formula, the transmission rate from vehicle $n$ to MEC server $m$ is:

$$v_{n,m} = B_m \log_2 \left(1 + \frac{p_n g_{n,m}}{N_0}\right)$$

where $p_n$ denotes vehicle $n$' s transmission power, $B_m$ denotes the MEC server' s total uplink bandwidth, and $N_0$ denotes noise power. $g_{n,m}$ represents the channel gain between vehicle $n$ and MEC server.

Driverless vehicles offload tasks to cloud servers via wireless channels. According to Shannon' s formula, the transmission rate from vehicle $n$ to cloud server is:

$$v_{n,c} = B_{n,c} \log_2 \left(1 + \frac{p_n g_{n,c}}{N_0}\right)$$

where $B_{n,c}$ denotes the uplink bandwidth between vehicle $n$ and cloud server, $N_0$ denotes noise power, and $g_{n,c}$ denotes the channel gain between vehicle $n$ and cloud server.

### 2.3 Computation Model

For the computation task generated by vehicle $n$ in the model, we describe it using a triple $\{D_n, C_n, \tau_n\}$, where $D_n$ represents the task data size of vehicle $n$, $C_n$ represents the required CPU cycles to complete the task, and $\tau_n$ represents the maximum tolerable delay. Let $A = \{a_1, a_2, \ldots, a_N\}$ denote the offloading decision, where $a_n = \{a_n^l, a_n^m, a_n^c\}$ represents the offloading choice for task $R_n$, with $a_n^l, a_n^m, a_n^c \in \{0, 1\}$ indicating whether the task executes locally, on the MEC server, or on the cloud server, respectively. A value of 1 indicates execution on that platform, and 0 indicates otherwise. Since each task can only execute on one platform, the following constraint must be satisfied:

$$a_n^l + a_n^m + a_n^c = 1$$

**2.3.1 Local Computation Model** When $a_n^l = 1$, the task executes locally. Let $f_n$ denote the computing capability of vehicle $n$, described by CPU cycle frequency (CPU cycles/s). Through Dynamic Voltage and Frequency Scaling (DVFS) technology [**?**], vehicles can balance execution time and energy consumption by adjusting CPU frequency per cycle. For simplicity, we assume $f_n$ remains constant within one scheduling period. Therefore, the local execution time is:

$$T_n^{loc} = \frac{C_n}{f_n}$$

The energy consumption for local task execution is:

$$E_n^{loc} = \kappa C_n f_n^2$$

where $\kappa$ is the power conversion coefficient, dependent on micro-circuit architecture, typically set to $\kappa = 10^{-27}$.

**2.3.2 MEC Computation Model** When $a_n^m = 1$, the task is offloaded to the MEC server for execution. The corresponding transmission time is:

$$T_n^{tran,m} = \frac{D_n}{v_{n,m}}$$

Assuming the computation result is very small, the delay and energy consumption for returning results from the MEC server to the vehicle are typically negligible [**?**, **?**].

The execution delay on the MEC server is:

$$T_n^{exe,m} = \frac{C_n}{f_m}$$

where $f_m$ represents the MEC server' s computing capability.

We assume the MEC server is single-core and can only execute one task at a time, so tasks offloaded to the MEC server must queue. When a task arrives at the MEC server, it is first added to a waiting queue where tasks are sorted by priority, with equal-priority tasks ordered by arrival time.

Let $T_n^{ts}$ denote the task data transmission start time, $T_n^{te}$ the transmission end time, $T_n^{cs}$ the computation start time, and $T_n^{ce}$ the computation completion time. We have:

$$T_n^{cs} = \max\{T_{n-1}^{ce}, T_n^{te}\}$$

The current task can only start computing after the previous task completes and the current task finishes transmission. This is expressed as:

$$T_n^{cs} = \max\{T_{n-1}^{ce}, T_n^{te}\}$$

where $T_{n-1}^{ce}$ denotes the previous task's completion time. The waiting time on the MEC server is the difference between computation start time and transmission end time:

$$T_n^{wait} = T_n^{cs} - T_n^{te}$$

Therefore, when a task executes on the MEC server, its total time consumption is the sum of transmission time, execution time, and waiting time:

