Deep Reinforcement Learning-Based Joint Caching and Offloading for Vehicular Edge Computing: Supporting Multiple Types of Service Requests for Single Task

Abstract

In recent years, vehicular edge computing (VEC) has become a promising paradigm. In previous studies on offloading in VEC, a task is always assumed to request only one type of service, which limits the universality of the research. To address this limitation, we propose a deep reinforcement learning (DRL)-based joint service caching and task offloading scheme, named DRLSCTO. The scheme offers three main advantages: 1) it allows each task to request multiple types of services simultaneously; 2) it enables service type-based fine-grained task decomposition and distributed offloading of subtasks; 3) it supports both cloud collaboration and edge collaboration. To thoroughly assess the effectiveness of DRLSCTO, we conduct extensive comparative experiments against four benchmark schemes. First, we investigate the convergence behavior of different schemes under dynamic task data size and vehicle density. Second, by varying the task data size and vehicle density respectively, we reveal the variation pattern of task computation delay. Third, with the vehicle density and task data size held constant, we explore the impact of the edge node CPU frequency on the energy consumption of mobile network operators (MNOs). Results demonstrate that DRLSCTO achieves the lowest task computation delay while maintaining reasonable energy consumption compared to its rivals.

Keywords: Service caching, Task offloading, Vehicular network, Deep reinforcement learning

1. Introduction

With the rapid development of autonomous vehicles and the Internet of Things (IoT), the Internet of Vehicles (IoV) has emerged as an integrated network connecting vehicles, small cloud servers deployed at roadside units (RSUs), and centralized cloud systems. The IoV enables various communication modes and fosters the deployment of vehicular applications with computational intensity and strict latency requirements. Due to the limited caching and computing resources of vehicles, as well as the high latency and backhaul overhead associated with cloud computing, supporting such applications remains challenging [1, 2]. To address these issues, mobile edge computing (MEC)—a technology that deploys computing, caching, and communication resources at the network edge close to end devices—has attracted wide attention in the IoV field.

Researchers have integrated MEC into the IoV, leading to the development of vehicular edge computing (VEC) [3]. This architecture enables vehicles to migrate applications to the RSUs equipped with edge servers or request popular content from these RSUs, reducing task computation and content request delays. Despite the numerous advantages of VEC, edge servers still suffer from significantly limited computing and caching resources compared to centralized cloud servers [4]. To overcome this limitation, a cloud-edge collaborative VEC framework has been applied [5, 6]. Under this framework, vehicular tasks can be executed locally, offloaded to nearby edge servers, or further forwarded to centralized cloud servers. Most existing works based on this framework focus primarily on content caching strategies, task offloading schemes, or their integration. However, the crucial role of service caching in enabling efficient task offloading within VEC is always neglected, especially given the limited caching resources of edge servers.

Service caching involves storing specific service programs required by vehicular applications. This practice reduces the computation delay associated with service requests and application initialization, thereby improving the quality of experience (QoE). In fact, service caching is a prerequisite for task execution, as only the nodes with the corresponding service program can process a task [7]. This dependency creates a strong coupling between service caching and task offloading decisions. Moreover, vehicular tasks exhibit significant variability in service dependencies, making preconfigured caching and offloading decisions infeasible. These technical challenges are further amplified by the rapid proliferation of new energy vehicles and their associated smart applications. These applications often integrate multiple functions—each corresponding to a distinct service program—and thus exacerbate the caching burden on edge servers [8]. Therefore, there is an urgent need for a joint service caching and task offloading scheme to address these challenges. However, many existing studies still rely on greedy heuristics or offline optimizations, which struggle to handle the dynamic characteristics of vehicular networks. Although some researchers have employed reinforcement learning (RL) to assist decision-making, the assumption that each task depends on a single type of service fails to capture the requirements of multi-function applications.

To address the aforementioned issues, we focus on a cloud-edge collaborative VEC framework and propose a deep reinforcement learning (DRL)-based joint service caching and task offloading scheme, named DRLSCTO. Specifically, a service-based task decomposition mechanism is introduced, where each

task is decomposed into multiple subtasks according to different service programs, allowing flexible offloading based on distributed service caches. The joint decision-making process is formulated as a Markov decision process (MDP), where the system state captures the dynamic network conditions, including vehicle density, service caching states, and heterogeneous service requirements. The action space simultaneously determines how services are cached and how tasks are offloaded. To handle the high-dimensional, continuous decision space, we adopt the deep deterministic policy gradient (DDPG) algorithm to learn an adaptive policy that minimizes the task computation delay. This paper makes the following primary contributions:

1. We propose a cloud-edge collaborative VEC framework that incorporates a service type-based task decomposition mechanism. It not only allows each task to request multiple types of services simultaneously but also supports fine-grained task decomposition. To minimize the task computation delay, the joint service caching and fine-grained task offloading optimization problem is formulated as a mixed integer nonlinear programming (MINLP) problem, which considers heterogeneous service types and caching constraints.
2. For the NP-hardness of the formulated problem, we design the DRLSCTO scheme, which is capable of jointly optimizing caching and offloading decisions under continuous and high-dimensional system states. Unlike prior works that decouple caching and offloading, our scheme jointly learns service caching and task offloading decisions in a continuous, high-dimensional action space, capturing the interdependencies between the two.
3. We conduct extensive experiments under varying conditions, including task data size, vehicle density, and edge node CPU frequency, to validate the adaptability and effectiveness of our proposed DRLSCTO scheme. The results show that DRLSCTO outperforms the four benchmark schemes in reducing task computation delay across diverse scenarios and maintains reasonable energy consumption as edge node CPU frequency increases.

The remainder of this paper proceeds as follows. Following the literature review in Sec. 2, Section 3 introduces the system model and MINLP formulation. Section 4 elaborates on the DRLSCTO scheme, with experimental results and analysis provided in Sec. 5. Finally, Section 6 concludes the paper.

2. Related Work

2.1. Studies on MEC

In recent years, task offloading has emerged as a research hotspot in MEC [9]. To minimize time cost, Lin et al. [10] proposed a Lyapunov-based resource allocation scheme in unmanned aerial vehicles (UAVs)-assisted heterogeneous MEC system. Zhang et al. [11] utilized Lagrangian relaxation (LR) in the bidirectional task offloading model, formulating the problem of minimizing the average bandwidth as an auxiliary problem to obtain a locally optimal solution. Nevertheless, traditional optimization methods often struggle to adapt to the highly dynamic MEC environments. For this reason, RL techniques have been widely adopted to derive adaptive strategies through online interactions. Wen et al. [12] tackle the joint challenge of task sequencing and resource assignment by introducing a deep learning architecture that intelligently weighs system constraints and performance objectives. Zhang et al. [13] developed a DRL-based offloading strategy for latency-sensitive home devices to reduce system costs and enhance user experience. Zeng et al. [14] proposed a three-layer cloud-edge-end collaboration (CEEC) architecture, jointly optimizing offloading strategies, computing resources, and network channels under a newly defined QoE metric. To cope with the resource-sharing nature of specialized tasks, Xie et al. [15] modeled granularity decisions as a multi-armed bandit problem and applied RL to effectively reduce edge server costs. Tang et al. [16] further proposed a fully distributed DRL-based scheme where each device independently determines its offloading strategy, significantly reducing packet loss and execution delay.

In parallel with the advances in task offloading, substantial research has been devoted to edge caching for improving content or service delivery efficiency in MEC systems. Bounaira et al. [17] focused on mitigating the privacy and security challenges caused by the trust gap between content providers and edge servers in a blockchain-based trust management framework. Beyond privacy and security, fairness in cache allocation has also attracted attention. Zhou et al. [18] investigated the fair edge data caching (FEDC) problem arising from limited storage and vendor competition, while Tang et al. [19] introduced a genetic algorithm-based two-level caching and offloading method to ensure equitable resource distribution. The dynamic nature of content popularity presents additional issues. Mao et al. [20] designed a collaborative meta-learning framework that leverages selective neighbor sampling to improve adaptability, achieving 10.12% increase in cache hit rate. Zhao et al. [21] proposed a two-phase proactive caching approach to address uncertain task requirements resulting from the constantly changing content popularity, improving request prediction accuracy.

In summary, MEC has been widely investigated to reduce task computation delay or improve content delivery efficiency. However, most existing MEC studies focus on relatively static or low-mobility scenarios. In contrast, VEC features highly dynamic topologies, rapidly changing wireless channels, and stringent task computation delay requirements. These characteristics make direct application of traditional MEC approaches suboptimal in VEC scenarios, thereby motivating dedicated research on VEC-oriented caching and offloading strategies.

