Deep Reinforcement Learning-based Joint Caching and Offloading Scheme in Vehicular Edge Computing Systems for Tasks With Heterogeneous Service Requirements

Jiaxuan Wang[a]

[a] *Yanshan University, QinHuangdao, Hebei, China*

Abstract

In recent years, vehicular edge computing (VEC) has emerged as a promising paradigm. In previous studies on offloading in VEC, a task is always supposed to only request one type of service, limiting the universality of research. To address these limitations, we propose a deep reinforcement learning (DRL)-based joint service caching and task offloading scheme, named DRLSCCO. The advantages of the proposed scheme are as follows: 1) it allows each task to request multiple types of services simultaneously, 2) it enables fine-grained task decomposition based on service types and distributed offloading of subtasks, and 3) it supports both cloud collaboration and edge collaboration. To comprehensively evaluate the performance of the proposed DRLSCCO scheme, we conduct extensive comparative experiments against four benchmark schemes. First, we investigate the convergence behavior of different schemes under dynamic task size and vehicle density. Second, by varying task size and vehicle density, respectively, we reveal the variation pattern of task delay underlying the governing of these factors. Third, with vehicle density and task size held constant, we explore the impact of edge server CPU frequency on energy consumption of mobile network operators (MNOs). Results demonstrate that DRLSCCO outperforms the four benchmark schemes in reducing task delay under various parameter configurations, and achieves the slowest increase of energy consumption as edge server CPU frequency increases.

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

1. Introduction

With the rapid development of autonomous vehicles (AVs) 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 enable various communication modes and foster the deployment of computation-intensive and latency-sensitive vehicular applications [1, 2]. Due to the limited computing and caching resources of vehicles, as well as the high latency and backhaul overhead associated with cloud computing, supporting such applications remains challenging [3].

Fortunately, researchers have integrated edge computing into the IoV, leading to the development of vehicular edge computing (VEC) [4, 5, 6]. This architecture enables vehicles migrate applications to the RSUs equipped with edge server or request popular content from these RSUs, reducing task execution 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 [7, 8]. To overcome this limitation, a cloud-edge collaborative vehicular edge computing (VEC) framework has been applied [9, 10]. 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, they overlook the crucial role of service caching in enabling efficient task offloading within VEC, especially given the limited caching resources of edge servers [11].

Service caching storing specific service programs required by vehicular applications to minimize the delay associated with service requests and application initialization, thereby improving the quality of experience (QoE). In fact, task execution fundamentally depends on service caching: a task can only be processed by vehicles or edge servers that have already cached the corresponding service program. 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 [12]. Therefore, there is

an urgent need for a 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 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 DRLSCCO. 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 task demands. The action space simultaneously determines the service caching and task offloading decisions across vehicles and edge nodes. 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 cumulative average task execution delay. The main contributions of this paper are summarized as follows:

1. We propose a novel cloud-edge collaborative VEC framework that incorporates a service-based task decomposition mechanism. It allows each task to request multiple types of services simultaneously and fine-grained task decomposition based on service types. To minimize the cumulative average task execution latency, 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 requirements of tasks, caching constraints at vehicles and edge nodes.
2. Given the NP-hardness of the formulated problem, we design the DRLSCCO 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 size, vehicle density and edge server CPU frequency, to validate the adaptability and effectiveness of our proposed DRLSCCO scheme. The results show that DRLSCCO outperforms the four benchmark schemes in reducing task delay across diverse scenarios and exhibits minimum growth rate of energy consumption as edge server CPU frequency increases.

The rest of this paper is organized as follows. Section II reviews the related work. Section III introduces the system model and formulates the MINLP problem. Section IV presents the proposed DRLSCCO scheme in detail. Section V provides simulation results and performance analysis. Finally, Section VI concludes the paper and outlines directions for future research.