$$T_n^{mec} = T_n^{tran,m} + T_n^{exe,m} + T_n^{wait}$$

Due to vehicle mobility, the total time to complete a task cannot exceed the connection time $T_{n,m}^{v2i}$ between vehicle $n$ and the MEC server:

$$T_n^{mec} \leq T_{n,m}^{v2i}$$

This paper optimizes energy consumption generated by driverless vehicles; energy consumption from MEC servers is not considered. Therefore, we only account for transmission energy consumption, expressed as:

$$E_n^{mec} = p_n T_n^{tran,m}$$

**b) Incomplete execution on MEC server requiring cloud offloading**

The MEC server processes tasks through wireless V2I communication. Due to vehicle mobility, task computation must complete within the limited V2I connection time. When a task cannot finish within the vehicle-MEC server connection time, it is transferred to the cloud server for re-execution, incurring additional wasted delay and energy from the MEC server attempt.

Regarding energy consumption, since this paper only optimizes vehicle energy consumption and disregards MEC server energy consumption, the wasted energy consists solely of transmission energy from vehicle to MEC server. This wasted time depends on several scenarios: (a) The vehicle moves out of V2I communication range before completing task transmission, disconnecting from the server—the wasted time equals the connection time. (b) The vehicle moves out of V2I range after full transmission but before MEC computation begins—the wasted time equals both transmission time and connection time. (c) The

vehicle moves out of range after MEC computation begins but before completion—the wasted time equals transmission time. Therefore, the wasted time on the MEC server is the minimum of transmission time and connection time:

$$T_n^{extra} = \min\{T_n^{tran,m}, T_{n,m}^{v2i}\}$$

The wasted energy on the MEC server is:

$$E_n^{extra} = p_n T_n^{extra}$$

The total delay on the cloud server is the sum of connection time with the MEC server and retransmission time to the cloud server:

$$T_n^{cloud'} = T_{n,m}^{v2i} + T_n^{tran,c}$$

The total energy consumption on the cloud server is the sum of wasted energy and retransmission energy:

$$E_n^{cloud'} = E_n^{extra} + p_n T_n^{tran,c}$$

### 2.3.3 Cloud Computing Model a) Direct cloud execution

When $a_n^c = 1$, the task executes on the cloud server, which is considered to have unlimited resources. Therefore, we ignore queuing delay on the cloud server and only consider transmission time, transmission energy, and computation time.

Transmission time to the cloud server is:

$$T_n^{tran,c} = \frac{D_n}{v_{n,c}}$$

Execution time on the cloud server is:

$$T_n^{exe,c} = \frac{C_n}{f_c}$$

where $f_c$ represents the cloud server' s computing capability.

Therefore, total delay on the cloud server is:

$$T_n^{cloud} = T_n^{tran,c} + T_n^{exe,c}$$

Energy consumption from cloud servers is not considered, so total energy consumption on the cloud server is:

$$E_n^{cloud} = p_n T_n^{tran,c}$$

### 2.4 Priority Model

This system considers task urgency and completion rate in task offloading to meet driverless performance requirements. It determines how to prioritize highly urgent tasks and maximize task completion. Given the special nature of driverless vehicle tasks, different tasks have different urgency levels requiring classification. This paper categorizes tasks into three levels based on urgency: very important tasks, important tasks, and generally important tasks [**?**], as shown in Table 2. Very important tasks typically have small data volumes and short tolerable delays [**?**]. Important tasks typically have larger data volumes and longer tolerable delays. Generally important tasks typically have high delay tolerance.

**Table 2. Prioritization of Vehicular Tasks**

| Priority Level | Task Examples | Characteristics |
|---|---|---|
| Very Important | Route planning, collision avoidance | Small data volume, short tolerable delay |
| Important | Navigation, information services | Larger data volume, longer tolerable delay |
| Generally Important | Music, video, games | Large data volume, long tolerable delay |

Very important tasks require priority service, while generally important tasks can be served last. We define the task priority parameter for vehicle $n$ as $P_n \in \{1, 2, 3\}$. Different vehicle tasks have different priority levels. In the MEC server queue, this paper employs a non-preemptive priority scheduling algorithm for task scheduling.