2.2. Studies on VEC

Recent studies have increasingly focused on developing and investigating VEC-specific solutions to address the unique challenges of vehicular networks, such as high mobility and dynamic topology. In the following, we conduct a review of representative research centered on caching and computation offloading. To address energy concerns from the perspective of mobile network operators (MNOs), Kong et al. [22]

adopted the DDPG algorithm to optimize joint computing and caching resources allocation. Li et al. [23] explored a novel resource augmentation paradigm by harnessing idle capacities from underutilized vehicles, establishing a stable, incentive-driven market model to ensure reliable collaboration. Similarly, Chen et al. [24] proposed a hybrid optimization framework combining deep Q-network (DQN) for task offloading with a greedy strategy for resource management. The framework effectively reduced system energy consumption under delay constraints. Beyond centralized solutions, distributed learning techniques have gained progress in facilitating collaborative caching and offloading. For instance, Wu et al. [25] proposed a social-aware decentralized cooperative caching (SADC) algorithm that leverages vehicle social networks to estimate contact rates, reducing the average content access delay. To enhance caching performance, Wu et al. [26] designed a multi-agent federated DRL-based collaborative caching strategy (MFDRL-CCS). This approach employs recurrent neural networks (RNN) to select optimal caching vehicles (CVs) and uses a multi-head attention popularity prediction (MHAPP) model to forecast content demand.

The abovementioned studies on VEC mainly focus on terrestrial vehicular networks, while some researchers have extended the VEC architecture into more heterogeneous and hierarchical environments. Gong et al. [27] conducted the first survey on edge computing with space, air and ground. They applied an offline deep imitation learning (DIL) algorithm to enable real-time caching and offloading decisions, and also discussed potential integration directions. Similarly, Yu et al. [28] addressed high-traffic density by integrating UAVs into cellular networks for cooperative caching. They used a temporal-evolving bipartite graph neural network (TBGN) to predict mobility patterns and optimize UAV trajectories. Liu et al. [29] established a cooperative model incorporating mobile aerial units to expand service capacity, employing a strategic incentive structure to align the objectives of heterogeneous participants and maximize systemic utility.

In addition to content caching, service caching has emerged as a critical component in VEC due to its requirement for both storage and computation resources. Xue et al. [30] proposed a DRL-based joint service caching and task offloading scheme that emphasizes the inherent coupling between service caching and computation task decisions. Extending this foundation, Liu et al. [31] investigated a model with linearly related requests where the output of one task serves as the input for the next. They analyzed the impact of various factors on task execution latency and system energy consumption. Cheng et al. [32] proposed an algorithm which integrates dynamic programming based service caching with game theory for task offloading. This algorithm jointly considered caching preferences to minimize the overall task processing cost. Service caching preferences were also considered by Ling et al. [33] with particular emphasis on this aspect. They developed a transportation-aware, data-driven cache replacement mechanism that dynamically adapts to real-time changes in vehicle mobility and service preferences.

Although VEC has been extensively studied, literature focusing on the joint optimization of service caching and task offloading remains limited. Furthermore, most existing works assume that edge nodes possess sufficient caching resources, with only a few works incorporating explicit upper bounds on caching capacities [30, 31]. Among these studies that incorporate resource constraints, offloading decisions only depend on a single type of cached service. To fill this gap, this paper jointly optimizes service caching and task offloading in a VEC system with heterogeneous service requirements. Specifically, we consider a task model in which each task may require multiple types of services. To capture the highly dynamic nature of vehicular environments, we simulate scenarios by varying key parameters such as task data size and edge node CPU frequency. The goal of this paper is to minimize the task computation delay and maintain a reasonable growth rate of MNOs' energy consumption.

3. System Model

We first provide an overview of our system, and then elaborate on the system models from five perspectives: the communication model, the service generation model, the joint service caching and task offloading model, the computation delay and energy consumption model, and the extremum analysis model. Next, we analyze the extremum values of task computation delay and energy consumption to gain insights into the task computation delay and MNOs' energy consumption under different caching scenarios.Finally, we formulate the optimization problem. For better readability, the main notations used in this section are summarized in Table 1.

3.1. System Overview

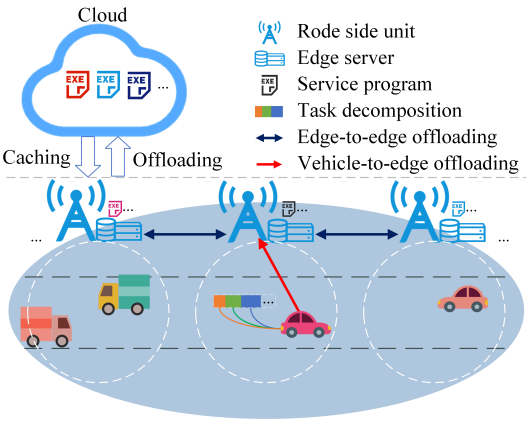

Figure 1: System model.

It is assumed that each edge node obtains complete information about all the vehicles within its coverage area, and that the coverage areas of different edge nodes are non-overlapping, collectively covering the entire road network. The system is modeled as a three-layer VEC architecture as shown in Fig. 1, consisting of N RSUs equipped with edge servers, L vehicles and a cloud server. RSUs are denoted by $\mathcal{N} = \{n \mid$

Table 1: Main Notations

Notation	Meaning
$\mathcal{L}$	The set of vehicles
L	The total number of vehicles
$\mathcal{N}$	The set of RSUs
N	The total number of RSUs
C_n	The set of cooperative edge nodes for edge node n
$\mathcal{T}$	The set of time slots
T	The total number of time slots for each episode
$\mathcal{K}$	The set of service
d_j	The input data size of task j
$s_{j,k}$	The type-k subtask for task j
$d_{j,k}$	The input data size of $s_{j,k}$
$f_{j,k}$	The dependency variable of task j with type-k service
$b_{l,n}$	The bandwidth allocated to vehicle l by edge node n
$\delta^V_{l,k}$	The cache hit state of vehicle l for type-k service
$\delta^R_{n,k}$	The cache hit state of edge node n for type-k service
f^V	The CPU frequency of a vehicle
f^R	The CPU frequency of an edge node
$\omega^V_{l,j,k}$	The offloading proportion of $s_{j,k}$ from vehicle l to edge node
$\omega^R_{n,j,k}$	The offloading proportion of $s_{j,k}$ from edge node n to cooperative edge node

$n = 1, 2, \ldots, N\}$ and the vehicles are denoted by $\mathcal{L} = \{l \mid l = 1, 2, \ldots, L\}$. Here, $L = \sum_{n=1}^{N} \rho_n(t)$, where $\rho_n(t)$ denotes the vehicle density—i.e., the number of vehicles within the coverage area of edge node n during time slot t. Time is discretized into T equal-length slots, represented by the set $\mathcal{T} = \{t \mid t = 0, 1, \ldots, T-1\}$, where each time slot has a fixed duration Δt.

The system supports K types of services, denoted by $\mathcal{K} = \{k \mid k = 1, 2, \ldots, K\}$, and all services are available on the cloud server. At the beginning of each time slot, the number of vehicles within the coverage area of each edge node is updated. Each vehicle generates a task that depends on multiple types of services. During time slot t, vehicle l generates a task, denoted as task j. The task j is characterized by a tuple $(d_j(t), \mathbf{f}_j(t))$, where $d_j(t)$ represents the input data size of task j and $\mathbf{f}_j(t) = [f_{j,1}(t), f_{j,2}(t), \ldots, f_{j,K}(t)] \in [0, 1]^K$ denotes the service request vector. Specifically, $f_{j,k}(t)$ represents the proportion of the task associated with the required service of type k, satisfying $\sum_{k=1}^{K} f_{j,k}(t) = 1$. The effective service set of task j is defined as $\mathcal{K}_j = \{k | f_{j,k}(t) > 0\}$. Accordingly, task j is decomposed into $|\mathcal{K}_j|$ subtasks, each corresponding to a specific type of service. For simplicity, the type-k subtask of task j at time slot t is denoted as $s_{j,k}(t)$. Such a decomposition allows each subtask to be individually processed on an entity (vehicle, edge node, or cloud), thereby improving the overall resource utilization and flexibility of the system.

Subtasks can be processed locally on the vehicle or offloaded to the edge node that the vehicle is connected to, in order to optimize resource utilization. Due to the limited caching capacity, only a subset of all services can be deployed on vehicles and edge nodes. To effectively offload tasks, each edge node considers its neighboring edge nodes as collaborative edge nodes, which are indexed by m. Consequently, if neither the vehicle, the associated edge node, nor its collaborative nodes can fulfill the offloading requirements of the task, the task will be offloaded to the cloud server.

3.2. Communication Model

This study focuses on the uplink communication scenario, where each edge node is allocated a total bandwidth of B, with identical bandwidth assignments across all edge nodes. Given the orthogonality of the communication channels, intra-cell interference within the coverage area of a single edge node is neglected. The signal-to-interference-plus-noise rate (SINR) of the communication link between vehicle l and edge node n at time slot t is given by

$$\gamma_{l,n}(t) = \frac{p_l(t) g_{l,n}(t)}{\sum_{n=1}^{N} \left(g_{l,n}(t) \sum_{l=1}^{\rho_n(t)} p_l(t) \right) + \sigma^2}, \tag{1}$$

where $p_l(t)$ denotes the uplink transmission power of vehicle l, $g_{l,n}(t)$ represents the average channel gain between vehicle l and edge node n, and σ^2 is the variance of the additive white Gaussian noise.