2. Related work

2.1. Studies on MEC

In recent years, computation offloading has emerged as a research hotspot in mobile edge computing (MEC) [13]. To minimize time cost while ensuring low computational delay and high reliability, Lin et al. [14] proposed a Lyapunov-based resource allocation scheme in unmanned aerial vehicles (UAVs)-assisted heterogeneous edge computing system. Zhang et al. [15] 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. To overcome this, reinforcement learning (RL) techniques have been widely adopted to derive adaptive strategies through online interactions. Zhang et al. [16] developed a DRL-based offloading strategy for latency-sensitive home devices to reduce system costs and enhance user experience. Zeng et al. [17] 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. [18] modeled granularity decisions as a multi-armed bandit problem and applied RL to effectively reduce edge server costs. Tang et al. [19] 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 computation offloading, substantial research has been devoted to edge caching for improving content or service delivery efficiency in MEC systems. Bounaira et al. [20] addressed the privacy security concerns arising from the lack of trust between content providers and edge servers in a blockchain-based trust management system. Beyond privacy security, fairness in cache allocation has also attracted attention. Zhou et al. [21] investigated the fair edge data caching (FEDC) problem arising from limited storage and vendor competition, while Tang et al. [22] 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. [23] designed a collaborative meta-learning framework (CMCES) that leverages selective neighbor sampling to improve adaptability, achieving a 10.12% increase in cache hit rate. Then Zhao et al. [24] 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 delay or improve content delivery efficiency. However, most existing MEC studies focus on relatively static or low-mobility scenarios. In contrast, VEC feature highly dynamic topologies, rapidly changing wireless channels, and stringent 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 begun to investigate VEC-specific solutions. In the following, we review representative works on VEC caching and computation offloading. To address energy concerns from the perspective of mobile network operators (MNOs), Kong et al. [25] adopted the DDPG algorithm for joint computing and caching resource allocation. Similarly, Chen et al. [26] proposed a hybrid optimization framework combining deep Q-network (DQN) for task offloading with a greedy strategy for resource management, The framwork 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. [27] proposed a social-aware decentralized cooperative caching (SADC) algorithm that leverages vehicle social networks to

estimate contact rates to reduce the average content access delay. To enhance caching performance, Wu et al. [28] 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 researchers has extended the VEC architecture into more heterogeneous and hierarchical environments. Yu et al. [29] conducted the first survey on edge computing in space-air-ground-integrated networks (SAGINs). They applied an offline deep imitation learning (DIL) algorithm to enable real-time caching and offloading decisions, and also discussed potential integration directions for edge computing in SAGINs. Similarly, Yu et al.[30] addressed high-traffic density by integrating UAVs into cellular networks for cooperative caching. The authors used a temporal-evolving bipartite graph neural network (TBGN) to predict mobility patterns and optimize UAV trajectories.

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. [31] 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. [32] 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. [33] proposed a CO-MATCH algorithm, which integrates dynamic programming-based service caching (DPSC) with a many-to-one matching game (MOMG) for computation offloading. This algorithm jointly considers service caching preferences and social similarity to minimize the overall task processing cost. Service caching preferences are also considered by Ling et al. [34] 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 [31, 32]. 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 with constrained caching and computing resources of both vehicles and edge nodes. 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 sizes and edge server computing capacities. Our objective is to minimize the average task delay while maintaining a low growth rate of MNO's energy consumption.

3. System Model

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

3.1. System Overview

It is assumed that each edge node obtains complete information about the vehicles within its coverage area, and that the coverage areas of different edge nodes are non-overlapping, collectively covering the entire road network. As illustrated in Fig. 1, the system is modeled as a three-layer VEC architecture, consisting of N RSUs equipped with edge servers, L vehicles and a remote cloud server. The set of RSUs is denoted by $\mathcal{N} = \{n \mid n = 1, 2, \ldots, N\}$ and the set of vehicles is denoted by $\mathcal{L} = \{l \mid l = 1, 2, \ldots, L\}$. Here, $L = \sum_{n=1}^{N} \rho_n(t)$ and $\rho_n(t)$ represents the vehicle density within the range of edge node n at 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 in 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

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
$\mathcal{C}_n$	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 program types
$\mathcal{J}_t$	The set of tasks at time slot t
d_j	The input data size of task j
$d_{j,k}$	The input data size of type-k subtask for task j
$f_{j,k}$	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 max CPU frequency of each vehicle
f^R	The max CPU frequency of each edge node
$\omega^V_{l,j,k}$	The offloading proportion of vehicle-to-edge
$\omega^R_{n,j,k}$	The offloading proportion of edge-to-edge

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 demand set of task j is defined as $\mathcal{K}_j = \{k | f_{j,k} > 0\}$. 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

