Journal of Cloud Computing:
Advances, Systems and Applications

RESEARCH

Open Access

A deep reinforcement learning assisted task offloading and resource allocation approach towards self-driving object detection

Lili Nie[1], Huiqiang Wang[1*], Guangsheng Feng[1], Jiayu Sun[1], Hongwu Lv[1] and Hang Cui[2]

Abstract

With the development of communication technology and mobile edge computing (MEC), self-driving has received more and more research interests. However, most object detection tasks for self-driving vehicles are still performed at vehicle terminals, which often requires a trade-off between detection accuracy and speed. To achieve efficient object detection without sacrificing accuracy, we propose an end–edge collaboration object detection approach based on Deep Reinforcement Learning (DRL) with a task prioritization mechanism. We use a time utility function to measure the efficiency of object detection task and aim to provide an online approach to maximize the average sum of the time utilities in all slots. Since this is an NP-hard mixed-integer nonlinear programming (MINLP) problem, we propose an online approach for task offloading and resource allocation based on Deep Reinforcement learning and Piecewise Linearization (DRPL). A deep neural network (DNN) is implemented as a flexible solution for learning offloading strategies based on road traffic conditions and wireless network environment, which can significantly reduce computational complexity. In addition, to accelerate DRPL network convergence, DNN outputs are grouped by in-vehicle cameras to form offloading strategies via permutation. Numerical results show that the DRPL scheme is at least 10% more effective and superior in terms of time utility compared to several representative offloading schemes for various vehicle local computing resource scenarios.

Keywords Mobile edge computing, Object detection, Deep reinforcement learning, Task offloading

Introduction

Thanks to the explosive growth of MEC, self-driving technology has undergone significant development. As an important component in self-driving vehicles, object detection has been widely used to help self-driving vehicles detect surrounding objects, such as other vehicles, pedestrians, traffic signs, and lanes.

To improve the detection accuracy, there is a major trend to building convolutional neural networks (CNNs) with deeper layers and more complex structures. For example, networks such as AlexNet [1], visual geometry group (VGG) [2], deep residual network (ResNet) [3], densely connected convolutional network (DenseNet) [4] and ResNeXt [5] have been widely used in tasks such as image classification [6], object detection [7, 8] and semantic segmentation [9]. Although the accuracy of these networks has been improved, their depths also increased significantly. Well-trained network models typically have tens of millions of weight hyperparameters, which can result in heavy demands on computing resources. In general, object detection tasks of self-driving vehicles have strict latency constraints and inference accuracy requirements. Hence it is challenging for resource-constrained vehicle terminals to perform such computationally-intensive tasks.

*Correspondence:
Huiqiang Wang
wanghuiqiang@hrbeu.edu.cn
[1] College of Computer Science and Technology, Harbin Engineering University, Harbin 150001, Heilongjiang, China
[2] TungThih Electronic company, Xiamen 361006, Fujian, China

Benefiting from MEC technology, vehicle-to-everything (V2X) cellular telematics are growing exponentially. V2X aims to enable vehicle-to-vehicle (V2V) [10, 11], vehicle-to-infrastructure (V2I) [12], and vehicle-to-network (V2N) [13] communications to support the efficient processing of terminal tasks by offloading all or part of them to the surrounding infrastructures. Edge devices can provide assistance for self-driving vehicles in executing object detection tasks, for example, by using a novel context-aware method [14] to accelerate the object detection speed or by extracting and compressing some regions of interest [15] to be sent to the edge cloud. These methods transfer the object detection tasks to edge clouds or cloud centers, which can effectively alleviate local computational pressures; however, these methods excessively rely on edge servers, which are prone to network congestion under the influx of a large number of tasks. Once the wireless network state deteriorates, it is difficult to guarantee task execution efficiency. In addition, compressing image to ensure that the detection results are transmitted back within a specified time, which will inevitably lead to a loss in detection accuracy.

To improve the efficiency of object detection while ensuring accuracy, we need to work out a more intelligent end–edge collaboration approach to cope with the time-varying wireless environment and complex traffic conditions. In this paper, we propose an end–edge collaboration object detection approach based on DRL to generate task offloading and local computing resource allocation strategies. According to time–varying wireless network environment and road traffic conditions, the approach can maximize the average sum of the time utilities for each object detection task in all slots. The main contributions of this work are summarized as follows:

- To achieve efficient object detection tasks for self-driving vehicles without sacrificing accuracy, we formulate a mixed integer non-linear programming (MINLP) problem to jointly optimize the task offloading and local computing resource allocation strategies. Specifically, detection tasks are offloaded to edge servers and completed with the maximize the average sum of the time utilities without losing accuracy. As far as we are concerned, previous studies only focus on one of these aspects.
- We propose an online approach based Deep Reinforcement learning and Piecewise Linearization (DRPL) to solve the MINLP problem mentioned above. In this approach, the MINLP problem is decomposed into an offloading strategy subproblem and a resource allocation subproblem.
- We develop a prioritization mechanism in accordance with vehicle navigation commands and historical object detection results to adapt to complex road traffic environment. In addition, to speed up the DRPL algorithm, we group the deep neural network (DNN) outputs by cameras and expand them to form candidate offloading strategies via permutation.

The remaining parts of this paper are organized as follows. In Related work section, we review the related work. In System model and problem formalization section, we describe the system model and formalize the problem. In The DRPL algorithm section, we present the detailed design of the DRPL algorithm. In Numerical results analysis section, we report numerical results. Finally, we conclude the paper.

Related work

Edge computing technology can provide services as close as possible to the device or data source by means of an open platform that integrates core networking, computing, storage, and application capabilities. Such technology can reduce the energy and resource consumption of terminal devices while responding quickly to terminal requests and meeting real-time needs.

Edge computing technology

Edge computing technology is of great importance and has attracted extensive research attention. Some recent research works have focused on different application scenarios. For example, in terms of unmanned aerial vehicles (UAVs), Nan et al. [16] studied the problem of joint task offloading and resource allocation for vehicular edge computing with result feedback delay. Gao et al. [17] investigated the problem of joint task offloading, task scheduling, and resource allocation in vehicle edge computing, and the fast changing channel between a vehicle and an edge server to minimize the delay and energy consumption of vehicular edge computing. Deng et al. [18] took DNN as the typical AI application and formulated an optimization problem that optimizes the DNN model decision, computation, communication resource allocation, and UAV trajectory control. Zhou et al. [19] proposed a gradient-based dynamic iterative search algorithm to obtain the approximate optimal solution. In terms of wireless powered mobile edge computing, Mao et al. [20] investigated the fundamental tradeoff between energy efficiency and delay in a multi-user wireless powered MEC system. They filled the gap by jointly scheduling energy, radio, and computational resources to coordinate heterogeneous performance requirements in wireless powered MEC systems. Chen et al. [21] presented an augmented two-staged deep Q-network for online optimization of wireless power transfer MEC systems to minimize the long-term average energy

requirement of the systems. Deng et al. [22] proposed a dynamic throughput maximum algorithm based on perturbed Lyapunov optimization to maximize the system throughput under task and energy queue stability constraints.

While some research work has focused on the design of network resource scheduling or computation offloading algorithms with various optimization objectives. Mao et al. [23] proposed utilizing the intelligent reflecting surfaces technique to improve the efficiency of wireless energy transfer and task offloading in order to achieve a higher total computation rate. Shnaiwer et al. [24] designed new methods for jointly optimizing the reflection coefficients of intelligent reflecting surfaces and path selection. They presented a general mathematical formulation for the problem of minimizing the total energy consumption of the system. Song et al. [25] proposed a computation offloading scheme and a dynamic road network state update model for proximity detection in dynamic road networks, aiming to efficiently reduce the computational time of the optimal latency each time. Zhou et al. [26] proposed a novel deep reinforcement learning-based computation offloading and service caching mechanism to jointly optimize the offloading decision, service caching, and resource allocation strategies. The aim is to minimize the cost while ensuring the delay requirements of mobile users.

However, the aforementioned studies are lacking in the context of self-driving object detection, despite the wide range of application scenarios and the variety of target problems being addressed.

Edge computing technology-based visual object detection methods for self-driving vehicles

The boom in edge computing has simultaneously led to significant growth in self-driving technology [27]. Recently, researchers have started to investigate edge computing methods to assist in self-driving object detection. Guo et al. [14] collected contextual information (weather, time, traffic, etc.) from the current road environment and combined these contextual features with the visual features of images on the MEC server. Kim et al. [15] deployed object detection networks on an edge server. When the channel quality was not sufficient to support real-time object detection, the self-driving vehicles compressed the image data based on the regions of interest and transmitted the compressed data to the edge cloud. However, the above mentioned studies of edge-computing-assisted object detection for self-driving vehicles have certain limitations: the self-driving vehicles rely too much on the edge servers, ignoring the time-varying wireless transmission environment; moreover, compressing images to speed up object detection may lead to loss of key traffic information and affect the object detection accuracy. Hence, developing an efficient, accurate and intelligent object detection approach is still an open problem.

DRL-based task offloading methods

The DRL algorithm observes the surroundings in real time and relies on deep neural networks (DNNs) to learn from the training data samples. It eventually produces the optimal mapping from the time-varying state space [28] to the action space [29]. A number of works have recently begun to investigate how to use DRL to make task offloading strategies for mobile terminals. To cope with the joint optimization problem of computation offloading and resource allocation in MEC, Chen et al. [30] proposed a temporal attentional deterministic policy gradient based on deep Q-network (DQN). Aiming at trust issues for service migration in vehicular edge computing, Ren et al. [31] designed a dynamic service offloading and migration algorithm based on A3C. To ensure the quality of internet of vehicles services, Hazarika et al. [32] proposed a priority-sensitive task offloading and resource allocation scheme based on deep deterministic policy gradient (DDPG) and twin delayed DDPG algorithms. However, in our scenario, if we choose DQN-based networks, we may suffer from slow convergence when we take the time–varying wireless channel gains and traffic conditions as the input state vector. Besides, because of its exhaustive search nature in selecting the action in each iteration, DQN is not suitable for handling problems with high-dimensional action spaces [33].

In this paper, based on deep reinforcement learning and piecewise linearization, we propose an end–edge collaboration object detection approach for self-driving vehicles, which can maximize the average sum of the time utilities for each object detection task in all slots. Moreover, to speed up our netwok, inspired by [34], we group the deep neural network (DNN) outputs by cameras and expand them to form candidate offloading strategies via permutation.