Based on Shannon's theorem, the data transmission rate from vehicle l to edge node n is given by

$$r_{l,n}(t) = b_{l,n}(t) \log_2(1 + \gamma_{l,n}(t)), \tag{2}$$

where $b_{l,n}(t) = B/\rho_n(t)$ is the bandwidth allocated to vehicle l by edge node n.

3.3. Service Generation Model

To model the heterogeneous service requirements observed in VEC systems, the service type selection for each task is assumed to follow the Zipf distribution [28] with a skewness parameter $z > 0$. Specifically, the required probability P_k of type-k service is given by

$$P_k = \frac{\frac{1}{k^z}}{\sum_{k'=1}^{K} \frac{1}{k'^z}}. \tag{3}$$

For task j, the proportion of type-k service it requires is modeled by the Dirichlet distribution. This ensures that each proportion $f_{j,k}(t)$ is non-negative and that they all sum to one.

The Zipf and Dirichlet distributions not only capture the sparsity and bias in service generation but also simulate the diverse subtask generation model in VEC networks.

3.4. Service Caching and Task Offloading Model

We assume that a task is processed within its originating time slot. The agent, deployed on the cloud server, determines the caching strategy for all service types across vehicles and edge nodes prior to the end of each time slot.

The caching capacity constraint of vehicle l is given by

$$\beta \cdot \boldsymbol{\delta}_l^V(t)\mathbf{e} \leq S^V \tag{4}$$

and that of edge node n is given by

$$\beta \cdot \boldsymbol{\delta}_n^R(t)\mathbf{e} \leq S^R. \tag{5}$$

Here, parameter β represents the size of a service file. Vectors $\boldsymbol{\delta}_l^V(t)$ and $\boldsymbol{\delta}_n^R(t)$ denote the caching states of vehicle l and edge node n for all services, respectively. Each component of the vectors is a binary variable indicating the caching state for a specific service: 1 means the service is cached, while 0 indicates it is not.

To enhance subtask execution efficiency, the agent determines the offloading destinations and proportions for all subtasks at the start of each time slot based on the caching states of all entities. Let continuous variables $\omega_{l,j,k}^V(t)$ and $\omega_{n,j,k}^R(t)$ denote the offloading proportion of subtask $s_{j,k}(t)$ for vehicle-to-edge and edge-to-edge offloading, respectively. Meanwhile, let binary variables $\delta_{l,k}^V(t)$, $\delta_{n,k}^R(t)$, and $\delta_{m,k}^C(t)$ represent the caching states of the type-k service on vehicle l, edge node n, and cooperative edge node m, respectively. The offloading proportions of subtask $s_{j,k}(t)$ corresponding to different service caching states are summarized in Table 2.

The agent performs fine-grained subtask offloading for each task. Specifically, for the type-k service required by subtask $s_{j,k}(t)$, the agent first checks whether the service is cached locally. If so, the agent first confirms the service's availability at the associated edge node n before determining the optimal execution location—either local computation or partial offloading to an edge node—for each subtask. If the service is not cached at vehicle l, the subtask is fully uploaded to the associated edge node n. Based on the service caching states of edge node n and its cooperative edge nodes, the agent then decides whether to offload the entire subtask to node n, to a cooperative edge node, or partially to both. It is worth noting that when multiple neighbor edge nodes meet the requirements, one of them is randomly selected for cooperative edge node. If neither the vehicle, the edge node, nor the cooperative edge nodes cache the type-k service, the subtask is offloaded to the cloud server.

3.5. Computation Delay and Energy Consumption

Based on Secs. 3.2 and 3.4, within the coverage area of edge node n, the computation delay of subtask $s_{j,k}(t)$ is calculated as follows.

The execution delay locally is given by

$$D_{l,j,k}^{\text{local}}(t) = \delta_{l,k}^V(t)(1 - \omega_{l,j,k}^V(t))\frac{\lambda d_{j,k}(t)}{f^V}, \tag{6}$$

where f^V represents the CPU frequency of a vehicle and λ denotes the number of CPU cycles required to compute one bit of data.

The transmission delay from the vehicle to the edge node is given by

$$D_{l,j,k}^{\text{v2e}}(t) = \left(1 - \delta_{l,k}^V(t)(1 - \delta_{n,k}^R(t))\right)\omega_{l,j,k}^V(t)\frac{d_{j,k}(t)}{r_{l,n}(t)}. \tag{7}$$

The execution delay at the edge node is expressed as

$$D_{n,j,k}^{\text{edge}}(t) = \delta_{n,k}^R(t)\omega_{l,j,k}^V(t)(1 - \omega_{n,j,k}^R(t))\frac{\lambda d_{j,k}(t)}{f^R}, \tag{8}$$

where f^R denotes CPU frequency of an edge node.

The transmission delay between two adjacent edge nodes can be expressed as

$$D_{n,j,k}^{\text{e2e}}(t) = (1 - \delta_{l,k}^V(t))\delta_{m,k}^C(t)\omega_{l,j,k}^V(t)\omega_{n,j,k}^R(t)\frac{d_{j,k}(t)}{r^{\text{e2e}}}, \tag{9}$$

where r^{e2e} represents data transmission rate between edge nodes.

The execution delay at the cooperative edge node is expressed as

$$D_{m,j,k}^{\text{co}}(t) = (1 - \delta_{l,k}^V(t))\delta_{m,k}^C(t)\omega_{l,j,k}^V(t)\omega_{n,j,k}^R(t)\frac{\lambda d_{j,k}(t)}{f^R}. \tag{10}$$

The transmission delay from the edge node to the cloud server is given by

$$D_{n,j,k}^{\text{e2c}}(t) = \left(1 - \left(\delta_{l,k}^V(t) + \delta_{n,k}^R(t) + \delta_{m,k}^C(t)\right)\right)\frac{d_{j,k}(t)}{r^{\text{e2c}}} \tag{11}$$

where r^{e2c} denotes the data transmission rate for the edge-to-cloud link.

Since the computational results are substantially smaller than the raw input data, and the cloud server possesses abundant computing resources, the delay caused by result transmission and execution on the cloud server is negligible [34].

Considering all the service caching states in Table 2, the computation delay of subtask $s_{j,k}(t)$ can be expressed as

$$\begin{aligned} D_{j,k}^{\text{total}}(t) = \max\Bigg\{ & D_{l,j,k}^{\text{local}}(t), D_{l,j,k}^{\text{v2e}}(t) + \max\left\{D_{n,j,k}^{\text{edge}}(t),\right. \\ & \left. D_{n,j,k}^{\text{e2e}}(t) + D_{m,j,k}^{\text{co}}(t), D_{n,j,k}^{\text{e2c}}(t)\right\}\Bigg\}. \end{aligned} \tag{12}$$

The computation delay of task j at time slot t is defined as

$$D_j^{\text{total}}(t) = \sum_{k \in \mathcal{K}_j} D_{j,k}^{\text{total}}(t). \tag{13}$$

The average computation delay for processing a task at time slot t is defined as

$$D^{\text{ave}}(t) = \frac{1}{N}\sum_{n=1}^{N}\left(\frac{1}{\rho_n(t)}\sum_{j=1}^{\rho_n(t)} D_j^{\text{total}}(t)\right). \tag{14}$$

Referring to [22], we also consider MNOs' energy consumption in this study. The energy consumption for processing subtask $s_{j,k}(t)$ is given by

$$\begin{aligned} E_{j,k}^{\text{total}}(t) = & \kappa(f^R)^2\left(D_{n,j,k}^{\text{edge}}(t) + D_{m,j,k}^{\text{co}}(t)\right) \\ & + p^{\text{e2e}}D_{n,j,k}^{\text{e2e}} + p^{\text{e2c}}D_{n,j,k}^{\text{e2c}}(t). \end{aligned} \tag{15}$$

Here, parameter κ represents the processor energy efficiency parameter of the edge node, which typically depends on hardware

Table 2: Offloading proportion of a subtask

Service caching states		Proportion of subtask $s_{j,k}(t)$ to be offloaded to different entities			
		Vehicle l	Edge node n	Cooperative edge node m	Cloud server
M1:	$\delta^V_{l,k}(t)=1, \delta^R_{n,k}(t)=0, \delta^C_{m,k}(t)\geq 0$	1	0	0	0
M2:	$\delta^V_{l,k}(t)=1, \delta^R_{n,k}(t)=1, \delta^C_{m,k}(t)\geq 0$	$1-\omega^V_{l,j,k}(t)$	$\omega^V_{l,j,k}(t)$	0	0
M3:	$\delta^V_{l,k}(t)=0, \delta^R_{n,k}(t)=1, \delta^C_{m,k}(t)=0$	0	1	0	0
M4:	$\delta^V_{l,k}(t)=0, \delta^R_{n,k}(t)=1, \delta^C_{m,k}(t)=1$	0	$1-\omega^R_{n,j,k}(t)$	$\omega^R_{n,j,k}(t)$	0
M5:	$\delta^V_{l,k}(t)=0, \delta^R_{n,k}(t)=0, \delta^C_{m,k}(t)=1$	0	0	1	0
M6:	$\delta^V_{l,k}(t)=0, \delta^R_{n,k}(t)=0, \delta^C_{m,k}(t)=0$	0	0	0	1

conditions. Parameters p^{e2e} and p^{e2c} denote the edge-to-edge transmission power and edge-to-cloud transmission power, respectively.