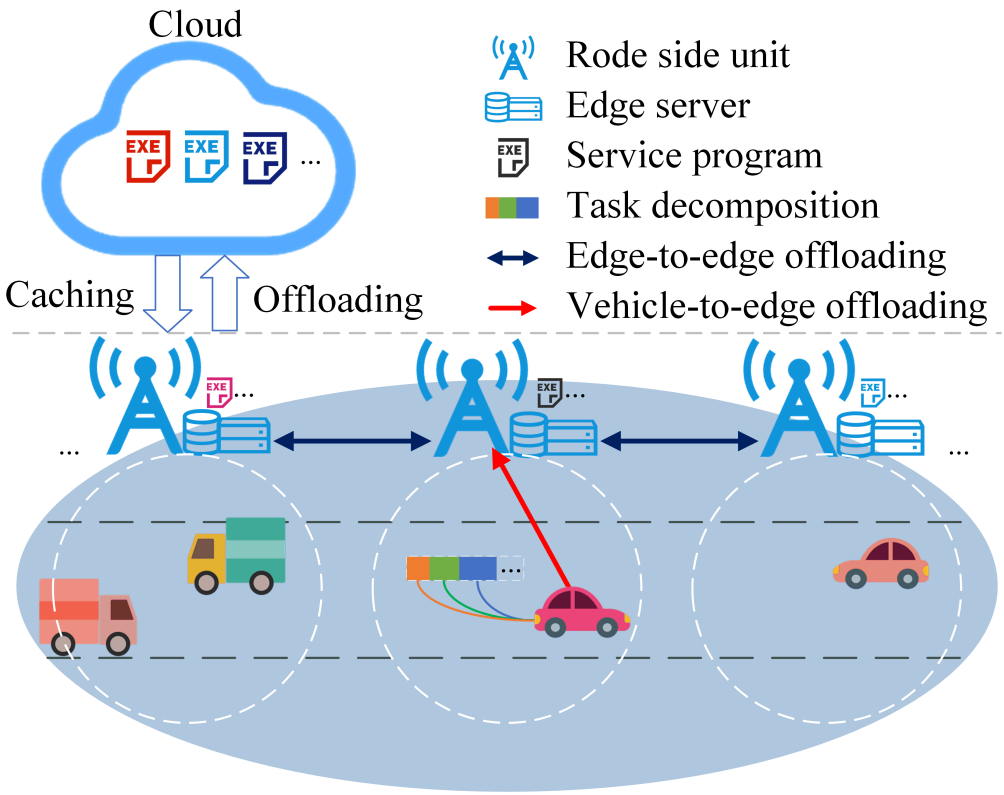

Figure 1: System model.

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. Consequently, if neither the vehicle, the associated edge node, nor its collaborative node can fulfill 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. With 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 proportion (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 the 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 between the vehicle l and the 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 Request Generation Model

For each task, the number of required service types is sampled from a discrete distribution. and this reflects that tasks typically depend on a few types of services. To model the heterogeneous service demands observed in vehicular edge computing systems, the selection of service dependencies for each task is assumed to follow by the Zipf distribution with a skewness parameter $z > 0$ [30]. Specifically, the selection probability P_k of type-k service is given by

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

After selecting the required service types, the demand proportions for each selected service are assigned by a Dirichlet distribution ensuring that $f_{j,k}(t) > 0$ and $\sum_{k=1}^{K} f_{j,k}(t) = 1$. This modeling approach not only reflects the sparse and biased nature of service dependencies but also simulate diverse task generation patterns in vehicular edge computing networks.

3.4. Service Caching and Task Offloading Model

To fully utilize the limited caching resources of vehicles and edge nodes, we assume that each task can be processed within its originating time slot t; An agent that deployed on the cloud server determines the caching strategy for each type of service at both the vehicle and edge node before the end of

Table 2: Service deployment mode and subtask offloading proportion

Service deployment mode		Proportion offloaded to entity			
		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

each time slot. The caching capacity constraints for the vehicle l and edge node n are given by

$$\beta \boldsymbol{\delta}^V_l(t) \cdot \mathbf{1}^{\mathrm{T}} \leq S^V \tag{4}$$

and

$$\beta \boldsymbol{\delta}^R_n(t) \cdot \mathbf{1}^{\mathrm{T}} \leq S^R, \tag{5}$$

respectively. Here, the constant β represents the size of each service file. The vectors $boldsymbol\delta^V_l(t)$ and $\boldsymbol{\delta}^R_n(t)$ denote the caching states of vehicle l and edge node n for all services, respectively. Each component of the vector is a binary variable indicating the caching status of a specific service at an entity: a value of 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 cache states of all entities. Let the continuous variables $\omega^V_{l,j,k}(t)$ and $\omega^R_{n,j,k}(t)$ denote the offloading proportion of subtask $s_{j,k}(t)$ of vehicle-to-edge and edge-to-edge respectively. The offloading strategies corresponding to different caching deployment modes are summarized in Table 2.