System model and problem formalization

As shown in Fig. 1, we consider a visual object detection problem for one self-driving vehicle with I cameras, denoted by $\mathcal{I} = \{1, 2, 3, \ldots, I\}$. $I + 1$ well-trained CNNs with the same structure are embedded, one in each camera in $\mathcal{I}$ and one in an edge server. At the same time, the vehicle terminal is equipped with a driving control system (DCS), which is responsible for collecting wireless environment information and vehicle navigation command in each time slot $j = \{1, 2, 3, \ldots, N\}$, and generating corresponding task execution priorities in accordance with the object detection results of each camera in time

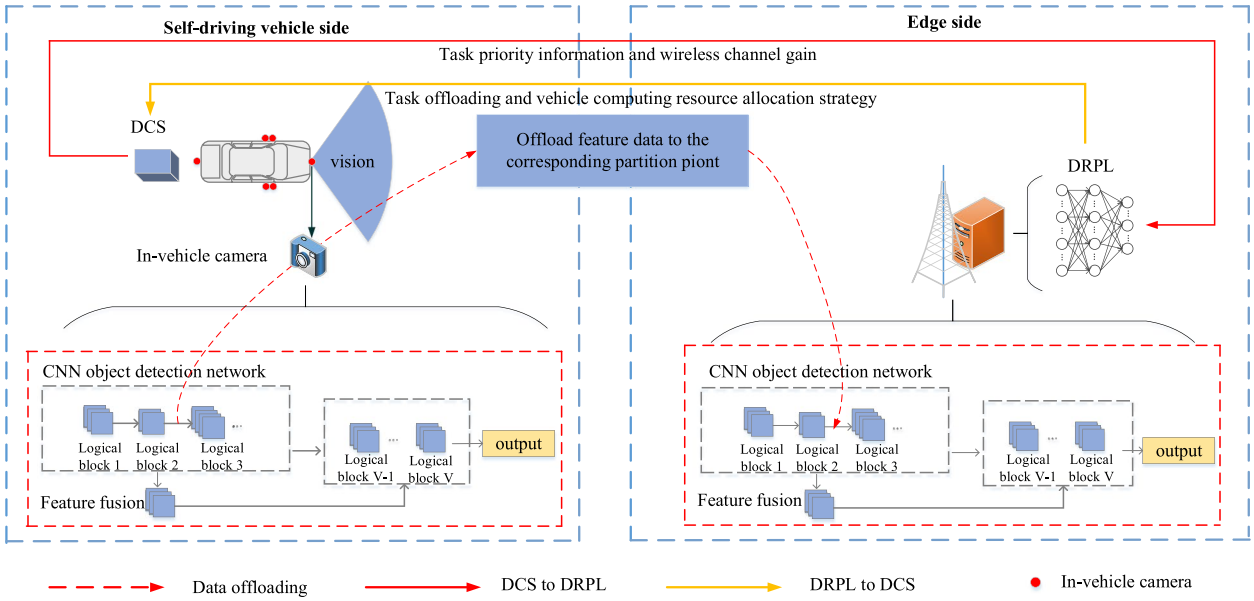


Fig. 1 An offloading approach for object detection tasks based on end–edge collaboration

slot. Moreover, the DCS transmits the priority information and wireless channel gain to DRPL on the edge side, and DRPL decides whether to execute locally or offload the output of logical block in the detection network of in-vehicle camera i, to edge side as the input to logical block. At the same time, DRPL allocates the local computing resources of the vehicle terminal. Here we consider that the self-driving vehicle driving within the communication range of the edge server at all time slots. The notations we adopt are summarized in Table 1.

The priority of each in-vehicle camera

The traffic conditions of self-driving vehicles are complex and rapidly changing, unexpected statuses may occur at any time and place. Moreover, the channel state between a vehicle terminal and the edge server also changes with variations in the transmission medium. Therefore, for a vehicle terminal with limited computing resources, efficient and accurate detection of the surrounding environment is a great challenge. In this section, we develop a prioritization mechanism for each object detection task in accordance with the navigation commands in time slot j and the object detection results in time slot $j-1$.

The impact of vehicle navigation commands on the execution priority of each task for every in-vehicle camera

Different cameras in different parts of the self-driving vehicle are mainly responsible for monitoring different ranges. For example, suppose that in a certain time slot, the navigation command is to proceed directly ahead; then, the camera mainly responsible for monitoring the road environment in front of the self-driving vehicle (such as detecting the road, other surrounding, other vehicles, pedestrians and other targets) will be mobilized first, and its priority will be higher than that of the rest of the cameras.

The impact of object detection results on the execution priority of each task for every in-vehicle camera

The self-driving traffic environment is ever-changing and unexpected conditions may occur at any time, so the vehicle needs to detect the surrounding environment always and make emergency operation in time. Therefore, the priority of each camera should be determined not only taking into account the actual vehicle navigation commands, but also the road conditions.

We suppose that in time slot j, X_i^j objects are detected in the visual range of camera i. For each object x, $x \in X_i^j$, its features can be quantified as a six-tuple $Z_{i,x}^j = \left\{ Y_{i,x}^j, R_{i,x}^j, P_{i,x,y}^j, \overline{P}_{i,x,y}^j, A_{i,x}^{j,y}, \epsilon_{i,x}^{j,y} \right\}$, where $Y_{i,x}^j$ denotes the category of result $Z_{i,x}^j$, $R_{i,x}^j$ denotes the detection frame size of result $Z_{i,x}^j$, $P_{i,x,y}^j$ denotes the probability that result $Z_{i,x}^j$ belongs to category y when the object detection algorithm achieves correct detection, $\overline{P}_{i,x,y}^j$ denotes the probability that result $Z_{i,x}^j$ belongs to category y when the algorithm suffers from detection error, $A_{i,x}^{j,y}$ denotes the score for result $Z_{i,x}^j$ corresponding to a dangerous object when it is determined to belong to category y, and $\epsilon_{i,x}^{j,y}$ denotes the threshold for result $Z_{i,x}^j$ corresponding to a dangerous object when it is determined to belong to category y.

Table 1 Notations

Notation	Meaning
	The object detection task of camera i in time slot j
	The local computation size of task before offloading partition point v
	The offload data size of task at partition point v
	The delay tolerance of high-priority tasks
τ_L	The delay tolerance of low-priority tasks
O_i^j	The priority of task S_i^j
V	The collection of alternative offloading partition points
X_i^j	The number of objects included in the detection results of task
	The detection result x of task S_i^j
$Y_{i,x}^j$	The category of result $Z_{i,x}^j$
$R_{i,x}^j$	The detection frame size of result $Z_{i,x}^j$
$P_{i,x,y}^j$	The probability that result $Z_{i,x}^j$ belongs to category y
$A_{i,x}^{j,y}$	The score for result $Z_{i,x}^j$ corresponding to a dangerous object when it is determined to belong to category y
	The threshold for result $Z_{i,x}^j$ corresponding to a dangerous object when it is determined to belong to category y
f_i^j	The proportion of the local computing resources assigned to task S_i^j
r_i^j	The data rate of transmitting task S_i^j to edge server
h_j	The channel gain in time slot j
$t_{i,l}^{j,v}$	The time cost of locally processing task S_i^j before partition point v
$t_{i,up}^{j,v}$	The time cost of offloading the feature data for task S_i^j at partition point v
$t_i^{j,v}$	The total time cost for task S_i^j when offloading partition point v is selected
$u_{i,v}^{j,H}$	The time utility for high-priority task S_i^j when offloading partition
	point v is selected
$u_{i,v}^{j,L}$	The time utility for low-priority task S_i^j when offloading partition
	point v is selected
u_i^j	The time utility for a task S_i^j of a certain priority in any time slot j

In general, we can assume that the larger the detection frame size of the detected object, the higher danger level of it. However, in some complex and variable traffic environments (e.g., weather, light, and shading), even a well-trained object detection algorithm will inevitably produce detection errors. Figure 2 shows examples of object detection results against simple and complex backgrounds. The range of a detected object is represented by an orange rectangular box, and the probability value of the the detected object belonging to a certain category is shown above the rectangular box. Figure 2(a) shows detection result against a simple background. Since the object in this figure is clear and there is no interference from other factors, the precision is credible, and the framing of the object detection result is accurate. In contrast, in the scene with a complex background shown in Fig. 2(b), the black vehicle in the shadow of the sun is similar in color to its surroundings, while the vehicle in direct sunlight has characteristics similar to those of the white wall in its vicinity; in both cases, these similarities lead to detection errors. If we were to use the detection frame size as the only indicator to determine the danger level of an object, then the priorities of some tasks would be mismatched. So, we need to combine the detection frame size and the detection accuracy to make a joint judgment on the danger level of each object. In this paper, we propose to use the product of the detection frame size and the detection accuracy to express the danger level of an object, which is calculated as shown in Eq. (1).

$$A_{i,x}^{j,y} = E \times R_{i,x}^j \times \overline{P}_{i,x,y}^j + (1-E) \times R_{i,x}^j \times P_{i,x,y}^j, \quad (1)$$

where, considering the complexity of the traffic conditions, E denotes the potential for the object detection algorithm to suffer from detection error. Referring back to the above, we can see that $P_{i,x,y}^j$ denotes the probability that result $Z_{i,x}^j$ belongs to category y when the object detection algorithm achieves correct detection, $\overline{P}_{i,x,y}^j$ denotes the probability that result $Z_{i,x}^j$ belongs to category y when the algorithm suffers from detection error.

In any time slot j, if there is at least one object detection result for the camera i with a danger value larger than its danger threshold, i.e., $A_{i,x}^j \geq \epsilon_{i,x}^{j,y}, \exists x \in X_i^j$, then the priority of the in-vehicle camera i is set to high. If there is no dangerous object is detected, i.e., $A_{i,x}^j < \epsilon_{i,x}^{j,y}, \forall x \in X_i^j$, the priority of each detection task is determined in accordance with the navigation

(a) A simple background (b) A complex background

Fig. 2 Detection accuracy against different backgrounds

command. The priority values for tasks determined to be of high priority are set to 1, and the rest are set to 0.

In summary, the rules for adjusting the priorities of object detection tasks belonging to each in-vehicle camera are as follows: if no dangerous object is detected, the priority of each detection task is determined in accordance with the navigation command; if any camera detects at least one dangerous object, the priority of that camera is set to high, and the priorities of the other object detection tasks of the remaining cameras are still determined in accordance with the navigation command. The details of the prioritization algorithm are given in Algorithm 1.

if $j = 1$ **then**
  Start to perform the object detection algorithm in each in-vehicle camera, and set $O_i^j = 0$, $\mathcal{I} = \{1, 2, 3, \ldots, I\}$
end if
if $j >= 2$ **then**
  for $j = 2$ to N **do**
    for $i = 1$ to I **do**
      In accordance with the car navigation commands, set $O_i^j = O_{i,order}^j$
    end for
    for $i = 1$ to I **do**
      for $x = 0$ to X_i^j **do**
        Calculate the danger value for each rectangular box, $A_{i,x}^{j,y} = E \times R_{i,x}^j \times \overline{P}_{i,x,y}^j + (1 - E) \times R_{i,x}^j \times P_{i,x,y}^j$
      end for
      if there is at least one object detection result for the camera with a danger value larger than its danger threshold, i.e., $A_{i,x}^j \geq \epsilon_{i,x}^{j,y}, \exists x \in X_i^j$ **then**
        $O_i^j = 1$
      end if
      if the danger values of all object detection results for the current camera are smaller than its danger threshold, i.e.,$A_{i,x}^j < \epsilon_{i,x}^{j,y}, \forall x \in X_i^j$ **then**
        $O_i^j = O_{i,order}^j$
      end if
    end for
  end for
end if

Algorithm 1 The priority determination algorithm for the in-vehicle camerasThe task execution time utility model

In time slot j, the features of the object detection task of camera i can be represented by a four-tuple $S_i^j\{C_{i,v}^j, M_{i,v}^j, \tau_i^j, O_i^j\}$, where S_i^j denotes the object detection task of in-vehicle camera i in time slot j, $C_{i,v}^j$ denotes the local computation size of task S_i^j before offloading partition point v, $M_{i,v}^j$ denotes the data size of task S_i^j at offloading partition point v, τ_i^j denotes the delay tolerance of task S_i^j, and O_i^j denotes the priority of task S_i^j. As the environment between the self-driving vehicle and the edge server changes, the wireless channel conditions change accordingly. If the wireless link is available, the self-driving vehicle can choose to offload object detection tasks to the edge server and can also receive the results from the edge server via the wireless link. Otherwise, for example, when the wireless channel suffers from deep fading, all object detection tasks must be executed locally. Here, we suppose that the computational power of the edge server is much stronger than the self-driving vehicle, so we set the execution time on the edge server as a constant Υ, and the time utility functions for tasks with different priorities are shown as follows.