Then the MNO's energy consumption for processing task j at time slot t is defined as

$$E^{\mathrm{total}}_j(t)=\sum_{k\in\mathcal{K}_j}E^{\mathrm{total}}_{j,k}(t). \tag{16}$$

From the perspective of the whole system, the average energy consumption for processing a task at time slot t is defined as

$$E^{\mathrm{ave}}(t)=\frac{1}{N}\sum_{n=1}^{N}\left(\frac{1}{\rho_n(t)}\sum_{j=1}^{\rho_n(t)}E^{\mathrm{total}}_j(t)\right). \tag{17}$$

3.6. Extremum Analysis of Computation Delay and Energy Consumption

We further evaluate the computation delay and energy consumption under different service caching states and derive their extremum values. The extremum analysis helps to understand the best and worst cases for subtask offloading and reveal the performance boundaries.

Within the coverage area of edge node n, the computation delay and MNOs' energy consumption for processing subtask $s_{j,k}(t)$ under different service caching states are discussed as follows.

M1: The subtask is computed locally.

$$D^{M1}(t)=\frac{\lambda d_{j,k}(t)}{f^V}, \tag{18}$$

$$E^{M1}(t)=0. \tag{19}$$

M2: A part of the subtask is computed locally, and the remaining part is offloaded to edge node.

$$\begin{aligned}D^{M2}(t)=\max\Bigg\{&(1-\omega^V_{l,j,k}(t))D^{M1},\\ &\omega^V_{l,j,k}(t)\left(\frac{d_{j,k}(t)}{r_{l,n}(t)}+\frac{\lambda d_{j,k}(t)}{f^R}\right)\Bigg\},\end{aligned} \tag{20}$$

$$E^{M2}(t)=\kappa f^R\omega^V_{l,j,k}(t)\lambda d_{j,k}(t). \tag{21}$$

M3: The subtask is fully offloaded to edge node.

$$D^{M3}(t)=\frac{d_{j,k}(t)}{r_{l,n}(t)}+\frac{\lambda d_{j,k}(t)}{f^R}, \tag{22}$$

$$E^{M3}(t)=\kappa f^R\lambda d_{j,k}(t). \tag{23}$$

M4: One part of the subtask is offloaded to the edge node, while the remainder is offloaded to the cooperative edge node.

$$\begin{aligned}D^{M4}(t)=&\frac{d_{j,k}(t)}{r_{l,n}(t)}+\max\Bigg\{\frac{(1-\omega^R_{n,j,k}(t))\lambda d_{j,k}(t)}{f^R},\\ &\omega^R_{n,j,k}(t)\left(\frac{d_{j,k}(t)}{r^{e2e}}+\frac{\lambda d_{j,k}(t)}{f^R}\right)\Bigg\}.\end{aligned} \tag{24}$$

$$E^{M4}(t)=\kappa f^R\lambda d_{j,k}(t)+\omega^R_{n,j,k}(t)p^{e2e}\frac{d_{j,k}(t)}{r^{e2e}}, \tag{25}$$

M5: The subtask is fully offloaded to cooperative edge node.

$$D^{M5}(t)=\frac{d_{j,k}(t)}{r_{l,n}(t)}+\frac{d_{j,k}(t)}{r^{e2e}}+\frac{\lambda d_{j,k}(t)}{f^R}, \tag{26}$$

$$E^{M5}(t)=\kappa f^R\lambda d_{j,k}(t)+p^{e2e}\frac{d_{j,k}(t)}{r^{e2e}}. \tag{27}$$

M6: The subtask is fully offloaded to cloud server.

$$D^{M6}(t) = \frac{d_{j,k}(t)}{r_{l,n}(t)} + \frac{d_{j,k}(t)}{r^{e2c}}, \tag{28}$$

$$E^{M6}(t) = p^{e2c}\frac{d_{j,k}(t)}{r^{e2c}}. \tag{29}$$

Considering the computation delays under different service caching states, the maximum computation delay for processing the subtask is readily derived as

$$\max\{D^{\text{total}}_{j,k}(t)\} = \max\{D^{\text{M1}}(t), D^{\text{M5}}(t), D^{\text{M6}}(t)\}, \tag{30}$$

and the minimum value can be derived as

$$\min\{D^{\text{total}}_{j,k}(t)\} = \min\{D^{\text{M2}}(t), D^{\text{M4}}(t), D^{\text{M6}}(t)\}. \tag{31}$$

According to convex piecewise optimization, the offloading proportion in Eqs. (20) and (24) can be eliminated, i.e., when $\omega^V_{l,j,k}(t) = 1/(1 + \frac{f^V}{\lambda r_{l,n}(t)} + \frac{f^V}{f^R})$, the minimum value of $D^{\text{M2}}(t)$ can be expressed as

$$\min\{D^{\text{M2}}(t)\} = \frac{\lambda d_{j,k}(t)\left(f^R + \lambda r_{l,n}(t)\right)}{\lambda f^R f^V + f^R r_{l,n}(t) + \lambda r_{l,n}(t) f^V}, \tag{32}$$

and when $\omega^R_{n,j,k}(t) = 1/(2 + \frac{f^R}{\lambda r^{e2e}})$, the minimum value of $D^{M4}(t)$ can be expressed as

$$\min\{D^{\text{M4}}(t)\} = \frac{d_{j,k}(t)}{r_{l,n}(t)} + \frac{\lambda d_{j,k}(t)\left(f^R + \lambda r^{\text{e2e}}\right)}{f^R\left(f^R + 2\lambda r^{\text{e2e}}\right)}. \tag{33}$$

Combing Eqs. (32) and (33), the minimum value of the computation delay for processing the subtask can be expressed as

$$\begin{aligned}\min\{D^{\text{total}}_j(t)\} = \min\Big\{ &\min\{D^{\text{M2}}(t)\}, \min\{D^{\text{M4}}(t)\}, \\ &D^{\text{M6}}(t)\Big\}.\end{aligned} \tag{34}$$

Considering all the energy consumption values under different service caching states, the maximum value of the energy consumption for processing the subtask is readily derived as

$$\max\{E^{\text{total}}_j(t)\} = \max\left\{E^{\text{M5}}(t), E^{\text{M6}}(t)\right\}. \tag{35}$$

It should be noted that if a task is executed locally, the MNOs consume no energy, i.e., $\min\{E^{\text{total}}_j(t)\} = E^{\text{M1}}(t) = 0$.

According to Eqs. (30) and (34), the computation delay is closely related to factors such as the data size of a subtask and the computing resources of an edge node. The maximum computation delay occurs under M1, M5, or M6, while the minimum computation delay occurs under M2, M4, or M6. Meanwhile, Eq. (35) shows that the maximum energy consumption occurs under M5 or M6. Although the participation of edge nodes increases energy consumption, if the agent is more likely to select M2 or M4, the computation delay can be reduced effectively while keeping MNOs' energy consumption at a low level.

3.7. Problem Formulation

To minimize the average computation delay, the joint problem of service caching and task offloading is formulated as a MINLP optimization problem. Let $\mathbf{Z}(t)$ denote the decision at time slot t, i.e., $\mathbf{Z}(t) = [\omega^V_{l,j,k}(t), \omega^R_{n,j,k}(t), X^V_{l,k}(t), X^R_{n,k}(t)]$. The detailed problem formulation is presented as follows.

$$\min_{\mathbf{Z}(t)} \quad \sum_{t=0}^{T-1} D^{\text{ave}}(t), \tag{36}$$

s.t.

$$\forall l \in \mathcal{L}, \forall n \in \mathcal{N}, \forall k \in \mathcal{K}, \forall t \in \mathcal{T},$$

$$\omega^V_{l,j,k}(t) \in [0,1], \tag{36a}$$

$$\omega^R_{n,j,k}(t) \in [0,1], \tag{36b}$$

$$X^V_{l,k}(t) \in \{0,1\}, \tag{36c}$$

$$X^R_{n,k}(t) \in \{0,1\}, \tag{36d}$$

$$\beta \cdot \boldsymbol{\delta}^V_l(t)\mathbf{e} \le S^V, \tag{36e}$$

$$\beta \cdot \boldsymbol{\delta}^R_n(t)\mathbf{e} \le S^R. \tag{36f}$$

$$E^{\text{total}}_j(t) < E^{\max}, \tag{36g}$$

In Eq. (36), constraints (36a) and (36b) ensure that the offloading proportions of a subtask are continuous values within the range of $[0, 1]$. Parameters $X^V_{l,k}(t)$ and $X^R_{n,k}(t)$ denote the caching decisions of vehicle l and edge node n for type-k service, respectively, at time slot t. Constraints (36c) and (36d) specify that the caching states of type-k service at a vehicle or an edge node are binary. Constraints (36e) and (36f) ensure that the cached services do not exceed the storage capacity of a vehicle or an edge node. Constraint (36g) restricts the energy consumption to be less than $E^{\max}$, where $E^{\max}$ is the maximum value of energy consumption under different service caching states.