### 2.5 Problem Formulation

We define the total cost of completing a task as the weighted sum of latency and energy consumption. For task $R_n$, the latency cost is expressed as:

$$T_n = a_n^l T_n^{loc} + a_n^m T_n^{mec} + a_n^c T_n^{cloud}$$

Similarly, the energy cost can be expressed as:

$$E_n = a_n^l E_n^{loc} + a_n^m E_n^{mec} + a_n^c E_n^{cloud}$$

Therefore, the total cost of task $R_n$ is:

$$Z_n = \beta_T T_n + \beta_E E_n$$

where $\beta_T, \beta_E \in [0,1]$ and $\beta_T + \beta_E = 1$ are weight coefficients assigned to latency and energy consumption, respectively. $m = 1$ indicates the task is offloaded to the MEC server and can complete within the V2I connection time.

To meet different users' specific requirements, we can select different latency-energy weight coefficients. For example, driverless vehicles with smaller battery capacity would choose larger $\beta_E$ to conserve more resources, while vehicles with larger battery capacity would prefer larger $\beta_T$ to reduce delay and obtain better experience.

The system' s optimization problem within one scheduling period can be formulated as:

$$\begin{aligned}
\min_{A,F} \quad & \sum_{n \in N} Z_n \\
\text{s.t.} \quad & C1: a_n^l + a_n^m + a_n^c = 1, \quad \forall n \in N \\
& C2: T_n \le \tau_n, \quad \forall n \in N \\
& C3: 0 \le f_n \le f_n^{\max}, \quad \forall n \in N \\
& C4: T_n^{mec} \le T_{n,m}^{v2i}, \quad \text{if } a_n^m = 1 \\
& C5: P_n \in \{1,2,3\}, \quad \forall n \in N
\end{aligned}$$

where $C1$ ensures each task executes locally, on the MEC server, or on the cloud server; $C2$ ensures total delay for completing task $R_n$ does not exceed its maximum tolerable delay $\tau_n$; $C3$ constrains vehicle $n$' s computing capability within its maximum capacity; $C4$ ensures tasks complete on the MEC server within V2I connection time; and $C5$ defines task priority levels.

## 3 Algorithm Design

Problem (24) can be solved by finding optimal offloading decision variables $A$ and vehicle computing capability $F$. Since $A$ is a discrete variable set while $F$ is a continuous variable set, this problem constitutes a mixed-integer nonlinear programming problem that is NP-hard. We address this problem using a two-step solution approach.

### 3.1 Optimal Vehicle Computing Capability Solution

With offloading decisions $A$ fixed, the only remaining variable in the optimization objective is vehicle computing capability $f_n$, which only affects tasks running locally. The latency and energy consumption for tasks on MEC and cloud servers can be treated as constant $C$. Therefore, the optimization model in problem (24) simplifies to:

$$\begin{aligned} \min_{F} \quad & \sum_{n\in N}\left(\beta_T \frac{C_n}{f_n} + \beta_E \kappa C_n f_n^2\right) \\ \text{s.t.} \quad & C1: 0 \le f_n \le f_n^{\max}, \quad \forall n \in N \\ & C2: \frac{C_n}{f_n} \le \tau_n, \quad \forall n \in N \end{aligned}$$

From these constraints, we derive:

$$\frac{C_n}{\tau_n} \le f_n \le f_n^{\max}$$

Taking the derivative of the objective function with respect to $f_n$ and setting it to zero yields the optimal $f_n^*$:

$$f_n^* = \sqrt[3]{\frac{\beta_T}{2\beta_E \kappa}}$$

Clearly, the objective function increases monotonically on $[0, f_n^*)$ and decreases monotonically on $(f_n^*, \infty)$, reaching its maximum at $f_n^*$. Therefore, without considering constraints, $f_n^*$ is the optimal computing capability. We observe that the optimal vehicle computing capability increases with the delay weight coefficient.

However, due to constraints and the fact that vehicle computing capability may not always meet real-time task requirements, we must discuss different cases:

**a) When $f_n^{\max} \ge \sqrt[3]{\frac{\beta_T}{2\beta_E \kappa}}$:**

- If $\frac{C_n}{\tau_n} \le \sqrt[3]{\frac{\beta_T}{2\beta_E \kappa}}$, then $f_n = \sqrt[3]{\frac{\beta_T}{2\beta_E \kappa}}$ minimizes the objective.
- If $\sqrt[3]{\frac{\beta_T}{2\beta_E \kappa}} < \frac{C_n}{\tau_n} \le f_n^{\max}$, then $f_n = \frac{C_n}{\tau_n}$ minimizes the objective.