Local computing

We use $t_{i,l}^{j,v}$ to denote the local computing time of an object detection task before offloading partition point v, which can be calculated as shown in Eq. (2).

$$t_{i,l}^{j,v} = \frac{C_{i,v}^j}{f_i^j \times F_L}, \quad \forall i \in I, j \in N, v \in V, \tag{2}$$

where F_L denotes the computational resources of the vehicle terminal and f_i^j denotes the proportion of the computational resources allocated to in-vehicle camera i in time slot j.

Edge computing

We use $t_{i,up}^{j,v}$ to denote the time to transmit the feature data to the edge server at offloading partition point v. Here, we assume that the bandwidth is sufficient and

there is no need to allocate bandwidth among the in-vehicle cameras. $t_{i,up}^{j,v}$ and data transmission rate r_i^j can be calculated as shown in Eqs. (3) and (4) respectively.

$$t_{i,up}^{j,v} = \frac{M_{i,v}^j}{r_i^j}, \quad \forall i \in I, j \in N, v \in V, \tag{3}$$

$$r_i^j = B \times \log_2(1 + \frac{P \times h_j}{N_0}), \quad \forall i \in I, j \in N, \tag{4}$$

where h_j denotes the channel gain in time slot j.

The total time for the object detection task S_i^j can be calculated as shown in Eq. (5).

$$t_i^{j,v} = t_{i,l}^{j,v} + t_{i,up}^{j,v} + \Upsilon, \quad \forall i \in I, j \in N, v \in V. \tag{5}$$

Here, thanks to the ultra-low transmission latency in the context of 5G, we do not focus on the delay caused by data transmission failure. At the same time, we ignore the transmission time of the computation results from the edge server back to the vehicle terminal.

Time utility computation

The completion times of different-priority tasks have different impacts on the self-driving vehicle. Here, we set different time utility functions for different-priority tasks [35], which can be calculated as shown in Eqs. (6) and (7).

The time utility function of high-priority tasks can be written as follow:

$$u_{i,v}^{j,H} = \begin{cases} \log_2(1 + \tau_H - \frac{C_{i,v}^j}{f_j^j \times F_L} - \frac{M_{i,v}^j}{B \times \log_2(1 + \frac{P \times h_j}{N_0})}) \\ \qquad\qquad t_i^{j,v} \le \tau_H, \\ -\Upsilon^H \qquad\qquad otherwise. \end{cases} \tag{6}$$

The time utility function of low-priority tasks can be written as follow:

$$u_{i,v}^{j,L} = \begin{cases} \Upsilon^L & t_i^j \le \tau_L, \\ \Upsilon^L \times e^{-c(\frac{C_{i,v}^j}{f_i^j \times F_L} + \frac{M_{i,v}^j}{B \times \log_2(1 + \frac{P \times h_j}{N_0})} - \tau_L)} & otherwise, \end{cases} \tag{7}$$

where τ_H and τ_L denote the delayed tolerance of high-priority and low-priority tasks.

Problem formulation

According to the tasks priorities, the average sum of the time utilities u_i^j in a given time frame j is expressed as shown in Eq. (13).

$$\begin{aligned} Q(h_j, O_i^j, f_i^j, a_i^j) &= \frac{1}{N} \times \sum_{j=1}^{N} \sum_{i=1}^{I} u_i^j \\ &= \frac{1}{N} \times \sum_{j=1}^{N} \sum_{i=1}^{I} O_i^j \times u_{i,v}^{j,H} + (1 - O_i^j) \times u_{i,v}^{j,L}. \end{aligned} \tag{8}$$

Here, we set $O_i^j \in \{0, 1\}$, where $O_i^j = 1$ denotes high priority, $O_i^j = 0$ denotes low priority, and a_i^j denotes the offloading partition point for the object detection task of camera i in time slot j. In each time slot, we aim to maximize the average sum of the time utilities for each in-vehicle camera under a given channel gain and tasks priorities. The specific calculation are shown as follow:

$$(Q1) : \hat{Q}(h_i^j, O_i^j) = \max_{f_i^j, a_i^j} Q(h_j, O_i^j, f_i^j, a_i^j) \tag{9}$$

$$s.t. \quad \sum_{i=1}^{I} f_i^j \le 1, \quad j \in \{1, 2, 3, \ldots, N\}, \tag{10}$$

$$0 \le f_i^j \le 1, \tag{11}$$

$$a_i^j \in \{0, 1, 2, \ldots, V\}. \tag{12}$$

In the set of constraints, constraint (10) guarantees that the sum of the proportions of vehicle terminal computing resources allocated to each in-vehicle camera does not exceed 1. Constraints (11) ensures that the proportion of vehicle terminal computational resources allocated to any in-vehicle camera lies between 0 and 1. Constraints (12) makes sure that the offloading partition point for the object detection task of camera i in time slot j, i.e. a_i^j, does not exceed the predetermined set of offloading partition points V for each in-vehicle camera.

Lemma 1 (Q1) *is NP-hard.*

Proof

We prove its NP-hardness by transforming the simplified form of (Q1) into an NP problem. □

Step 1: We first simplify the objective function as a linear function expressed as a closed form of f_i^j and a_i^j, i.e.:

$$\begin{aligned} Q_i^j(\omega_i^j, f_i^j, a_i^j) &= F_i^j(\omega_i^j, f_i^j, a_i^j) \\ &= \omega_i^j \times a_i^j \times g(f_i^j), \end{aligned} \tag{13}$$

where, $g(f_i^j)$ is an operation on f_i^j, which we assume is known.

Step 2: The original question Q_i^j is transformed as Q1'.

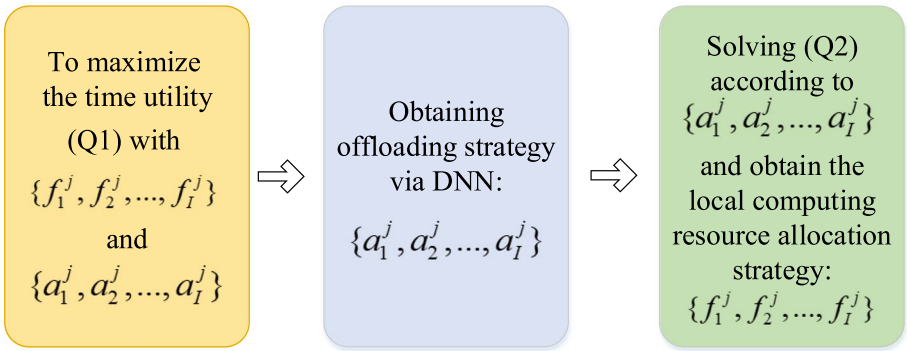


Fig. 3 The two-step optimization structure for solving (Q1)

$$Q1' : max \sum_{i\in I, j\in N} \omega_i^j \times a_i^j \times g(f_i^j) \tag{14}$$

$$s.t. \quad (10), (11), (12). \tag{15}$$

It can be seen that Q1' is a 0-1 backpack problem, which is a well-known NP problem. Since the simplified form of (Q1) is NP-hard, we can infer that (Q1) is NP-hard.

Problem (Q1) is a MINLP problem and is NP-hard [35–37]. However, once a_i^j is determined, the number of unknowns decreases, and (Q1) becomes solvable. When a_i^j is determined, problem (Q1) can be transformed into problem (Q2):

$$(Q2) : \hat{Q}(h_i^j, O_i^j, a_i^j) = \max_{f_i^j} Q(h_j, O_i^j, f_i^j, a_i^j) \tag{16}$$

$$s.t. \quad (10), (11). \tag{17}$$

Thus, problem (Q1) can be decomposed into two subproblems, i.e., the task offloading strategy problem and the resource allocation strategy problem (Q2), as shown in Fig. 3.

- Task offloading strategy: Intuitively, we need to search V^I possible offloading strategies to find a satisfactory one. However, due to the exponential growth of the search space, this method takes a long time to converge. We propose using a deep reinforcement learning based approach to assist in finding a reasonable offloading strategy.
- Resource allocation strategy: In problem (Q2), the optimal allocation of the local computing resources is still a nonlinear programming problem. Therefore, we need to use the piecewise linearization method (PLM) to transform this problem into a linear programming problem and find its approximate solution.

The main difficulty of solving problem (Q1) is handling the offloading strategy problem. Traditional optimization algorithms need to adjust the offloading strategy through multiple iterations, during which, the wireless environment and the road traffic are rapidly changing. It is very difficult to handle object detection tasks efficiently using such algorithms. To address the complexity problem, we propose a novel deep reinforcement learning based online offloading algorithm, DRPL, which can adapt well to time-varying environmental information to find a satisfactory offloading strategy.

The DRPL algorithm

As mentioned in the previous section, to obtain the maximum average time utility, we must first obtain candidate offloading strategies with a DNN and then input them into (Q2) to determine the best local computing resource allocation strategy. Intuitively, we can compute V^I feasible offloading strategies through enumeration (each self-driving vehicle has I cameras, and each of them has V possible partition points). However, this brute force search is computationally intensive, especially when the local computing resources need to be frequently reallocated due to time-varying channel gains and road traffic conditions, and it is difficult to obtain the object detection results efficiently. To address these problems, we propose DRPL, which can respond adaptively and quickly to the wireless and traffic environments.

Algorithm overview

The structure of the DRPL algorithm is illustrated in Fig. 4. We use a DNN as the fundamental network for generating candidate offloading policies, and we select the optimal strategy corresponding to the maximum time utility to participate in training in each time slot. Our goal is to derive an offloading strategy π_j based on the channel gain h_j and the execution priority of each object

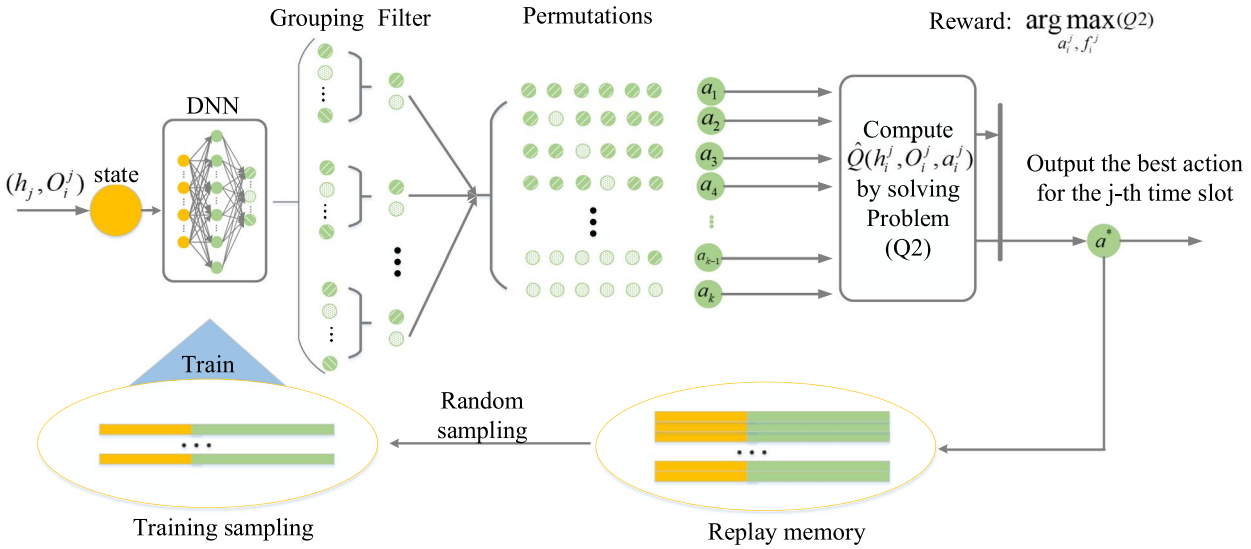


Fig. 4 DRPL-based offloading strategy generation and updating

detection task $O_i^j\{i = 1, 2, \ldots, I\}$ in time slot j, denoted by $\pi_j : \left\{h_j, O_i^j\right\} \rightarrow a_j^*$.