4. The Proposed DRLSCTO Scheme

Traditional model-based approaches struggle to adapt to dynamic environments and heuristic algorithms lack generality for unpredictable VEC scenarios. For this reason, we employ the DDPG algorithm—an actor-critic method that integrates deep neural networks with reinforcement learning—to optimize the joint caching and offloading problem. The algorithm enables an agent to interact with the environment and make decisions based on experiences, making it especially suited for high-dimensional systems like VEC [35]. Based on this formulation, we model the optimization problem presented in (36) as an MDP and propose DRLSCTO, a DDPG-based scheme for joint service caching and task offloading, designed to jointly optimize caching and offloading decisions.

4.1. Problem Formulation Based on RL

The Markov property fundamentally defines an MDP, establishing that future states are determined exclusively by the current state and remain conditionally independent of all preceding historical states. The Markov property provides a structure for

handling sequential decisions in the stochastic VEC environment and consequently enables policy learning through system interaction. A typical MDP consists of three key elements: a state space $\mathbf{S}$ representing system conditions, an action space $\mathbf{A}$ defining decisions, and a reward function r that evaluates actions and guides learning. Let $\mathbf{S}(t)$ and $\mathbf{A}(t)$ denote the state and action spaces at time slot t, respectively, with s_t and a_t representing a sample from each. The state space, action space, and reward are defined as follows.

4.1.1. State space

According to the models proposed in Secs. 3.2 and 3.4, the state space at time slot t includes the following components:

- $\boldsymbol{\delta}^{\mathrm{V}}(t)$: The service caching states of all vehicles.
- $\boldsymbol{\delta}^{\mathrm{R}}(t)$: The service caching states of all edge nodes.
- $\mathbf{d}(t)$: The data size of all tasks.
- $\mathbf{b}(t)$: The bandwidth allocated to all vehicles.
- $\boldsymbol{\gamma}(t)$: The received SINR of all edge nodes.

The state space $\mathbf{S}(t)$ at time slot t can be expressed as

$$\begin{aligned}\mathbf{S}(t) = \Big\{&[\boldsymbol{\delta}^{\mathrm{V}}(t)]^{L\times K}, [\boldsymbol{\delta}^{\mathrm{R}}(t)]^{N\times K}, [\mathbf{d}(t)]^{L},\\ &[\mathbf{b}(t)]^{L}, [\boldsymbol{\gamma}(t)]^{L}\Big\}.\end{aligned} \tag{37}$$

4.1.2. Action space

With the state observed at the beginning of a time slot, the agent determines offloading locations and offloading proportions, while also updating the service caching states. The action space at time slot t includes the following components:

- $\mathbf{X}^{\mathrm{V}}(t)$: The service caching decisions of all vehicles.
- $\mathbf{X}^{\mathrm{R}}(t)$: The service caching decisions of all edge nodes.
- $\boldsymbol{\omega}^{\mathrm{V}}(t)$: The offloading proportions of all subtasks from vehicles to edge nodes.
- $\boldsymbol{\omega}^{\mathrm{R}}(t)$: The offloading proportion of all subtasks from edge nodes to cooperative edge nodes.

The action space $\mathbf{A}(t)$ at time slot t can be expressed as

$$\begin{aligned}\mathbf{A}(t) = \Big\{&\mathbf{X}^{\mathrm{V}}(t)]^{L\times K}, [\mathbf{X}^{\mathrm{R}}(t)]^{N\times K},\\ &[\boldsymbol{\omega}^{\mathrm{V}}(t)]^{L}, [\boldsymbol{\omega}^{\mathrm{R}}(t)]^{L}\Big\}.\end{aligned} \tag{38}$$

4.1.3. Reward function

Under the reinforcement learning framework, the agent's policy is updated by maximizing the cumulative reward. To align the goal of computation delay minimization with the mechanism of reward maximization, we define the reward function at a time slot as the negative value of the instantaneous average computation delay. Thereby, minimizing the average computation delay is equivalent to maximizing the cumulative reward. Given state s_t and action a_t, the reward r_t at time slot t is expressed as

$$r_t = r(s_t, a_t) = -D^{ave}(t). \tag{39}$$

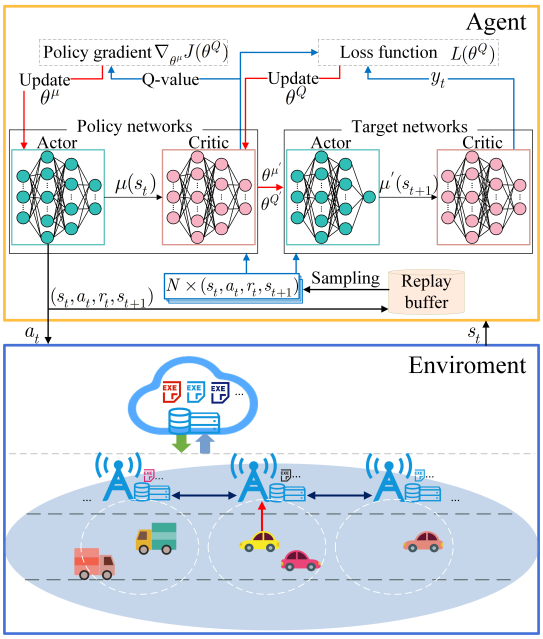

Figure 2: The framework of the DRLSCTO.

4.2. The DRLSCTO Scheme

To effectively learn and converge to the optimal policy, we propose the DRLSCTO scheme. The scheme incorporates policy networks, target networks, and an experience replay buffer. These components work in concert to stabilize the training process and ensure the agent's policy progressively converges through iterative interactions with the environment.The framework of the DRLSCTO scheme is illustrated in Fig. 2.

As shown in Fig. 2, the actor network of the policy networks implements the deterministic policy function $\mu(s_t|\theta^\mu)$, which maps the current state s_t to a deterministic action a_t, i.e., $a_t = \mu(s_t|\theta^\mu)$. The critic network of the policy networks estimates the Q-value $Q(s_t, a_t|\theta^Q)$, obtained by taking action a_t in state s_t, which represents the expected cumulative discounted reward and can be expressed as

$$Q(s_t, \mu(s_t|\theta^\mu)|\theta^Q) = \mathbb{E}_{s_{t+1}\sim \mathrm{P}}\left[\sum_{b=0}^{\infty} \alpha^b r_{t+a}\right]. \tag{40}$$

Here, parameter α is the discount factor, while θ^μ and θ^Q represent the actor network and critic network of the policy network, respectively.

Eq. (40) adheres to a recursive relationship known as the Bellman equation, which can be written as

$$\begin{aligned}Q(s_t, \mu(s_t|\theta^\mu)|\theta^Q) =&\mathbb{E}_{s_{t+1}\sim \mathrm{P}}[\alpha Q(s_{t+1}, \mu(s_{t+1}|\theta^\mu)|\theta^Q)\\ &+ r_t].\end{aligned} \tag{41}$$

Although the state transition probability is not explicitly modeled in this problem, the transition is implicitly defined and implemented by the simulation environment. Therefore, the expectation $\mathbb{E}_{s_{t+1}\sim \mathrm{P}}$ is still used in the theoretical formulation to preserve the mathematical completeness and standard representation of the Bellman equation.

To enhance training stability, the target Q-value is constructed by replacing the Q-value and the policy with their target networks' counterparts. The target Q-value y_t is defined as

$$y_t = r_t + \alpha Q'(s_{t+1}, \mu(s_{t+1}|\theta^{\mu'})|\theta^{Q'}). \tag{42}$$

Here, parameters $\theta^{\mu'}$ and $\theta^{Q'}$ represent the actor network and critic network of the target networks, respectively, and they are soft updated by slowly tracking the policy networks.

Since DDPG adopts a deterministic policy, the agent may become trapped in local optima during early training due to insufficient exploration. To encourage exploration, a noise term n_t is added to the action a_t. The modified action a_t is defined as

$$a_t = \mu(s_t|\theta^{\mu}) + n_t. \tag{43}$$

To reduce the correlation between consecutive experiences, DDPG employs a replay buffer to store interaction tuples of the form (s_t, a_t, r_t, s_{t+1}). Each tuple represents a transition, capturing the outcome of taking action a_t in state s_t. When the agent takes action a_t and then receives r_t and s_{t+1}, the transition (s_t, a_t, r_t, s_{t+1}) is stored in the experience replay buffer. During training, batches of data randomly sampled from this buffer are utilized to update parameters θ^{Q} and θ^{μ}.