**b) When $f_n^{\max} < \sqrt[3]{\frac{\beta_T}{2\beta_E \kappa}}$:**

- If $\frac{C_n}{\tau_n} \le f_n^{\max}$, then $f_n = f_n^{\max}$ minimizes the objective.
- If $f_n^{\max} < \frac{C_n}{\tau_n}$, the problem is infeasible.

### 3.2 DQN-Based Task Offloading Algorithm

This subsection designs a DQN-based algorithm to solve the task offloading decision problem for driverless vehicles. In this scenario, DQN' s three key elements include state, action, and reward function.

**a) State Space:** The state reflects the environment, comprising information about vehicles, tasks, and MEC servers. We denote the entire system state

space as $S = \{s_1, s_2, \ldots, s_N\}$, where $s_n$ is the system state for vehicle $n$, defined as $s_n = \{f_n^{\max}, p_n, D_n, C_n, \tau_n, v_{n,m}, v_{n,c}, B_m, f_m, \text{load}_m\}$. Here, $f_n^{\max}$ and $p_n$ represent the vehicle' s maximum computing capability and transmission power; $D_n, C_n, \tau_n$ represent task data volume, required computing resources, and maximum tolerable delay; $B_m, f_m, \text{load}_m$ represent the MEC server' s uplink bandwidth, computing capability, and current load. Since the MEC server makes offloading decisions sequentially for all tasks and previous offloading choices affect subsequent ones, we include the MEC server' s current load as part of the state.

**b) Action Space:** The action represents the task offloading decision—choosing whether to execute locally, on the MEC server, or on the cloud server. We define the system action space as $A = \{a_1, a_2, \ldots, a_N\}$, where $a_n = \{a_n^l, a_n^m, a_n^c\}$ represents the offloading choices for task $R_n$.

**c) Reward Function:** After executing an action $a_n$, the agent receives reward $r_n$ in the corresponding state $s_n$. Generally, the reward function relates to the optimization objective. Since our goal is to minimize total system cost, we consider making the reward function inversely related to $Z_n$. Additionally, task completion rate is an important performance metric, so we include it in the reward function. When a task fails to complete within its maximum tolerable delay, it should be penalized. The final reward function is defined as:

$$r_n = \begin{cases} -Z_n, & \text{if } T_n \leq \tau_n \\ -Z_n - (\tau_n - T_n), & \text{if } T_n > \tau_n \end{cases}$$

We use mean squared error [**?**] as the loss function and minimize it via gradient descent [**?**]. The loss function is defined as:

$$\text{Loss} = \frac{1}{2} \left( Q_{\text{eva}}(s_n, a_n; \theta_{\text{eva}}) - Q_{\text{tar}} \right)^2$$

where the target Q-value is calculated as:

$$Q_{\text{tar}} = r_n + \gamma \max_{a'} Q_{\text{tar}}(s_{n+1}, a'; \theta_{\text{tar}})$$

**Algorithm 1: DQN Algorithm**

**Input:** Evaluation network $Q_{\text{eva}}$, target network $Q_{\text{tar}}$, experience pool $D$, current state $s_n$
**Output:** Action $a_n$

1. Initialize evaluation network parameters $\theta_{\text{eva}}$ and target network parameters $\theta_{\text{tar}}$
2. Initialize experience pool $D$ with size $N$

3. **for** each episode $k = 1$ to $K$ **do**
4. **for** each step $t$ **do**
5. `Observe current state $s_n$`
6. `Select action $a_n$ using $\epsilon$-greedy policy based on $Q_{\text{eva}}(s_n, a_n; \`
7. `Execute action $a_n$, compute reward $r_n$, and observe next state $s_{n+1}$`
8. `Store transition $(s_n, a_n, r_n, s_{n+1})$ in experience pool $D$`
9. `Sample mini-batch of transitions from $D$`
10. **for** each transition **do**
11. `Calculate target Q-value: $Q_{\text{tar}} = r_n + \gamma \max_{a'} Q_{\text{tar}}(s_{n`
12. **end for**
13. Compute loss: Loss $= \frac{1}{2}\left(Q_{\text{eva}}(s_n, a_n; \theta_{\text{eva}}) - Q_{\text{tar}}\right)^2$
14. Update evaluation network parameters $\theta_{\text{eva}}$ using gradient descent
15. Every $C$ iterations, copy $\theta_{\text{eva}}$ to $\theta_{\text{tar}}$
16. **end for**
17. **end for**