Specifically, at the beginning of the first time slot, we randomly set the internal DNN hyperparameters θ_1 (i.e., the link weights between the hidden-layer neurons), and the DNN generates the first offloading strategy based on the channel state and the initial in-vehicle camera priority information (here, all cameras are set to low-priority in the first time slot by default). In time slot j ($j \geq 2$), the priority information which determined by the navigation command and the object detection results from time slot $j-1$, in combination with the current channel state is input into the DNN to obtain an initial vector $\hat{a}_j$. At this point, the internal DNN hyperparameters are updated to θ_j. We divide $\hat{a}_j$ into I groups based on the different in-vehicle cameras. Each group contains V elements, and the sum of the probability values of the V elements in each group is 1. We separately select the k' elements in each group with the maximum probability values, thus obtaining K candidate offloading strategies (each offloading strategy has I dimensions). We sequentially input the K candidate offloading vectors into (Q2) and select the offloading strategy a_j^* that corresponds to the maximum time utility value $\hat{Q}(h_i^j, O_i^j, a_j^*)$. a_j^* is then combined with the state (h_j, O_i^j) to form the state–action pair $\left\{(h_j, O_i^j), a_j^*\right\}$, which is added to the experience memory unit.

In a general time slot j, we randomly draw a batch of samples from memory to train the DNN, and its parameters are updated from θ_j to θ_{j+1}. The new offloading strategy π_{j+1} is then used for the next time frame. In time slot $j+1$, we generate offloading strategy a_{j+1}^* based on the new channel gain and the new in-vehicle camera priority information (h_{j+1}, O_i^{j+1}) observed by the DNN. Thereafter, with continued observations the environment and the repetition of these iterative operations, the strategies that the DNN generates gradually improve.

Offloading strategy generation based on grouping and expansion

The parameters of the DNN in time slot j are denoted by θ_j (here, the initial parameters θ_1 are randomly assigned using the He initialization method. By inputting the channel gain h_j and the priority information O_i^j into the DNN, we can obtain a vector $\hat{a}_j$ with $I \times V$ dimensions. The mapping relation is expressed as follows:

$$\hat{a}_j = G_{\theta_j}(h_j, O_i^j), \quad i = 1, 2, \ldots, N, \hat{a}_j \in [0, 1]. \tag{18}$$

Here, we group the outputs $\hat{a}_j$ based on the different cameras and add a softmax function to normalize the results before each group, such that the sum of the probabilities of the V offloading partition points within each group will be 1.

However, if the number of selected offloading decision partition points V is larger than a certain value, this will lead to very small differences among the probability in each group. If we select only the offloading decision partition point with the largest probability in each group during every training iteration, we will lose a great deal of information about other possible points. This will lead to slow convergence of the network, greatly increasing the number of training rounds, consuming too much time and affecting the judgment ability of the network.

Therefore, we propose selecting the top k' candidate offloading partition points with the largest probability values within each group and combining them via permutation to form $K = k'^I$ offloading strategies. We then select the offloading strategy corresponding to the maximum time utility among these K candidate offloading strategies to participate in training in each time slot.

Piecewise linearization of the time utility function

As mentioned in the previous subsection, we need to input K offloading strategies into (Q2) sequentially to determine the local computational resource allocation policy by maximizing the task time utility value and then choose the corresponding best offloading policy a_j^* in each time slot.

Since the time utility function in this paper is a segmentation function that is partially no-nconvex, the original function needs to be transformed into a piecewise linear function to be solved. The PLM is commonly used to approximate complex nonlinear functions as piecewise ones; in this way, a complicated optimization problem can be transformed into a linear optimization problem [38, 39]. The segmentation points of the time utility function are transformed from a relationship between the delay tolerance τ_i^j and the task execution time t_i^j into a relationship with the local computing resource allocation policy f_i^j. The time utility function u_i^j is schematically plotted in Fig. 5. The specific steps of the PLM are as follows:

Step 1: We divide the local computing resource allocation proportion f (which lies in the interval [0,1]) into D segments, each corresponding to an interval of Δf:

$$\Delta f = \frac{1}{D}. \tag{19}$$

Step 2: For each segment $d \in D$, we connect $(f_d, u_i^j(f_d))$ to $(f_{d+1}, u_i^j(f_{d+1}))$ and transform it into a linear function $F_{i,d}^j, d \in \{1, 2, \ldots, D\}, i \in \{1, 2, \ldots, I\}, j \in \{1, 2, \ldots, N\}$.

Step 3: We use two sets of parameters, $\varphi_{i,d}^j$ with $d \in \{1, 2, \ldots, D+1\}$ and $\theta_{i,d}^j$ with $d \in \{1, 2, \ldots, D\}$, and the following formulas to transform the original segmented nonlinear function into a D-segment linear function:

$$\sum_{d=1}^{D} \theta_{i,d}^j = 1, \quad \forall i \in I, j \in N; \tag{20}$$

$$f_i^j = \sum_{d=1}^{D+1} \varphi_{i,d}^j \times f_d, \quad \forall i \in I, j \in N; \tag{21}$$

$$\sum_{d=1}^{D+1} \varphi_{i,d}^j = 1, \quad \forall i \in I, j \in N; \tag{22}$$

$$u_i^j = \sum_{d=1}^{D+1} \varphi_{i,\alpha}^j \times F_{i,d}^j(f_d), \quad \forall i \in I, j \in N; \tag{23}$$

$$\varphi_{i,1}^j \le \theta_{i,1}^j, \quad \forall i \in I, j \in N; \tag{24}$$

$$\varphi_{i,d}^j \le \theta_{i,d-1}^j + \theta_{i,d}^j, \quad 2 \le d \le D, \quad \forall i \in I, j \in N; \tag{25}$$

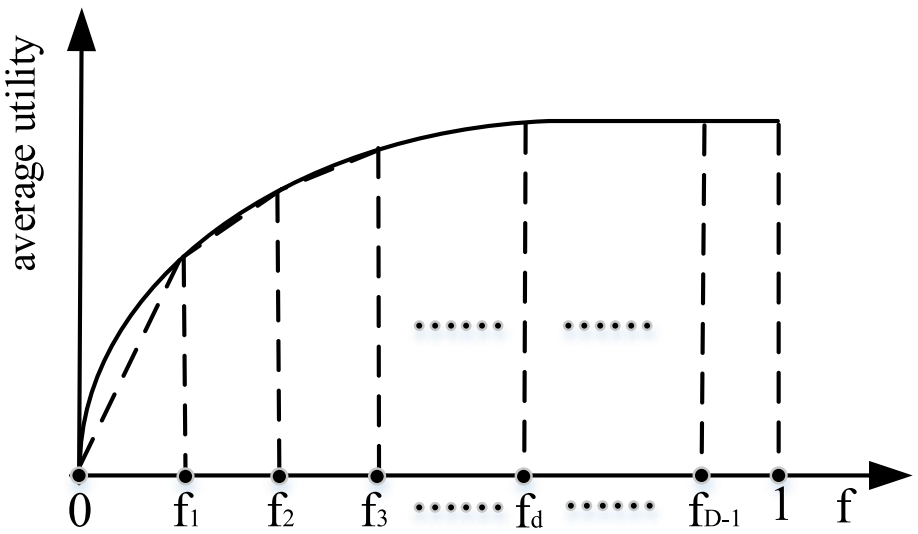


Fig. 5 Schematic diagram of the time utility function

$$\varphi^j_{i,D+1} \le \theta^j_{i,D}, \quad \forall i \in I, j \in N; \tag{26}$$

$$\theta^j_{i,d} \in \{0,1\}, \quad \forall i \in I, j \in N; \tag{27}$$

$$0 \le \varphi^j_{i,d} \le 1, \quad \forall i \in I, j \in N, \tag{28}$$

where $\varphi^j_{i,d} \in [0,1]$ is a weight associated with point f_d and $\theta^j_{i,d}$ is a binary variable indicating whether f^j_i falls within the d th segment.

Recall problem (Q2), by adding the new constraints given in Eqs. (20)–(28), we can obtain the following relaxed linear formulation (Q2').

$$\begin{aligned}(Q2'): \hat{Q}(h^j_i, O^j_i, a^j_i) &= \max_{\varphi^j_{i,d}, \theta^j_{i,d}} Q(h_j, O^j_i, f^j_i, a^j_i) \\ &\cong \max_{\varphi^j_{i,d}, \theta^j_{i,d}} \frac{1}{N} \times \sum_{i=1}^{I} \sum_{j=1}^{N} \sum_{d=1}^{D+1} \varphi^j_{i,\alpha} \times F^j_{i,d}(f_d)\end{aligned} \tag{29}$$

$$s.t. \quad (20) - (28). \tag{30}$$

After piecewise linearization, the original nonconvex nonlinear problem (Q2) is transformed into a piecewise linear problem, which can be solved using CPLEX. CPLEX is a commercial optimization software package and widely used for solving mathematical programming problems, including linear programming, mixed integer programming, and quadratic programming, etc.

Offloading policy update

In the DNN training phase, the training samples are correlated with each other because the priority of each camera is determined by the road condition information in consecutive frames. This may cause the offloading partition point selection algorithm to exhibit gradient descent in the same direction for a certain number of period of iterations in a row, and the training loss of the algorithm may not converge. To avoid this situation, we add an experience replay module to the algorithm to store past state–action pairs. In time slot j, the PLM is used to select the offloading actions a^*_j that corresponds to the maximum time utility among K candidate offloading strategies. a^*_j, together with the state information (h_j, O^j_i) of that time slot, then forms a new training sample $\left\{(h_j, O^j_i), a^*_j\right\}$.