θ^{Q} is optimized by minimizing the mean squared error (MSE) between the Q-value and its corresponding target. The loss function is formulated as follows:

$$L(\theta^{Q}) = \frac{1}{I}\sum_{i=1}^{I}\left(y_i - Q\left(s_i, \mu\left(s_i|\theta^{\mu}\right)|\theta^{Q}\right)\right)^2, \tag{44}$$

where I is the mini-batch size sampled from the experience replay buffer.

θ^{μ} is updated by following the deterministic policy gradient to maximize the Q-value. The gradient update can be expressed as

$$\begin{aligned}\nabla_{\theta^{\mu}} J(\theta^{Q}) = &\frac{1}{I}\sum_{i=1}^{I}\left[\nabla_a Q\left(s, a|\theta^{Q}\right)|_{s=s_i, a=\mu(s_i|\theta^{\mu})}\right. \\ &\left.\cdot\nabla_{\theta^{\mu}}\mu(s|\theta^{\mu})|_{s=s_i}\right].\end{aligned} \tag{45}$$

At each iteration, θ^{Q} is updated according to

$$\theta^{Q} \leftarrow \theta^{Q} - \eta \cdot \nabla_{\theta^{Q}}(L(\theta^{Q})) \tag{46}$$

and θ^{μ} is updated according to

$$\theta^{\mu} \leftarrow \theta^{\mu} + \eta \cdot \nabla_{\theta^{\mu}}(J(\theta^{\mu})), \tag{47}$$

where η denotes the learning rate.

The target networks are not directly copied from the policy networks but updated using a soft update mechanism, which slowly tracks θ^{Q} and θ^{μ}. At each iteration, $\theta^{Q'}$ is updated according to

$$\theta^{Q'} \leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q'} \tag{48}$$

and $\theta^{\mu'}$ is updated according to

$$\theta^{\mu'} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'}. \tag{49}$$

Here, $0 < \tau \ll 1$ is a smoothing factor, which prevents excessive fluctuations in the target Q-value during learning.

The DRLSCTO scheme is detailed in Algorithm 1.

Algorithm 1 The DRLSCTO Scheme for Joint Service Caching and Task Offloading.

Input: $\mathcal{L}, \mathcal{N}, \mathcal{T}, \mathcal{K}$.
Output: θ^{μ}.

```
Initialization: θ^μ, θ^Q, θ^μ', θ^Q', the soft update coefficient τ, the number of episodes, the batch sample size the experience replay buffer, the random noise n_t.
for each episode do
    Initialize state s_0 at time slot 0.
    for t = 0, 1, ..., T − 1 do
        The agent selects action a_t according to Eq. (43).
        Vehicles and edge nodes execute action a_t.
        The agent observes the next state s_{t+1} and receives an immediate reward r_t.
        Store transition (s_t, a_t, r_t, s_{t+1}) in experience replay buffer.
        if the experience replay buffer size exceeds a predefined threshold then
            Randomly sample a mini-batch of i transitions (s, a, r, s_next) from the experience replay buffer.
            Calculate Q-value by Eq. (40).
            Calculate target Q-value by Eq. (42).
            Update θ^Q by Eq. (44).
            Update θ^μ by Eq. (45).
            Perform a soft update on θ^Q' and θ^μ' according to Eqs. (48) and (49), respectively.
        end if
    end for
end for
```

5. Simulation Experiments and Performance Analysis

We conduct simulation experiments under different parameter settings to evaluate the performance of the proposed DRLSCTO scheme. The parameters in the VEC environment refer to [36, 37], and the main parameters have been summarized in Table 3.

We select four caching and offloading benchmark schemes to evaluate DRLSCTO. The descriptions of benchmark schemes are as follows.

1. No comparative edge nodes (NCE) [38]: This scheme disables cooperation among edge nodes, restricting subtask execution to three locations: local vehicle, edge node, and cloud server.
2. Least Recently Used (LRU)-based service caching replacement on vehicles and edge nodes (LRUSC) [39]: The offloading decisions for both vehicles and edge nodes are made by a agent. The LRU policy is used to update the cache when the storage capacity is insufficient.
3. Random service caching and task offloading (RSCTO) [40]: The caching decisions for both vehicles and edge nodes, along with the offloading proportions for subtasks, are all generated through a random process.
4. No edge nodes (NE) [30]: Each subtask can only be processed locally or offloaded to a cloud server. The caching and offloading decisons are made by a agent.

Table 3: Main parameters

Parameter	Meaning	Value
η^{actor}	The learning rate of actor networks	0.001
η^{critic}	The learning rate of critic networks	0.002
α	The discount factor	0.99
$\rho^{\max}$	The maximal vehicle density	10
N	The number of edge nodes	3
K	The number of service types	3
S^V	The storage capacity of a vehicle	30 Mb
S^R	The storage capacity of an edge server	60 Mb
r^{e2e}	The transmission rate between adjacent edge nodes	10 Mbps
r^{e2c}	The transmission rate from an edge node to cloud server	8 Mbps
p^{e2e}	The transmission power between adjacent edge nodes	1 W
p^{e2c}	The transmission power from an edge node to cloud server	2 W

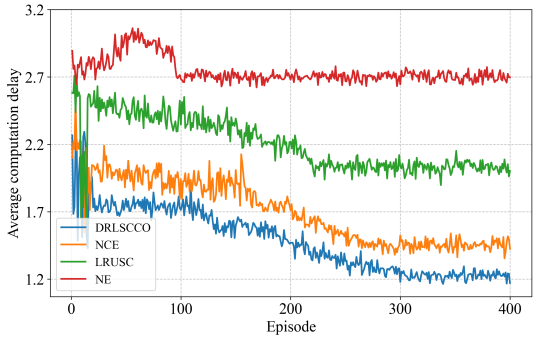

Figure 3: The average computation delay per episode

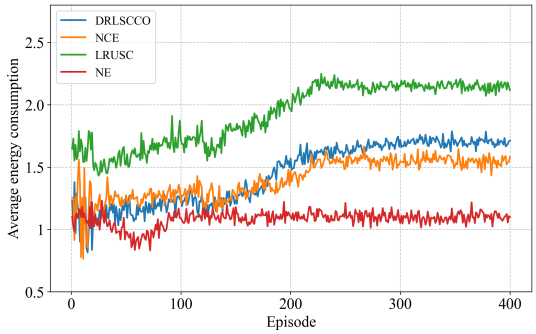

Figure 4: The average energy consumption per episode

Figure 3 compares the average computation delay per training episode across schemes involving agent participation. Figure 4 further illustrates the corresponding average energy consumption per training episode across these schemes.

As shown in Figs. 3 and 4, the average computation delay exhibits a decreasing trend with eventual convergence over successive episodes, whereas the average energy consumption rises correspondingly before reaching a stable level. These trends occur because deploying edge nodes and cloud server reduces execution delay but consumes more energy. NCE lacks cooperative caching and offloading, forcing subtasks to be offloaded to the cloud server when edge nodes are unavailable, which increases transmission delay. LRUSC, relying solely on historical access patterns, adopts a localized caching strategy without the global optimization capability. This limitation in the caching strategy consequently leads to inferior offloading decisions and higher transmission delay. In the case of NE, where edge nodes are absent, the average computation delay is dominated by local execution delay and cloud transmission delay, resulting in higher average computation delay and revealing the critical role of edge computing. These results confirm that the proposed DRLSCTO scheme consistently optimizes both caching and offloading, achieving the lowest delay while maintaining energy consumption growth within an acceptable range. The following experiments assess the performance by evaluating computation delay and energy consumption in the final time slot of each episode, utilizing models trained under specific parameter configurations.

By fixing the vehicle density at 10, Figure 5 explores the impact of task data size on the total computation delay across all schemes. This configuration evaluates total computation delay under growing computational demand, excluding interference from network scale variations and resource availability.