After defining these three key elements, the DQN-based task offloading process in driverless scenarios works as follows: The MEC server acts as the DQN agent, interacting with the environment through states, actions, and rewards. When the agent receives an offloading request from a vehicle, it determines the optimal offloading decision (action) based on the current environmental state and returns the action value to the vehicle. After the vehicle executes the action, the agent receives the corresponding reward.

Traditional Q-learning uses a Q-table to store Q-values for each state-action pair. The agent searches the Q-table to find the action yielding maximum reward in the current state. However, as state and action spaces grow, the memory required for the Q-table increases rapidly, making updates and searches time-consuming. DQN addresses this by using neural networks to estimate Q-values for different state-action combinations. To break data correlation and maintain training stability, DQN incorporates experience replay and a dual-network mechanism. Experience replay stores agent-environment interaction data in experience pool $D$ and randomly samples mini-batches for network updates during training. The dual-network mechanism introduces an evaluation network $Q_{\text{eva}}$ and a target network $Q_{\text{tar}}$ with parameters $\theta_{\text{eva}}$ and $\theta_{\text{tar}}$, respectively. The evaluation network assesses Q-values for current state actions, while the target network computes Q-values for next-state actions. We randomly initialize $\theta_{\text{eva}}$ and $\theta_{\text{tar}}$, create a fixed-capacity experience pool $D$, and for each step in an episode, input state $s_n$ into the evaluation network to obtain action probabilities. The agent selects action $a_n$ using an $\epsilon$-greedy policy, obtains reward

$r_n$, transitions to next state $s_{n+1}$, and stores the experience $(s_n, a_n, r_n, s_{n+1})$ in $D$. Finally, we randomly sample mini-batches from $D$ to update the evaluation network and synchronize its parameters to the target network every $C$ steps.

## 4 Performance Evaluation

This section first briefly describes the experimental platform and dataset. Experiments are implemented on an Intel(R) Core(TM) i5-10210U CPU with 16GB RAM, using Python 3.7 and PyTorch 1.5.0. The map covers a 5km×5km area in Xi' an city, with vehicle trajectories generated by the open-source traffic simulation software SUMO [**?**].

The experiment configures one MEC server with a V2I communication radius of 500 meters. The number of driverless vehicles increases over time, with each vehicle generating at most one task per scheduling period and a task generation probability of 0.6. Tasks have three priority levels, with parameters set according to references [**?**]. For priority levels 1 to 3, task data sizes are randomly selected from [200,300]KB, [400,500]KB, and [700,800]KB; required computing resources from [50,80]M CPU cycles, [150,180]M CPU cycles, and [250,280]M CPU cycles; and maximum tolerable delays of 150ms, 200ms, and 350ms, respectively. The maximum vehicle computing capability is 1.5G CPU cycles/s, with transmission power of 250mW. Channel gains are randomly selected from [1,2]. The MEC server bandwidth is 50MHz with computing capability of 4.5GHz. The cloud server bandwidth is 10MHz with computing capability of 1.5GHz. The MEC server performs task offloading scheduling every 2 seconds.

For algorithm implementation, following reference [**?**], the DQN network architecture and hyperparameters are configured as: 5-layer Q-network with 128 neurons in hidden layers, learning rate of 0.01, discount factor of 0.95, experience pool size of 100, and batch size of 20.

### 4.2 Comparative Experiments

We compare the DQN algorithm against three commonly used task offloading algorithms: all-local, all-MEC, and random, using total cost (weighted sum of latency and energy), latency, energy consumption, and completion rate as evaluation metrics.

**a) All-local execution:** All tasks execute locally on the driverless vehicle.
**b) All-MEC execution:** All tasks are offloaded to the MEC server for execution.
**c) Random offloading:** Tasks randomly select execution location among local, MEC server, or cloud server.