We use the experience memory unit to train the DNN by randomly selecting a batch of ξ training samples (h_ξ, O^ξ_i) from the state parts of the samples $((h_\xi, O^\xi_i), a^*_\xi, \xi \in \Delta_j)$ in the memory and feeding them into the DNN. Then, the results are compared with a^*_ξ to calculate the cross-entropy, and the result is used as the training loss $Loss(\theta_j)$ to train the DNN. The cross-entropy calculation formula is shown as follow:

$$\begin{aligned}Loss(\theta_j) = -\frac{1}{|\Psi_j|} \sum_{\xi \in \Delta_j} ((a^*_j)^{\mathsf{T}} log G_{\theta_j}(h_\xi, O^\xi_i) \\ + (1 - a^*_j)^{\mathsf{T}} log(1 - G_{\theta_j}(h_\xi, O^\xi_i))),\end{aligned} \tag{31}$$

where Ψ_j represents the set of time indexes selected from the memory unit.

Input : the time slot j, the channel gain h_j, the priority information O^j_i, and the permutation base k'
Output : the best offloading strategy a^*_j and the corresponding local computing resource allocation strategy
if $j = 1$ **then**
  Initialize the DNN with random parameters θ_1 and empty the cache memory; set the number of network training rounds N, the learning rate, the batch size, and the training interval
end if
if $j \ge 2$ **then**
  for $j = 2$ to N **do**
    (1) The DNN generates an initial vector $\hat{a} = G_{\theta_j}(h_j, O^j_i)$ of $I \times v$ dimensions based on the input information (h_j, O^j_i)
    (2) Group the outputs $\hat{a}_j$ by the different cameras and obtain K offloading vectors $a_{j,k}, \forall j \in K$, in accordance with the permutation base k'
    (3) Input each $a_{j,k}$ into (Q2') in turn and compute $\hat{Q}(h^j_i, O^j_i, a^j_i)$
    (4) Select the best offloading strategy $a^*_j = \underset{a_{j,k}, \forall k \in K}{\operatorname{argmax}} \hat{Q}(h^j_i, O^j_i, a^j_i)$
    (5) Update the memory unit with $\left\{(h^j_i, O^j_i), a^*_j\right\}$
    if j mod γ==0 **then**
      (1) Randomly sample a set of data from the memory unit, $(\left\{(h_\xi, O^j_\xi), a^*_\xi\right\}, \xi \in \Psi_j)$
      (2) Train the DNN with$(\left\{(h_\xi, O^j_\xi), a^*_\xi\right\}, \xi \in \Psi_j)$ and update the hyperparameters with θ_j
      (3) Update the priority information of each camera in the next time slot according to Algorithm 1
    end if
  end for
end if

Algorithm 2 The DRPL task offloading and resource allocation algorithm based on end–edge collaboration

In summary, in every time slot, the priority status of each in-vehicle camera of the self-driving vehicle is determined based on its navigation commands and its object detection results from the previous time slot, and the priority information is fed into the DNN as state information together with the channel gain for training. Then, we group the initial vector into several candidate actions and calculate the time utility values separately, select the action a^*_j corresponding to the maximum time utility, and then combine it with the state information $\left\{(h_j, O^j_i), a^*_j\right\}$ to obtain the current state–action pair, which is stored in the memory unit. Finally, the DNN iteratively learns from the stored state–action pairs to generate more reasonable offloading strategies over time. Here, due to the limited memory space, we set the DNN to learn only from the latest data samples generated from the offloading policy. For details, see Algorithm 2. Our

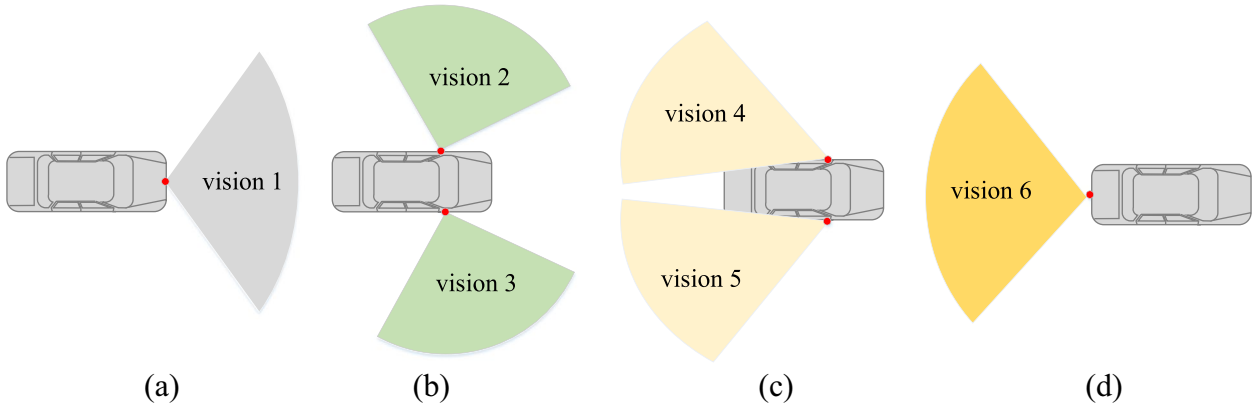


Fig. 6 The distribution of the in-vehicle cameras and their fields of view. **a** The directly front. **b** he left / left front, the right / tight front. **c** The left rear, the right rear. **d** The directly rear

algorithm does not involve any training or inference operations on the CNN. We mainly focuses on performing the inference of the DRL algorithm. We can easily obtain the computational complexity of our DRL model from formula $FLOPs = (2 \times I - 1) \times O$. Here, I denotes the dimension of the input layer, O denotes the dimension of the output layer. For example, a four-layer structured DNN model: one input layer, two hidden layers, and one output layer. The number of neurons are 7, 160, 80, 30. We can calculate that the computational effort of this model is about 30000$FLOPs$.

Numerical results analysis

In this section, we present the details of the experiments reported in this paper, including the setting of the in-vehicle cameras, the parameters of the simulation experiments, and the source of the training data. Here, we set the above parameters to closely approximate real-world traffic scenarios to the greatest extent possible. The experimental results are also analyzed and explained.

Experimental parameters

The in-vehicle cameras

Here, we assume that the self-driving vehicle has six in-vehicle cameras with views that collectively cover 360 degrees around the vehicle, and each CNN in every camera has five offloading partition points. The camera views including: the directly front, the left, the left front, the right, the right front, the left rear, the right rear, the directly rear to ensure all-round monitoring of the environment. The specific camera distribution is shown in Fig. 6.

The priority data for training the DNN

The vehicle navigation commands include directly ahead (DA), left turn/left front (L/LF), right turn/right front (R/RF), left rear (LR), right rear (RR), and directly rear (DR). In accordance with Fig. 6, the correspondence between the navigation commands and the priority of each in-vehicle camera is shown in Table 2.

Here, we consider four detection categories Y: pedestrian, car, truck, and bicycle. We assume that the probability of an object that appears on the road belonging to each of these four categories is 0.25. The ranges of pixel for the lengths and widths of the rectangular boxes corresponding to the detection results and their thresholds for each category are shown in Table 3.

Parameters of the simulation experiments

In this section, we use simulations to evaluate the DRPL algorithm. The simulation parameters used in the experiments are listed in Table 4. The equipment used in our simulation is a laptop with the following parameters: the CPU is AMD Ryzen 7 5800H with Radeon Graphics, running at 3.20 GHz; the GPU is an RTX 3060 with 12GB of memory; the RAM size is 32.0 GB. In DRPL, we consider a fully connected DNN consisting of one input layer, two

Table 2 In-vehicle camera priorities in accordance with different navigation commands (NCs)

Priority/NCs	DA	L/LF	R/RF	LR	RR	DR
High-priority	1,2,3	1,2,4	1,3,5	4,5,6	4,5,6	4,5,6
Low-priority	4,5,6	3,5,6	2,4,6	1,2,3	1,2,3	1,2,3

hidden layers and one output layer, where the first and second hidden layers have 160 and 80 hidden neurons, respectively, and the output layer has 30 neurons.

Analysis of numerical results

In this section, we evaluate the performance of our proposed DRPL algorithm through numerical simulations, which are divided into the following six topics for algorithm validation: convergence validation; piecewise segments number validation; permutation base value k' validation; task execution time analysis; offloading strategies and resource allocation results analysis; and the utility comparison of DRPL with other offloading partition point selection algorithms.

Convergence validation

In Fig. 7, we plot the training loss function $Loss(\theta_j)$ and the average sum of the time utilities for DRPL. As shown in Fig. 7(a), the average sum of time utilities gradually converges under DRPL, and when the number of training rounds is $\geq$ 300, the average time utility value exceeds 0.17. Meanwhile, as shown in Fig. 7(b), the training loss gradually decreases and stabilizes at approximately 0.05, after which its fluctuation is mainly due to the random sampling of the training data.

We also investigate the effects of different hyperparameters, including different learning rates, memory sizes, batch sizes, and training intervals, on the experimental convergence behavior. The effects of different training hyperparameters on the experimental results are shown in Fig. 8.

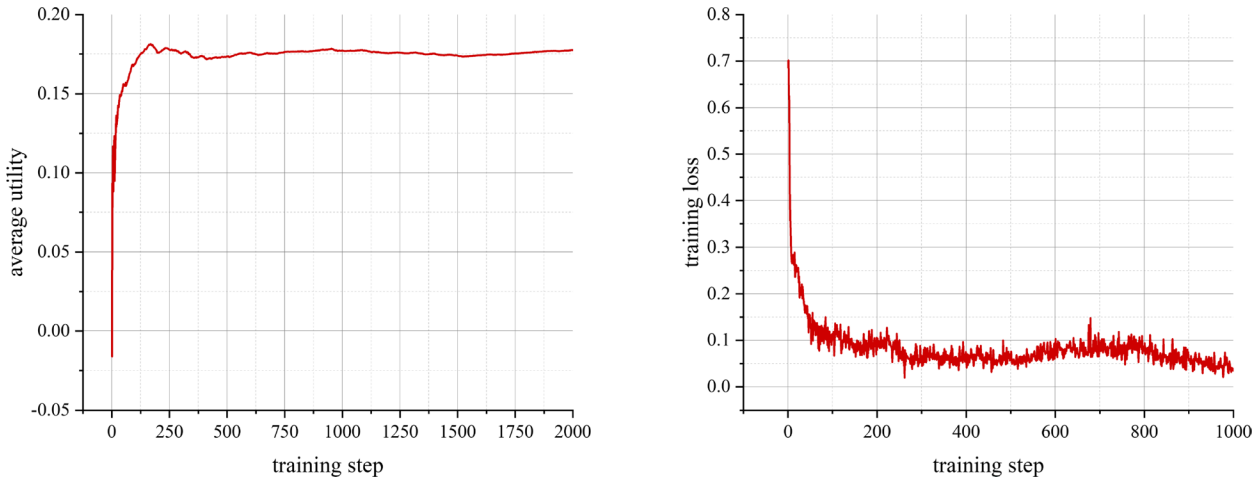
Figure 8(a) shows the convergence of the average time utility with different learning rates. When the learning rate is 0.1, the convergence of the time utility value reaches a local optimum. As the learning rate decreases, the time utility curve converges more slowly. Figure 8(b) shows the effect of different batch sizes on the convergence of the average time utility. When the batch size is set to 32 or 64, the training process often cannot fully utilize the abundance of data in the memory. On the other hand, when the batch size is too large, each iteration uses a large number of "old" data, which will greatly affect the network convergence performance. Figure 8(c) shows the effect of different memory sizes on the convergence of the average time utility. The time utility converges more slowly when the memory size is either too small or too large. In particular, when the memory size is equal to 1024, the DNN needs more training data to reach convergences. Figure 8(d) shows the effect of different training intervals on the convergence of the average time utility. The larger the training interval is, the more slowly the network converges; however, the training interval does not affect the final converged utility value.