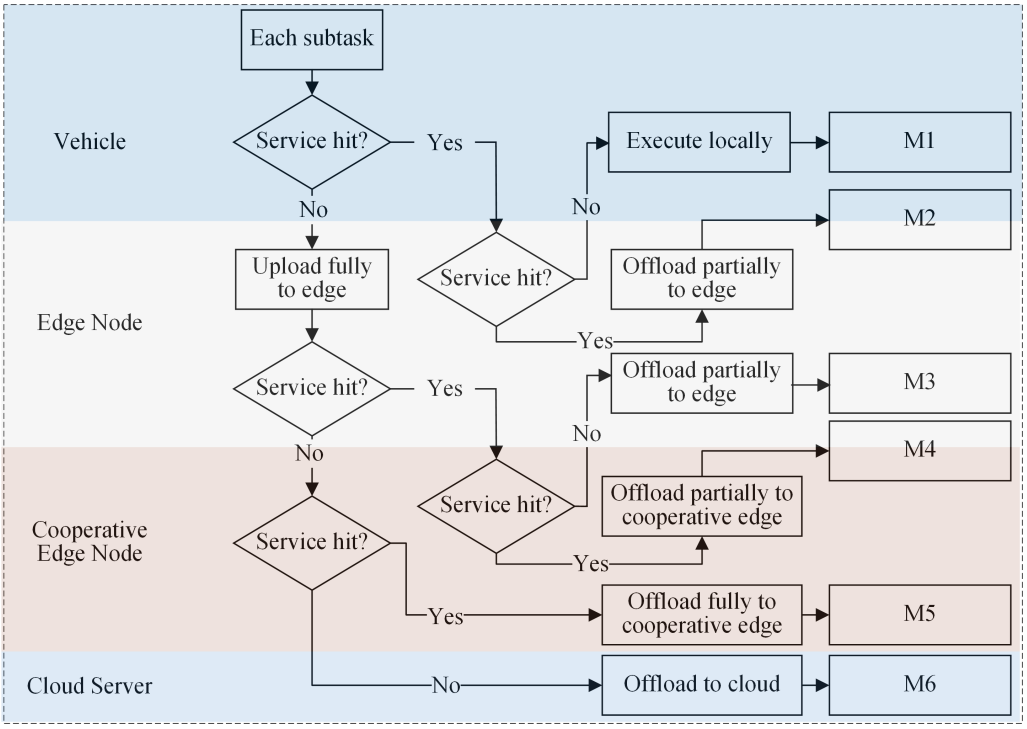

Figure 2: Flowchart of offloading.

Specifically, the agent performs fine-grained subtask offloading for each task. For the type-k service required by subtask $s_{j,k}(t)$, it first checks whether the service is cached locally. If so, the agent further verifies whether the associated edge node n also caches the service, to determine whether the subtask should be fully executed locally or partially offloaded to a edge node. If the service is not cached at the vehicle, the subtask is fully uploaded to the associated edge node n. Based on the 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 node, or partially to both. It is worth noting that when multiple cooperative edge nodes meet the subtask offloading requirements, one of them is randomly selected for collaboration. 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. The detailed offloading procedure is illustrated in Fig. 2.

3.5. Computation Delay and Energy Consumption

Based on Sec. 3.2 and Sec. 3.4, the delay of subtask $s_{j,k}(t)$ is calculated as follows.