### 4.3 Simulation Results

Figure 2 shows the convergence curve of cumulative average reward over 18,500 scheduling iterations. The DQN-based task offloading algorithm achieves stable

convergence after sufficient training steps. This occurs because DQN thoroughly explores the environment through multiple interactions and learns suitable offloading strategies. Additionally, experience replay effectively reduces data correlation and improves algorithm stability, facilitating convergence.

Figure 3 compares DQN against the three baseline algorithms under varying task loads by changing the task generation probability per vehicle from 0.3 to 0.9. Except for all-local execution, the total cost, latency, and energy consumption of the other three algorithms increase with task load, while completion rate decreases. All-local execution remains nearly constant as it is independent of offloading probability. DQN consistently achieves the lowest total cost, latency, and energy consumption among all algorithms, with completion rates comparable to all-local execution.

Figure 4 compares the four algorithms under different MEC server computing capabilities. All-local execution maintains constant performance across all metrics since MEC server capability does not affect vehicle computing. As MEC server computing capability increases, the total cost, latency, and completion rate of the other three algorithms improve (cost and latency decrease, completion rate increases), while energy consumption remains unaffected, showing a horizontal line. DQN consistently delivers the lowest total cost, latency, and energy consumption, with competitive completion rates.

Figure 5 examines the impact of energy consumption weight coefficients. DQN achieves the lowest total cost across all weight configurations. As the energy weight coefficient increases, the latency of DQN, random, and all-local algorithms gradually increases while energy consumption decreases, indicating that DVFS technology tends to reduce vehicle computing capability to save energy. All-MEC execution shows horizontal lines for latency, energy, and completion rate as it is unaffected by vehicle computing capability. Notably, all-local execution' s completion rate decreases significantly with larger energy weights because vehicle computing capability weakens as the energy weight increases, causing some tasks to exceed their maximum tolerable delays.

## 5 Conclusion

This paper proposes a driverless task offloading model based on a vehicle-edge-cloud three-layer network architecture, considering vehicle computing capability optimization, mobility issues, and task priority classification. The model jointly optimizes vehicle computing capability and task offloading strategy to minimize system latency and energy consumption. As this is a mixed-integer nonlinear programming problem, we employ a two-step solution to reduce computational complexity: first deriving the optimal vehicle computing capability analytically, then using DQN to obtain optimal offloading decisions. Extensive simulation experiments validate the feasibility and superiority of the proposed strategy. Future work will consider handover issues between adjacent RSUs and integrate caching technology into the task offloading architecture to further reduce task

processing delay and computational energy consumption.

## References

[1] Hee L G, Faundorfer F, Pollefeys M. Motion estimation for self-driving cars with a generalized camera [C]// Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2013: 2746-2753.

[2] Ackerman E. Lidar that will make self-driving cars affordable [News] [J]. IEEE Spectrum, 2016, 53 (10): 14-14.

[3] Ning Z, Xia F, Ullah N, et al. Vehicular social networks: enabling smart mobility [J]. IEEE Communications Magazine, 2017, 55 (5): 16-55.

[4] Chen L, Englund C. Cooperative intersection management: A survey [J]. IEEE Trans on Intelligent Transportation Systems, 2015, 17 (2): 570-586.

[5] Zhang W, Zhang Z, Chao H C. Cooperative fog computing for dealing with big data in the internet of vehicles: architecture and hierarchical resource management [J]. IEEE Communications Magazine, 2017, 55 (12): 60-67.

[6] Weisong S, Hui S, Jie C, et al. Edge computing—An emerging computing model for the Internet of everything era [J]. Journal of Computer Research and Development, 2017, 54 (5): 907.

[7] Lyu Pin, Xu Jia, Li Taoshen, et al. A review of edge computing technology research for autonomous driving [J]. Journal on Communications, 2021, 42 (3): 190-208.

[8] Wang Q, Mao Y, Wang Y, et al. Computation tasks offloading scheme based on multi-cloudlet collaboration for edge computing [C]// the Seventh International Conference on Advanced Cloud and Big Data (CBD). IEEE, 2019: 339-344.

[9] Zhao Haitao, Zhang Tangwei, Chen Yue, et al. DQN-based task distribution and offloading algorithm for in-vehicle edge network [J]. Journal on Communications, 2020, 41 (10): 172-178.