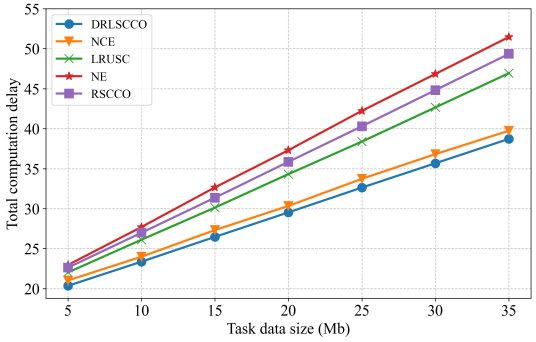

Figure 5: The impact of task data size on total computation delay

From Fig. 5, it can be clearly observed that as the data size increases from 5 Mb to 35 Mb, the total computation delay grows monotonically across all schemes. This growth is directly attributable to the positive correlation between execution delay

and data size. When the data size is smaller, the computation delays in all schemes are relatively close. As the data size increases, the delay differences among the schemes become more pronounced. The underlying reason for this phenomenon lies in the shift from a resource-abundant to a resource-constrained state as the data size grows. For smaller data size, the computing resources are underutilized, masking the deficiencies of LRUSC, NE and RSCTO. Under heavier load, however, intelligent resource management becomes critical. DRLSCTO excels in this context by continuously learning to optimize caching and offloading decisions. By making globally efficient decisions, DRLSCTO achieves the lowest total computation delay under different levels of computational demand.

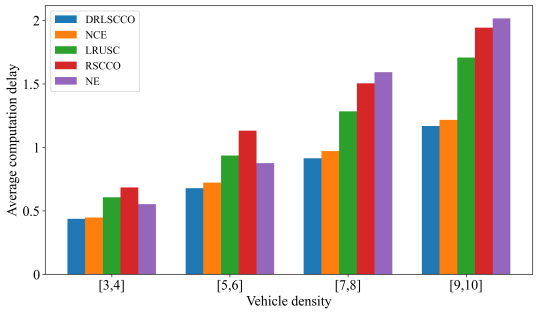

Figure 6: The impact of vehicle density on average computation delay

To isolate the impact of network scale, Figure 6 presents an investigation into how vehicle density affects the average computation delay by fixing the task data size at 20 Mb. This setup allows us to examine average computation delay of each scheme as the number of communication links increases.

As shown in Fig. 6, the results demonstrate that as vehicle density increases, the average computation delay of all schemes rises. This is because the increased vehicle density imposes higher demand on the communication channel, which reduces the bandwidth allocated to each vehicle and thereby leads to higher transmission delay. When the vehicle density is relatively low, each vehicle is allocated more bandwidth for task transmission, prompting the agent to offload tasks to the cloud server with higher computing capabilities and thus reducing average computation delay. Consequently, in low-density scenarios, NE exhibits a lower average computation delay compared to LRUSC and RSCTO, but this advantage vanishes as vehicle density increases. Notably, DRLSCTO consistently achieves the lowest average computation delay as vehicle density increases, highlighting its strong adaptability and effectiveness across different network scales.

With the vehicle density and task data size fixed at 10 and 20 Mb, respectively, we investigate the trends of total computation delay and energy consumption as the edge node CPU frequency increases, as shown in Figs. 7 and 8.

The results in Figs. 7 and 8 align with our expectations, i.e., higher CPU frequency enables lower total computation delay but leads to increased energy consumption. Since NE does not utilize edge nodes, its total computation delay and energy consumption remain constant. Both LRUSC and RSCTO exhibit considerable change in total computation delay and energy consumption. However, as the offloading decisions of LRUSC are governed by an agent, its performance trend demonstrates relative stability compared to RSCTO. An observation is that LRUSC and RSCTO exhibit higher computation delays than NE at a lower CPU frequency. This phenomenon indicates that the transmission delay associated with offloading a subtask to the cloud server may be shorter than the delay incurred by executing the subtask on an edge node with limited computing resources. This also suggests that in order to fully realize the advantages of edge computing, MNOs need to deploy edge nodes with abundant computing resources. The computation delays of DRLSCTO and NCE are relatively close at a lower CPU frequency. As the CPU frequency increases, the delay gap between DRLSCTO and NCE gradually widens. This is because with limited computing resources, the cooperative offloading in DRLSCTO introduces extra transmission delay. When computing resources become sufficient, the delay is increasingly offset by the growing advantages of cooperative offloading. Although DRLSCTO incurs higher energy consumption than NCE due to its cooperative offloading mechanism, the growth rate of energy consumption remains close to that of NCE. This controlled energy consumption is a feasible cost for the substantial computation delay reduction, enhancing QoE. The results show that as the CPU frequency increases, DRLSCTO consistently maintains the lowest total computation delay across all schemes while restricting the energy consumption within a reasonable range.

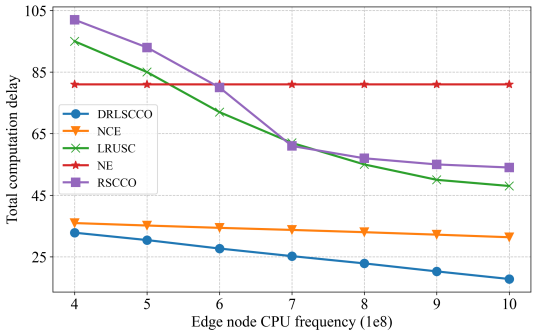

Figure 7: The influence of edge nodes CPU frequency on computation delay

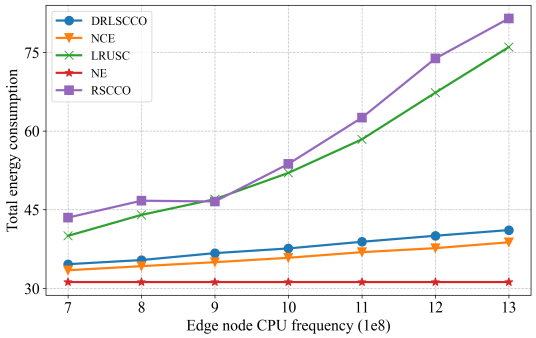

Figure 8: The influence of edge node CPU frequency on energy consumption

6. Conclusion

This paper addresses the joint service caching and task offloading problem in a VEC system, where a single task can concurrently request multiple types of services. To minimize the computation delay of such tasks, we formulate this joint optimization problem as an MDP and propose a DRL-based scheme named DRLSCTO. The scheme is designed to support heterogeneous service requirements, where a single task can request multiple types of services concurrently. DRLSCTO also enables service-type-based task decomposition and distributed subtask offloading. Furthermore, we introduce six caching and offloading states and conduct extremum analysis of task computation delay and energy consumption, providing a theoretical foundation for decision-making. The training results show that both task computation delay and energy consumption gradually stabilize as the caching and offloading decisions are updated. By selecting four benchmark schemes—NCE, LRUSC, RSCTO, and NE—we carry out comparative experiments under various task data sizes, vehicle densities, and edge node CPU frequencies. Experimental results demonstrate that DRLSCTO achieves the lowest task computation delay while maintaining reasonable energy consumption compared to the benchmark schemes.

References

[1] M. Talebkhah, A. Sali, V. Khodamoradi, T. Khodadadi, M. Gordan, Task offloading for edge-IoV networks in the Industry 4.0 era and beyond: A high-level view, Engineering Science and Technology 54 (2024) 101699. doi:10.1016/j.jestch.2024.101699.

[2] H. Wu, J. Zheng, S. Jin, Adaptive computation offloading scheme based on a collaborative architecture with heterogeneous MEC nodes: A DRL approach, IEEE Transactions on Mobile Computing 24 (11) (2025) 12692–12710. doi:10.1109/TMC.2025.3586623.

[3] L. Zhu, B. Li, L. Tan, Vehicular edge cloud computing content caching optimization solution based on content prediction and deep reinforcement learning, Ad Hoc Networks 165 (2024) 103643. doi:10.1016/j.adhoc.2024.103643.

[4] M. Reiss-Mirzaei, M. Ghobaei-Arani, L. Esmaeili, A review on the edge caching mechanisms in the mobile edge computing: A social-aware perspective, Internet of Things 22 (2023) 100690. doi:10.1016/j.iot.2023.100690.

[5] H. Yan, H. Li, X. Xu, M. Bilal, UAV-enhanced service caching for IoT systems in extreme environments, IEEE Internet of Things Journal 11 (16) (2024) 26741–26750. doi:10.1109/JIOT.2023.3288200.

[6] X. Ren, X. Chen, L. Jiao, X. Dai, Z. Dong, Joint optimization of trajectory, caching and task offloading for multi-tier UAV MEC networks, in: IEEE Wireless Communications and Networking Conference (WCNC), IEEE, 2024, pp. 1–6. doi:10.1109/WCNC57260.2024.10570765.

[7] B. Tolba, M. Abo-Zahhad, M. Elsabrouty, A. Uchiyama, A. H. A. El-Malek, Joint user association, service caching, and task offloading in multi-tier communication/multi-tier edge computing heterogeneous networks, Ad Hoc Networks 160 (2024) 103500. doi:10.1016/j.adhoc.2024.103500.

[8] W. Shi, J. Cao, Q. Zhang, Y. Li, L. Xu, Edge computing: Vision and challenges, IEEE Internet of Things Journal 3 (5) (2016) 637–646. doi:10.1109/JIOT.2016.2579198.

[9] D. L. Moura, A. L. Aquino, A. A. Loureiro, An edge computing and distributed ledger technology architecture for secure and efficient transportation, Ad Hoc Networks 164 (2024) 103633. doi:10.1016/j.adhoc.2024.103633.