The computation delay locally is given by

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

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. If the subtask is offloaded to edge node n, the transmission delay for uploading the input data is given by

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

The computation delay at edge node n is expressed as

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

where f^R denotes the CPU frequency of the edge server. If the subtask is further offloaded from edge node n to a cooperative edge node m, the transmission delay between edge nodes can be expressed as

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

where r^{e2e} represents the data transmission rate between edge nodes. The computation delay at the cooperative edge node m is expressed as

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

If the vehicle l, edge node n, and cooperative edge node m do not meet the offloading conditions, the subtask will be uploaded from the edge node n to the cloud server, with the transmission delay given by

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

where r^{e2c} denotes the data transmission rate from edge node to remote cloud server. In the delay calculations above, since the task output is much

smaller than the input and the cloud server has substantial computational resources, the delay caused by result transmission and cloud computation is negligible [35]. In summary, the execution delay of the subtask $s_{j,k}(t)$ can be expressed as:

$$\begin{aligned} D_{j,k}^{total}(t) = &\max\Bigg\{ D_{l,j,k}^{local}(t), D_{l,j,k}^{up}(t) + \max\Big\{ D_{n,j,k}^{edge}(t), \\ &D_{n,j,k}^{e2e}(t) + D_{m,j,k}^{co}(t), D_{n,j,k}^{cloud}(t) \Big\} \Bigg\}, \end{aligned} \tag{12}$$

which encompasses all the service deployment modes presented in Table 2. Then the execution delay of task j is defined as

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

The average delay for executing all tasks in the system at time slot t is defined as follows

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

According to [25], the MNOs' energy consumption is considered in our study, which includes the energy consumption by edge servers and cloud servers. The total energy consumption for executing subtask $s_{j,k}(t)$ is given as follows.

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

Here, κ represents the processor energy efficiency parameter of edge server, which typically depends on hardware conditions. Parameters p^{e2e} and p^{e2c} denote the edge-to-edge transmit power and edge-to-cloud transmit power, respectively. Then the execution energy consumption of task j is defined as

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

The average energy consumption is similar with (14), defined as

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

3.6. Computation Delay and Energy Consumption Extremum Analysis

In the Sec. 3.5, the delay and energy consumption was conducted from the perspective of subtask offloading flow. In this section, we further evaluate the delay and energy consumption under different service deployment modes and derive extremum values, which helps to understand the best-case and worst-case performance of subtask offloading and reveal the performance boundaries caused by different caching conditions.

The execution delay and MNOs' energy consumption of for subtask $s_{j,k}(t)$ in different modes is as follows:

1. M1: The subtask is computed locally

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

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

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

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

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

3. M3: The subtask is fully offloaded to edge node

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

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

4. M4: The subtask is offloaded to both edge node and cooperative edge node

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

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

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

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

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

6. M6: The task is fully offloaded to cloud server

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

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

By comparing the execution delay under different service deployment modes, the maximum value can be derived as

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

and the minimum value can be derived as

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

According to convex piecewise optimization, the offloading proportion of (20) and (24) can be eliminated, i.e., when $\omega_{l,j,k}^V(t)$ takes the following value

$$\omega_{l,j,k}^V(t) = \frac{1}{(1 + \frac{f^V}{\lambda r_{l,n}(t)} + \frac{f^V}{f^R})}, \tag{32}$$

the minimum value of $D^{M2}(t)$ can be expressed as

$$\min\{D^{M2}(t)\} = \frac{\lambda d_{j,k}(t)\left(f^R + \lambda r_{l,n}(t)\right)}{\lambda f^R f^V + f^R r_{l,n}(t) + \lambda r_{l,n}(t) f^V}, \tag{33}$$

and the minimum value of $D^{M4}(t)$ can be expressed as

$$\min\{D^{M4}(t)\} = \frac{d_{j,k}(t)}{r_{l,n}(t)} + \frac{\lambda d_{j,k}(t)\left(f^R + \lambda r^{e2e}\right)}{f^R\left(f^R + 2\lambda r^{e2e}\right)}, \tag{34}$$

when $\omega_{n,j,k}^R(t)$ takes the following value

$$\omega_{n,j,k}^R(t) = \frac{1}{(2 + \frac{f^R}{\lambda r^{e2e}})}. \tag{35}$$

Hence, the minimum value of the execution delay can be also expressed as

$$\begin{aligned}\min\{D_{j,k}^{total}(t)\} = \min\Big\{ & \min\{D^{M2}(t)\}, \min\{D^{M4}(t)\}, \\ & D^{M6}(t)\Big\}.\end{aligned} \tag{36}$$