Table 3 Pixel values of the rectangular detection boxes for each type of object

Category	Pixel ranges	Thresholds
Pedestrian	width [30,50], length [100,200]	width 45, length 115
Car	width [180,220], length [180,400]	width 210, length 350
Truck	width [380,420], length [380,800]	width 410, length 720
Bicycle	width [20,200], length [180,220]	width 170, length 215

Table 4 Simulation parameters

Parameter	Meaning	Value
B	The bandwidth	11 MB/s
τ_H	The execution delay tolerance of high-priority tasks	0.01 s
τ_L	The execution delay tolerance of low-priority tasks	0.02 s
Υ_H	The utility constant for high-priority tasks	0.2
Υ_L	The utility constant for low-priority tasks	0.1
N_0	The noise power	10^{-8}
P	The data transmission power	6 W
Υ	The computation time of the object detection task on the edge server	0.001s
F_L	The computing power of the vehicle terminal	1.08×10^6 bytes/s
$C^j_{i,v}$	The local computation size for task S^j_i at partition point v	[0, 2000] bytes
$M^j_{i,v}$	The data size of task S^j_i for offloading at partition point v	[0,5] MB
E	The potential for the object detection algorithm to suffer from detection error	0.02
$P^j_{i,x,y}$	The probability that the object x belongs to category y when the object detection algorithm achieves correct detection	[0.85,1]
$\overline{P}^j_{i,x,y}$	The probability that the object x belongs to category y when the object detection algorithm suffers from detection error	[0,0.85]



(a) Average utility (b) Training loss

Fig. 7 Convergence of the average utility and training loss values

The effect of the number of piecewise segments on the experimental results

In Fig. 9, we investigate the effect of different piecewise number on the experimental results. When the segment size is 2 or 5, the segmented curve does not fit the original function well, leading to poorer convergence results. As the segment size increases, the curve obtained through piecewise linearization (i.e., the curve of (Q2')) more closely approaches the original utility curve (i.e., the curve of Eq. (13)), and once the segment size reaches a certain value, the time utility curve converges with basically the same trend. Considering the computational cost, we set the segment size to 10 in this paper.

Influence of the permutation base value on the experimental results

During the quantized expansion phase, we select the top k' offloading partition points with the maximum probability in each group for permutation, and feed the candidate offloading strategies obtained in this way sequentially into the linear planning block.

As shown in Fig. 10, when $k' = 1$, we select only the offloading partition point corresponding to the maximum probability value in each group to form the offloading vector to participate in iterative network training. Since we update the network with only one offloading vector in each time slot, without providing any other option, many possibly better offloading solutions will be lost. So, it takes too many iterations for the network to converge. Therefore, we increase the value of k' appropriately. When the value is set to 2, the network convergence speed increases dramatically. However, when k' is further increased to 3, 4 or 5, the network convergence curves almost coincide. Considering that each increase in the k' value leading to an exponential increase in computation, we set $k'= 2$.

Analysis of task execution time

Figure 11 shows the object detection task execution time and the ratio of cameras with task execution times within their delay tolerance in each training step. Figure 11(a) shows the sum of the object detection task execution times within each camera of the self-driving vehicle for each training step. As the number of training steps increases, more appropriate offloading partition points are selected based on the channel state and the priority information, so that the task execution time decreases gradually. The task execution time fluctuations as the channel state changes and the different offload partition points with different sizes of offloading data and local computation sizes. Figure 11(b) shows the ratio of cameras with task execution times within their delay tolerance in each training step. At the beginning of training, the network has not yet converged, and the offload partition points and computational resources allocated for each camera are not well adapted to the time-varying wireless and the traffic environment. As the network converges, the object detection tasks for the cameras are basically completed within their delay tolerance time.

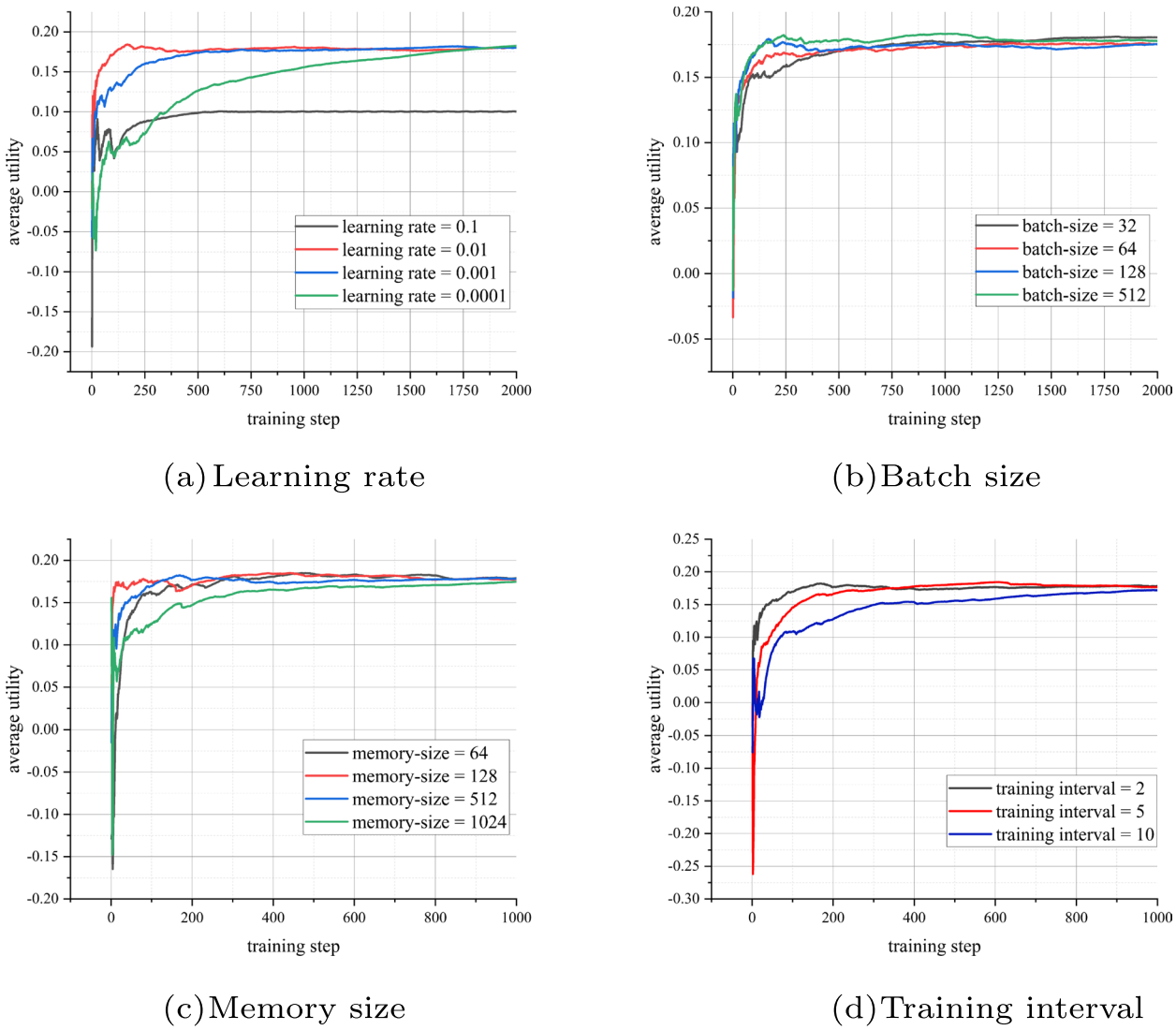


Fig. 8 Convergence of average utility under different typical hyperparameters

Analysis of offloading strategies and resource allocation results

We randomly select several time slots to further investigate the corresponding experimental results of task offloading and resource allocation after the neural network has converged. As shown in Fig. 12(a), (b), and (c), we divide the selected experimental results into three groups in accordance with the channel gain h. Each block from left to right presents the information of cameras 1 to 6 in order. Among them, the numbers in white blocks indicate the execution priority of the object detection task. Colored blocks indicate the offloading partition points selected for the object detection task of each camera, and the numbers marked on the colored blocks indicate the proportions of the local computing resources allocated to each in-vehicle camera. We can see that when h is small, offloading tasks to the edge server will require more time, and it is preferable to execute high-priority tasks locally, as shown in Fig. 12(a). At the same time, according to the numbers on the colored blocks, the resource allocation strategy tends to allocate more resources to high-priority tasks. When h is large, the data transfer time between the self-driving vehicle and the edge server is short. Then, the high-priority tasks are preferentially selected for offload execution, as shown in Fig. 12(c), and even when high-priority tasks are chosen to be locally executed, they are allocated more computing resources than low-priority tasks. When the h value is moderate, as shown in Fig. 12(b), the detection tasks of

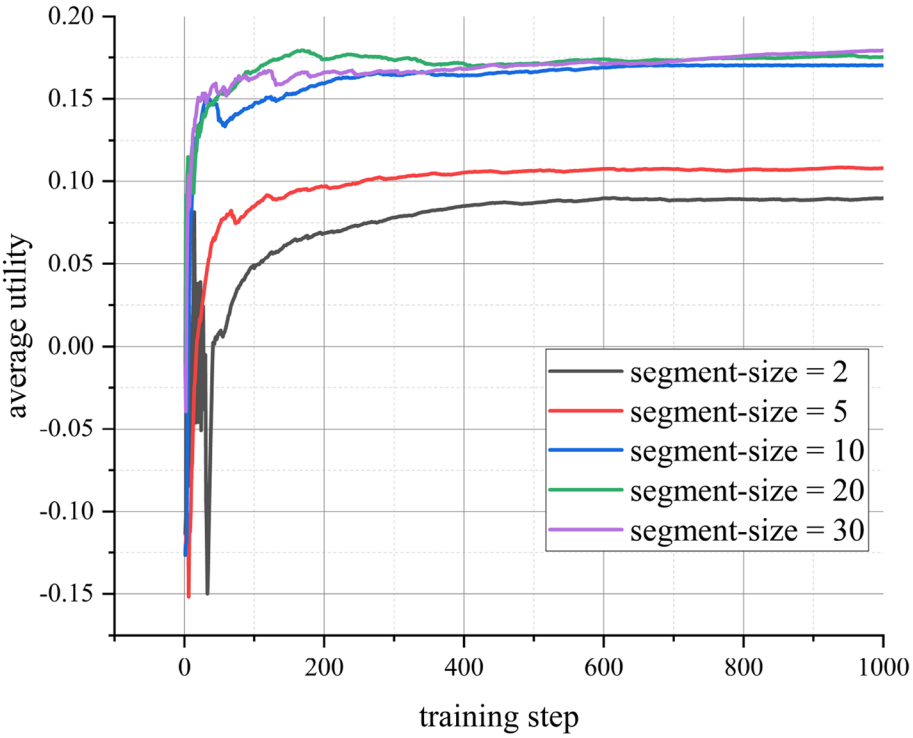


Fig. 9 Convergence of the average utility with different numbers of piecewise linearized segments