[10] Liu Guozhi, Dai Fei, Mo Qi, et al. Service offloading method based on deep reinforcement learning in vehicle edge computing environment [J/OL]. Computer Integrated Manufacturing Systems: 1-16. (2021-10-29) [2022-03-18].

[11] Tang J, Liu S, Liu L, et al. Lopecs: A low-power edge computing system for real-time autonomous driving services [J]. IEEE Access, 2020, 8: 169010-169023.

[12] Lin C C, Deng D J, Yao C C. Resource allocation in vehicular cloud computing systems with heterogeneous vehicles and roadside units [J]. IEEE Internet of Things Journal, 2017, 5 (5): 3692-3700.

[13] Belogaev A, Elokhin A, Krasilov A, et al. Cost-effective V2X task offloading in MEC-assisted intelligent transportation systems [J]. IEEE Access, 2020, 8: 169010-169023.

[14] Huang M, Liu W, Wang T, et al. A cloud–MEC collaborative task offloading scheme with service orchestration [J]. IEEE Internet of Things Journal, 2019, 7 (7): 5792-5805.

[15] Deng Shiquan, Ye Xuguo. Multi-objective task offloading algorithm based on deep Q network [J/OL]. Journal of Computer Applications: 1-9. (2022-02-25) [2022-03-18].

[16] Sun Y, Xu L, Tang Y, et al. Traffic offloading for online video service in vehicular networks: A cooperative approach [J]. IEEE Trans on Vehicular Technology, 2018, 67 (8): 7630-7642.

[17] Qu G. What is the limit of energy saving by dynamic voltage scaling? [C]// IEEE/ACM International Conference on Computer Aided Design. ICCAD 2001. IEEE/ACM Digest of Technical Papers (Cat. No.01CH37281). IEEE, 2001: 560-563.

[18] Miettinen A P, Nurminen J K. Energy efficiency of mobile clients in cloud computing [C]// 2nd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 10). 2010.

[19] Chen X, Jiao L, Li W, et al. Efficient multi-user computation offloading for mobile-edge cloud computing [J]. IEEE/ACM Trans on Networking, 2015, 24 (5): 2795-2808.

[20] Elgendy I A, Zhang W, Tian Y C, et al. Resource allocation and computation offloading with data security for mobile edge computing [J]. Future Generation Computer Systems, 2019, 100: 531-541.

[21] Dai H, Zeng X, Yu Z, et al. A scheduling algorithm for autonomous driving tasks on mobile edge computing servers [J]. Journal of Systems Architecture, 2019, 94: 14-23.

[22] Chen L, Qu H, Zhao J, et al. Efficient and robust deep learning with correntropy-induced function [J]. Neural Computing and Applications, 2016, 27 (4): 1019-1031.

[23] Ruder S. An overview of gradient descent optimization algorithms [J]. arXiv preprint arXiv:1609.04747, 2016.

[24] Lopez P A, Behrisch M, Bieker-Walz L, et al. Microscopic traffic simulation using sumo [C]// 21st International Conference on Intelligent Transportation Systems (ITSC). IEEE, 2018: 2575-2582.

[25] Hao Z, Sun Y, Li Q, et al. Delay-energy efficient computation offloading and resources allocation in heterogeneous network [C]// IEEE Global Communications Conference (GLOBECOM). IEEE, 2019: 1-6.

[26] Tran T X, Pompili D. Joint task offloading and resource allocation for multi-server mobile-edge computing networks [J]. IEEE Trans on Vehicular Technology, 2018, 68 (1): 856-868.

[27] Zhan W, Duan H, Zhu Q. Multi-user offloading and resource allocation for vehicular multi-access edge computing [C]// IEEE International Conferences on Ubiquitous Computing & Communications (IUCC) and Data Science and Computational Intelligence (DSCI) and Smart Computing, Networking and Services (SmartCNS). IEEE, 2019: 50-57.

[28] Elgendy I A, Zhang W Z, He H, et al. Joint computation offloading and task caching for multi-user and multi-task MEC systems: reinforcement learning-based algorithms [J]. Wireless Networks, 2021, 27 (3): 2023-2036.

*Note: Figure translations are in progress. See original paper for figures.*