[10] J. Lin, L. Huang, H. Zhang, X. Yang, P. Zhao, A novel Lyapunov-based dynamic resource allocation for UAVs-assisted edge computing, Computer Networks 205 (2022) 108710. doi:10.1016/j.comnet.2021.108710.

[11] L. Zhang, Y. Sun, Z. Chen, S. Roy, Communications-caching-computing resource allocation for bidirectional data computation in mobile edge networks, IEEE Transactions on Communications 69 (3) (2021) 1496–1509. doi:10.1109/TCOMM.2020.3041343.

[12] M. Wen, X. Liu, X. Ning, C. Liu, X. Chen, J. Nian, L. Cheng, Deep reinforcement learning for energy-efficient workflow scheduling in edge computing, Computer Networks (2025) 111790doi:10.1016/j.comnet.2025.111790.

[13] R. Zhang, H. Xia, Z. Chen, Z. Kang, K. Wang, W. Gao, Computation cost-driven offloading strategy based on reinforcement learning for consumer devices, IEEE Transactions on Consumer Electronics 70 (1) (2024) 4120–4131. doi:10.1109/TCE.2024.3357459.

[14] C. Zeng, X. Wang, R. Zeng, Y. Li, J. Shi, M. Huang, Joint optimization of multi-dimensional resource allocation and task offloading for QoE enhancement in cloud-edge-end collaboration, Future Generation Computer Systems 155 (2024) 121–131. doi:10.1016/j.future.2024.01.025.

[15] R. Xie, J. Fang, J. Yao, X. Jia, K. Wu, Sharing-aware task offloading of remote rendering for interactive applications in mobile edge computing, IEEE Transactions on Cloud Computing 11 (1) (2023) 997–1010. doi:10.1109/TCC.2021.3127345.

[16] M. Tang, V. W. S. Wong, Deep reinforcement learning for task offloading in mobile edge computing systems, IEEE Transactions on Mobile Computing 21 (6) (2022) 1985–1997. doi:10.1109/TMC.2020.3036871.

[17] S. Bounaira, A. Alioua, I. Souici, Blockchain-enabled trust management for secure content caching in mobile edge computing using deep reinforcement learning, Internet of Things 25 (2024) 101081. doi:10.1016/j.iot.2024.101081.

[18] J. Zhou, F. Chen, Q. He, X. Xia, R. Wang, Y. Xiang, Data caching optimization with fairness in mobile edge computing, IEEE Transactions on Services Computing 16 (3) (2023) 1750–1762. doi:10.1109/TSC.2022.3197881.

[19] C. Tang, Y. Ding, S. Xiao, H. Wu, R. Li, Joint optimization of service caching task offloading and resource allocation in cloud-edge cooperative network, in: IEEE International Conference on Communications (ICC), IEEE, 2024, pp. 4036–4041. doi:10.1109/ICC51166.2024.10622677.

[20] Y. Mao, B. He, S. Zhou, C. Ma, Z. Wang, Collaborative edge caching: A meta reinforcement learning approach with edge sampling, in: IEEE International Conference on Multimedia and Expo (ICME), IEEE, 2023, pp. 972–977.

[21] M. Zhao, M. R. Nakhai, Deep reinforcement learning based two-phase proactive caching for collaborative edge networks, in: IEEE Wireless Communications and Networking Conference (WCNC), IEEE, 2024, pp. 1–6.

[22] X. Kong, G. Duan, M. Hou, G. Shen, H. Wang, X. Yan, M. Collotta, Deep reinforcement learning-based energy-efficient edge computing for Internet of Vehicles, IEEE Transactions on Industrial Informatics 18 (9) (2022) 6308–6316. doi:10.1109/TII.2022.3155162.

[23] C. Li, K. Xiao, J. Xu, W. Ji, Parked-vehicle-assisted task offloading in vehicular edge computing: A Stackelberg game approach, Computer Networks (2025) 111800doi:10.1016/j.comnet.2025.111800.

[24] C.-L. Chen, B. Bhargava, V. Aggarwal, B. Tonshal, A. Gopal, A hybrid deep reinforcement learning approach for jointly optimizing offloading and resource management in vehicular networks, IEEE Transactions on Vehicular Technology 73 (2) (2024) 2456–2467. doi:10.1109/TVT.2023.3312340.

[25] H. Wu, Y. Fan, J. Jin, H. Ma, L. Xing, Social-aware decentralized cooperative caching for Internet of Vehicles, IEEE Internet of Things Journal 10 (16) (2023) 14834–14845. doi:10.1109/JIOT.2022.3229009.

[26] H. Wu, B. Wang, H. Ma, X. Zhang, L. Xing, Multiagent federated deep-reinforcement-learning-based collaborative caching strategy for vehicular edge networks, IEEE Internet of Things Journal 11 (14) (2024) 25198–25212. doi:10.1109/JIOT.2024.3392329.

[27] S. Yu, X. Gong, Q. Shi, X. Wang, X. Chen, EC-SAGINs: Edge-computing-enhanced space-air-ground-integrated networks for Internet of Vehicles, IEEE Internet of Things Journal 9 (8) (2022) 5742–5754. doi:10.1109/JIOT.2021.3052542.

[28] G. Yu, J. Wu, R. Liu, Y. He, Z. Chen, J. Pan, Joint cooperative caching and UAV trajectory optimization based on mobility prediction in the Internet of Connected Vehicles, IEEE Transactions on Intelligent Transportation Systems 25 (11) (2024) 17392–17406. doi:10.1109/TITS.2024.3429305.

[29] Z. Liu, L. Gao, Z. Ma, J. Su, F. Li, Y. Yuan, X. Guan, Joint task

offloading and resource allocation scheme with UAV assistance in vehicle edge computing networks, Computer Networks 273 (2025) 111746. doi:10.1016/j.comnet.2025.111746.

[30] Z. Xue, C. Liu, C. Liao, G. Han, Z. Sheng, Joint service caching and computation offloading scheme based on deep reinforcement learning in vehicular edge computing systems, IEEE Transactions on Vehicular Technology 72 (5) (2023) 6709–6722. doi:10.1109/TVT.2023.3234336.

[31] L. Liu, Z. Chen, Joint optimization of multiuser computation offloading and wireless-caching resource allocation with linearly related requests in a vehicular edge computing system, IEEE Internet of Things Journal 11 (1) (2024) 1534–1547. doi:10.1109/JIOT.2023.3289994.

[32] C. Cheng, L. Zhai, X. Zhu, Y. Jia, Y. Li, Dynamic task offloading and service caching based on game theory in vehicular edge computing networks, Computer Communications 224 (2024) 29–41. doi:10.1016/j.comcom.2024.05.020.

[33] C. Ling, W. Zhang, Q. Fan, Z. Feng, J. Wang, R. Yadav, D. Wang, Cooperative service caching in vehicular edge computing networks based on transportation correlation analysis, IEEE Internet of Things Journal 11 (12) (2024) 22754–22767. doi:10.1109/JIOT.2024.3382723.

[34] J. Wu, J. Wang, Q. Chen, Z. Yuan, P. Zhou, X. Wang, C. Fu, Resource allocation for delay-sensitive vehicle-to-multi-edges (V2Es) communications in vehicular networks: A multi-agent deep reinforcement learning approach, IEEE Transactions on Network Science and Engineering 8 (2) (2021) 1873–1886. doi:10.1109/TNSE.2021.3075530.

[35] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, M. Riedmiller, Deterministic policy gradient algorithms, Proceedings of Machine Learning Research 32 (2014) 387–395.

[36] S. Bi, L. Huang, Y.-J. A. Zhang, Joint optimization of service caching placement and computation offloading in mobile edge computing systems, IEEE Transactions on Wireless Communications 19 (7) (2020) 4947–4963. doi:10.1109/TWC.2020.2988386.

[37] G. Zhang, S. Zhang, W. Zhang, Z. Shen, L. Wang, Joint service caching, computation offloading and resource allocation in mobile edge computing systems, IEEE Transactions on Wireless Communications 20 (8) (2021) 5288–5300. doi:10.1109/TWC.2021.3066650.

[38] T. C. Lam, N.-S. Vo, V. V. Lam, T. Hoang, M.-P. Bui, T. Q. Duong, Multi-rate selection and power allocation assisted probabilistic edge caching for cooperative video transmission in dense D2D networks, Alexandria Engineering Journal 126 (2025) 555–564. doi:10.1016/j.aej.2025.04.057.

[39] G. Liu, Z. Qian, G. Li, Proactive retention-aware online video caching scheme in mobile edge computing, Computer Communications 242 (2025) 108313. doi:10.1016/j.comcom.2025.108313.

[40] P. Benedicte, C. Hernandez, J. Abella, F. J. Cazorla, Locality-aware cache random replacement policies, Journal of Systems Architecture 93 (2019) 48–61. doi:10.1016/j.sysarc.2018.12.007.