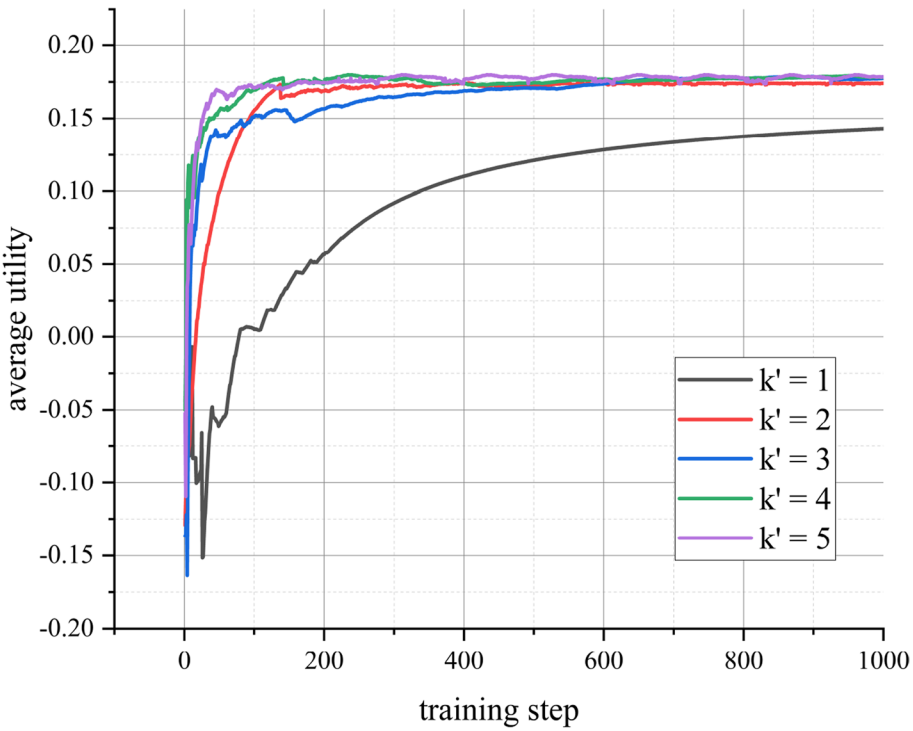


Fig. 10 Convergence of the average utility with different permutation base values

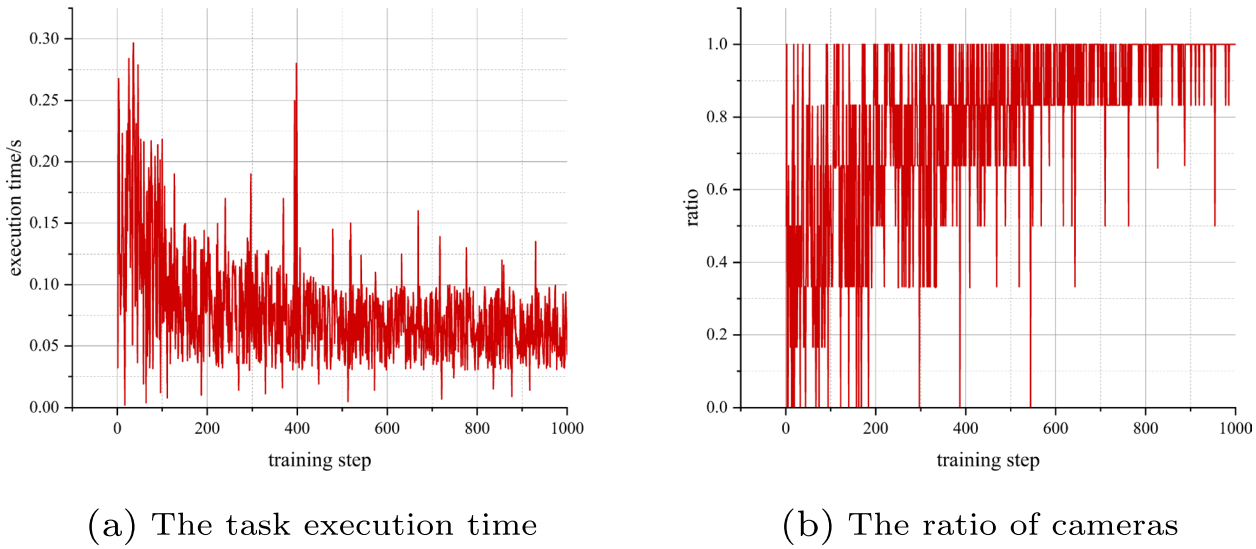


Fig. 11 The sum of the tasks execution time and the ratio of cameras with task execution time within their delay tolerance in each training step

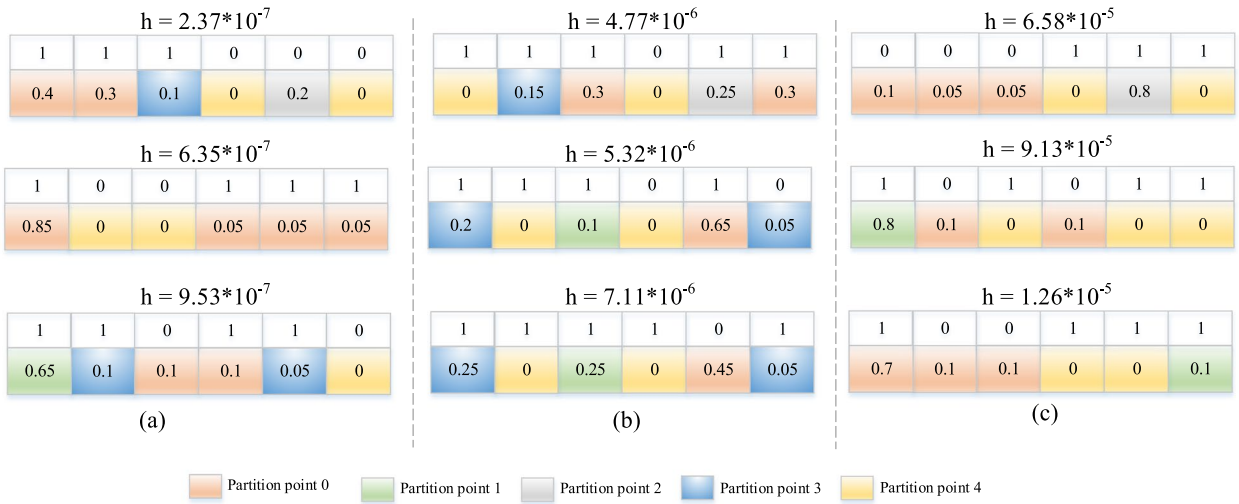


Fig. 12 Task offloading and resource allocation results under different channel states

each in-vehicle camera are selected for offloading and resource allocation in accordance with the actual situation, with the goal of maximizing the task execution time utility value for each camera.

Comparative analysis of the execution time utility of different task offloading algorithms

To validate the effectiveness of DRPL, we have selected several representative algorithms for comparison.

- Edge computing [23]: In each time slot *j*, all object detection tasks of the in-vehicle cameras are offloaded to the edge server for execution, i.e., $a_i^j = 4$, $i = 1, 2, \ldots, I$, $j = 1, 2, \ldots, N$.
- Local computing: In each time slot j, all object detection tasks of the in-vehicle cameras are executed locally, i.e., $a_i^j = 0, i = 1, 2, \ldots, I$, $j = 1, 2, \ldots, N$.
- Greedy: In each time slot *j*, we select the partition point that contributes the most to maximize the average sum of time utilities for each object detection task in all slots. Here, we define the contribution degree at partition point v of camera i as $degree_{i,v}^j$, which is calculated according to communication to computation ratio (CCR) [40].

If the $degree_{i,v}^{j}$ values are all less than 1, we select the offloading partition point corresponding to the minimum $degree_{i,v}^{j}$ for the high-priority tasks, and the low-priority tasks for local computation. If the $degree_{i,v}^{j}$ values are all greater than 1, the high-priority tasks perform the local calculation and the low-priority tasks select the partition point corresponding to the minimum $degree_{i,v}^{j}$ for offloading calculation. If both $degree_{i,v}^{j}$ values greater than 1 and less than 1 coexist, we calculate the farthest distance between the two parts of $degree_{i,v}^{j}$ from 1,i.e. $d_{greater}^{j}$ and d_{less}^{j} respectively, if $d_{greater}^{j} > d_{less}^{j}$, the situation is treated as if the $degree_{i,v}^{j}$ values are all greater than 1, if $d_{greater}^{j} < d_{less}^{j}$, the case is treated as if the $degree_{i,v}^{j}$ values are all less than 1.

- Random offloading: In this algorithm, we randomly select local or offloaded calculation for the detection task of each in-vehicle camera in each time slot.

In Fig. 13, we compare different offloading algorithms in terms of the average sum of time utilities for each object detection task in all slots under different sizes of local computing resources. Series 1 is the average time utility with all computations offloaded, series 2 is the average time utility with all computations executed locally, series 3 is the average time utility with the greedy algorithm, series 4 is the average time utility with the random offloading algorithm, and series 5 is the average time utility with the DRPL algorithm proposed in this paper.

As seen in Fig. 13, since both the local computing resources and the wireless bandwidth are limited, if we select only local computing or offloaded computing, the average task execution time utility values are small and may even be negative when the abundance of local computing resources is sufficiently low. This indicates that even the high-priority tasks are not completed within their time delay tolerance. The greedy algorithm takes into account the time-varying wireless environment and obtains better time utility values. However, it does exploit the historical strategy experience. The offloading strategies chosen by the random algorithm are unstable; therefore, this algorithm is not effective. In contrast, DRPL produces offloading strategies which taking into account historical task offloading experience; consequently, it shows much better capabilities. We can see that under the different considered F_L values, DRPL achieves improvements of 12.8%, 17.4% and 15.5% in average time utility compared to the task offloading method corresponding to the maximum average utility within each group.

Conclusion

In this paper, we propose DRPL for the object detection tasks of self-driving vehicles. We maximize the average sum of the time utilities for each object detection task

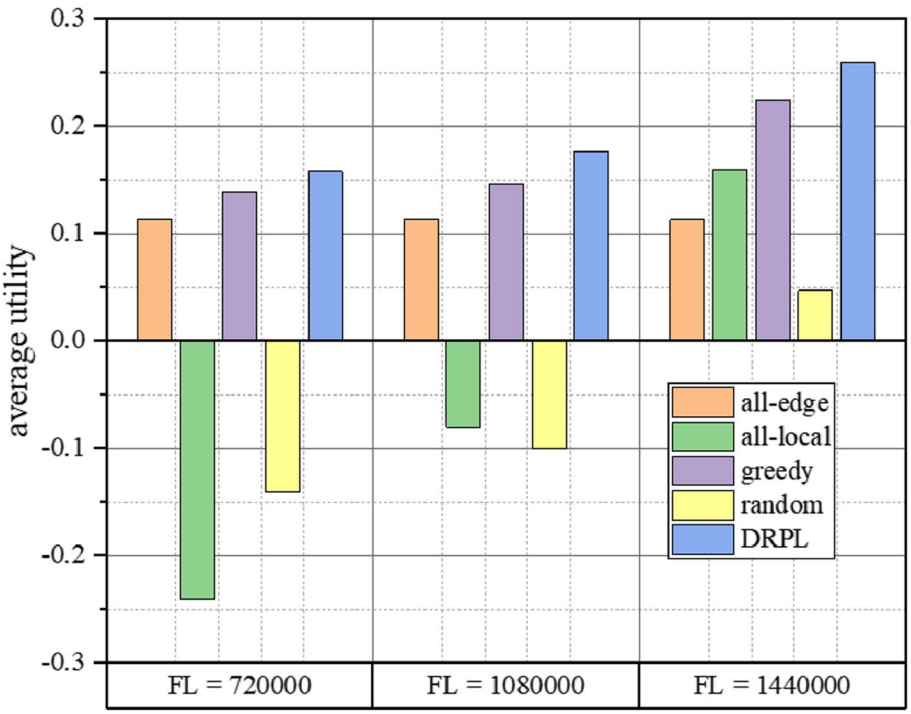


Fig. 13 Comparisons of average utility performance for different offloading algorithms under different F_L values

in all slots. The proposed algorithm can make full use of historical task offloading experience, jointly with a PLM-based local computing resource allocation strategy, and thus progressively improve a DNN to generate better offloading strategies. We also develop a priority determination mechanism based on the vehicle navigation commands and historical object detection results. Meanwhile, to speed up network convergence, we group the DNN outputs by cameras and expand them via permutation. The DRPL algorithm well addresses the problem of object detection task offloading and local computing resource allocation for self-driving vehicles in complex traffic scenarios. Numerical results show that DRPL obviously superior effectiveness compared with the traditional algorithm schemes.

In this paper, we have explored the problem of task offloading and resource allocation for one self-driving vehicle. However, the proposed DRPL approach is also applicable to multiple self-driving vehicles. In future work, we will use actual traffic and wireless environment data to further validate our experiments. Additionally, we will explore training acceleration algorithms for DRL network models to ensure the timeliness of our proposed approach in practical scenarios as much as possible.

Acknowledgements
This work is supported by the Natural Science Foundation of China (No. 61872104), the Fundamental Research Fund for the Central Universities in China and Tianjin Key Laboratory of Advanced Networking (TANK) in College of Intelligence and Computing of Tianjin University. This work is partially supported by the project "PCL Future Greater-Bay Area Network Facilities for Large-scale Experiments and Applications (LZC0019)".

Authors' contributions
Lili Nie wrote the main manuscript text. All authors reviewed the manuscript.

Availability of data and materials
Not applicable.

Declarations

Ethics approval and consent to participate
Not applicable.

Consent for publication
Not applicable.

Competing interests
The authors declare no competing interests.

Received: 20 March 2023 Accepted: 11 August 2023
Published online: 12 September 2023

References

1. Krizhevsky A, Sutskever I, Hinton GE (2017) Imagenet classification with deep convolutional neural networks. Commun ACM 60(6):84–90
2. Deng J, Ding N, Jia Y, Frome A, Murphy K, Bengio S, Li Y, Neven H, Adam H (2014) Large-scale object classification using label relation graphs. In: European Conference on Computer Visio. vol 8689. Springer, Zurich, p 48–64. https://doi.org/10.1007/978-3-319-10590-1_4
3. He K, Zhang X, Ren S, Sun J (2016) Deep residual learning for image recognition. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). IEEE, Las Vegas, p 770–778. https://doi.org/10.1109/CVPR.2016.90
4. Huang G, Liu Z, van der Maaten L, Weinberger KQ (2017) Densely connected convolutional networks. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). IEEE, Honolulu, p 4700–4708. https://doi.org/10.1109/CVPR.2017.243
5. Xie S, Girshick R, Dollár P, Tu Z, He K (2017) Aggregated residual transformations for deep neural networks. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). IEEE, Honolulu, p 1492–1500. https://doi.org/10.1109/CVPR.2017.634
6. Deng J, Ding N, Jia Y, Frome A, Murphy K, Bengio S, Li Y, Neven H, Adam H (2014) Large-scale object classification using label relation graphs. In: Fleet DJ, Pajdla T, Schiele B, Tuytelaars T (eds) European conference on computer vision, vol 8689. pp 48–64
7. Chen L, Hu X, Xu T, Kuang H, Li Q (2017) Turn signal detection during nighttime by CNN detector and perceptual hashing tracking. IEEE Trans Intell Transp Syst 18(12):3303–3314
8. Chen L, Zou Q, Pan Z, Lai D, Zhu L, Hou Z, Wang J, Cao D (2020) Surrounding vehicle detection using an FPGA panoramic camera and deep cnns. IEEE Trans Intell Transp Syst 21(12):5110–5122
9. Zou Q, Zhang Z, Li Q, Qi X, Wang Q, Wang S (2019) Deepcrack: Learning hierarchical convolutional features for crack detection. IEEE Trans Image Process 28(3):1498–1512
10. Qi Q, Wang J, Ma Z, Sun H, Cao Y, Zhang L, Liao J (2019) Knowledge-driven service offloading decision for vehicular edge computing: A deep reinforcement learning approach. IEEE Trans Veh Technol 68(5):4192–4203
11. Liu Y, Yu H, Xie S, Zhang Y (2019) Deep reinforcement learning for offloading and resource allocation in vehicle edge computing and networks. IEEE Trans Veh Technol 68(11):11158–11168
12. Fan W, Liu J, Hua M, Wu F, Liu Y (2022) Joint task offloading and resource allocation for multi-access edge computing assisted by parked and moving vehicles. IEEE Trans Veh Technol 71(5):5314–5330
13. Tan LT, Hu RQ, Hanzo L (2019) Twin-timescale artificial intelligence aided mobility-aware edge caching and computing in vehicular networks. IEEE Trans Veh Technol 68(4):3086–3099
14. Guo J, Song B, Chen S, Yu FR, Du X, Guizani M (2020) Context-aware object detection for vehicular networks based on edge-cloud cooperation. IEEE Internet Things J 7(7):5783–5791
15. Kim S, Ko K, Ko H, Leung VCM (2021) Edge-network-assisted real-time object detection framework for autonomous driving. IEEE Netw 35(1):177–183
16. Zhaojun N, Sheng Z, Yunjian J, Zhisheng N (2023) Joint task offloading and resource allocation for vehicular edge computing with result feedback delay. IEEE Trans Wirel Commun 1–1. https://doi.org/10.1109/TWC.2023.3244391
17. Gao J, Kuang Z, Gao J, Zhao L (2023) Joint offloading scheduling and resource allocation in vehicular edge computing: A two layer solution. IEEE Trans Veh Technol 72(3):3999–4009
18. Deng C, Fang X, Wang X (2023) Uav-enabled mobile-edge computing for AI applications: Joint model decision, resource allocation, and trajectory optimization. IEEE Internet Things J 10(7):5662–5675
19. Zhou H, Wang Z, Min G, Zhang H (2023) Uav-aided computation offloading in mobile-edge computing networks: A stackelberg game approach. IEEE Internet Things J 10(8, April 15):6622–6633
20. Mao S, Leng S, Maharjan S, Zhang Y (2020) Energy efficiency and delay tradeoff for wireless powered mobile-edge computing systems with multi-access schemes. IEEE Trans Wirel Commun 19(3):1855–1867
21. Chen X, Dai W, Ni W, Wang X, Zhang S, Xu S, Sun Y (2023) Augmented deep reinforcement learning for online energy minimization of wireless powered mobile edge computing. IEEE Trans Commun 71(5):2698–2710
22. Deng X, Li J, Shi L, Wei Z, Zhou X, Yuan J (2022) Wireless powered mobile edge computing: Dynamic resource allocation and throughput maximization. IEEE Trans Mob Comput 21(6):2271–2288
23. Mao S, Zhang N, Liu L, Wu J, Dong M, Ota K, Liu T, Wu D (2021) Computation rate maximization for intelligent reflecting surface enhanced wireless

powered mobile edge computing networks. IEEE Trans Veh Technol 70(10):10820–10831
24. Shnaiwer YN, Kaneko M (2023) Minimizing iot energy consumption by irs-aided UAV mobile edge computing. IEEE Netw Lett 5(1):16–20
25. Song Y, Liu Y, Zhang Y, Li Z, Shou G (2023) Latency minimization for mobile edge computing enhanced proximity detection in road networks. IEEE Trans Netw Sci Eng 10(2):966–979
26. Zhou H, Wang Z, Zheng H, He S, Dong M (2023) Cost minimization-oriented computation offloading and service caching in mobile cloud-edge computing: An a3c-based approach. IEEE Trans Netw Sci Eng 10(3):1326–1338
27. Liu S, Liu L, Tang J, Yu B, Wang Y, Shi W (2019) Edge computing for autonomous driving: Opportunities and challenges. Proc IEEE 107(8):1697–1716
28. Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, Graves A, Riedmiller MA, Fidjeland A, Ostrovski G, Petersen S, Beattie C, Sadik A, Antonoglou I, King H, Kumaran D, Wierstra D, Legg S, Hassabis D (2015) Human-level control through deep reinforcement learning. Nat 518(7540):529–533
29. Wei F, Feng G, Sun Y, Wang Y, Qin S, Liang Y (2020) Network slice reconfiguration by exploiting deep reinforcement learning with large action space. IEEE Trans Netw Serv Manag 17(4):2197–2211
30. Chen J, Xing H, Xiao Z, Xu L, Tao T (2021) A DRL agent for jointly optimizing computation offloading and resource allocation in MEC. IEEE Internet Things J 8(24):17508–17524
31. Ren Y, Chen X, Guo S, Guo S, Xiong A (2021) Blockchain-based VEC network trust management: A DRL algorithm for vehicular service offloading and migration. IEEE Trans Veh Technol 70(8):8148–8160
32. Hazarika B, Singh K, Biswas S, Li C (2022) Drl-based resource allocation for computation offloading in iov networks. IEEE Trans Ind Inform 18(11):8027–8038
33. Lillicrap TP, Hunt JJ, Pritzel A, Heess N, Erez T, Tassa Y, Silver D, Wierstra D (2016) Continuous control with deep reinforcement learning. In: 4th International Conference on Learning Representations (ICLR). San Juan, p 2–4. http://arxiv.org/abs/1509.02971
34. Huang L, Bi S, Zhang YA (2020) Deep reinforcement learning for online computation offloading in wireless powered mobile-edge computing networks. IEEE Trans Mob Comput 19(11):2581–2593
35. Shi J, Du J, Wang J, Wang J, Yuan J (2020) Priority-aware task offloading in vehicular fog computing based on deep reinforcement learning. IEEE Trans Veh Technol 69(12):16067–16081
36. Murty KG, Kabadi SN (1987) Some np-complete problems in quadratic and nonlinear programming. Math Program 39(2):117–129
37. Ao W, Psounis K (2018) Fast content delivery via distributed caching and small cell cooperation. IEEE Trans Mob Comput 17(5):1048–1061
38. Correa-Posada CM, Sanchez-Martin P (2014) Integrated power and natural gas model for energy adequacy in short-term operation. IEEE Trans Power Syst 30(6):3347–3355
39. Shao C, Wang X, Shahidehpour M, Wang X, Wang B (2016) An milp-based optimal power flow in multicarrier energy systems. IEEE Trans Sustain Energy 8(1):239–248
40. Li W, Yang T, Delicato FC, Pires PF, Tari Z, Khan SU, Zomaya AY (2018) On enabling sustainable edge computing with renewable energy resources. IEEE Commun Mag 56(5):94–101

Publisher's Note