From the energy consumption analysis under different modes, the minimum energy consumption incurred by MNOs for subtask execution is 0, while the maximum energy consumption is represented as

$$\max\{E_{j,k}^{total}(t)\} = \max\left\{E^{M5}(t), E^{M6}(t)\right\}. \tag{37}$$

According to (30) and (36), the subtask execution delay is closely related to factors such as subtask data size and device computing capabilities. The maximum delay may occur in M1, M5 or M6, while the minimum delay is likely to occur in M2, M4, or M6. The reason why M6 appears in both the maximum and minimum delay cases is that, when a subtask is offloaded to the cloud server, the delay is highly sensitive to the data size. Once the data size exceeds a certain threshold, the maximum delay tends to occur in M6. This suggests that to minimize execution delay, the agent should involve edge nodes in subtask computation as much as possible, i.e., prioritize caching services at the edge nodes. Meanwhile, (37) shows that the maximum energy consumption arises in M5 or M6. Although the participation of edge nodes increases energy consumption, if the agent can realize more service deployment modes M2 and M4, it will effectively reduce the delay while keeping MNOs' energy consumption within a reasonable range.

Deriving the extremum values of execution delay and energy consumption offers guidance for the design of subsequent scheme design such the parameter setting in simulation experiments and informs the optimization direction by clarifying the effects of service availability on offloading decisions.

3.7. Problem Formulation

In this section, to minimize the long-term cumulative average latency, the problem of service caching and task offloading is formulated as a MINLP optimization problem. Let $\mathbf{Z}(t)$ denote decision variables at time slot t, i.e., $\mathbf{Z}(t) = [\omega_{l,j,k}^{V}(t), \omega_{n,j,k}^{R}(t), \delta_{l,k}^{V}(t), \delta_{n,k}^{R}(t)]$. The detailed problem formulation

is presented as follows.

$$\min_{\mathbf{Z}(t)} \quad \sum_{t=0}^{T-1} D^{\text{ave}}(t), \tag{38}$$

s.t.

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

$$\omega^V_{l,j,k}(t) \in [0,1], \tag{38a}$$

$$\omega^R_{n,j,k}(t) \in [0,1], \tag{38b}$$

$$\omega^V_{l,j,k}(t) + \omega^R_{n,j,k}(t) = 1 \tag{38c}$$

$$\delta^V_{l,k}(t) \in \{0,1\}, \tag{38d}$$

$$\delta^R_{n,k}(t) \in \{0,1\}, \tag{38e}$$

$$\beta \boldsymbol{\delta}^V_l(t) \cdot \mathbf{1}^{\mathrm{T}} \leq S^V, \tag{38f}$$

$$\beta \boldsymbol{\delta}^R_n(t) \cdot \mathbf{1}^{\mathrm{T}} \leq S^R. \tag{38g}$$

In (38), constraints (38a), (38b) and (38c) ensure that the offloading ratios of a subtask are continuous values within the range $[0, 1]$ and sum to one, thereby guaranteeing the completeness of task offloading; constraints (38d) and (38e) specify that the caching states of a particular service at a vehicle or edge node is represented by binary variables; constraints (38f) and (38g) ensure that the cached services do not exceed the storage capacity of vehicles or edge nodes.

4. The Proposed DRLSCCO Scheme

Traditional methods, such as convex optimization and game theory, are commonly used in edge computing but struggle in the dynamic VEC environment due to their static nature. Heuristic and rule-based algorithms, though computationally feasible, fail to generalize across varying VEC scenarios. The DDPG algorithm, a DRL algorithm combining deep neural networks (DNN) and reinforcement learning (RL), enables agents to interact with the environment and make decisions based on experiences, especially suited for high-dimensional systems like VEC, where continuous decision variables such as offloading proportion are involved [36]. In this section, we transform the problem in (38) as a Markov Decision Process (MDP) and propose a DDPG-based joint service caching and computation offloading scheme, DRLSCCO, to optimize caching and offloading decisions, with a detailed explanation of the DRLSCCO scheme.

4.1. Problem Formulation Based on RL

In the proposed DRLSCCO scheme, the DDPG algorithm operates within RL framework, which is typically formulated as MDP. The MDP leverages the Markov property, where future state transitions depend solely on the current state, independent of past states. We first transform the problem as an MDP, providing a formal structure for sequential decision-making in stochastic VEC environments, allowing the agent to learn policies through interaction with the system over time. A typic 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. These three elements are defined as follows:

4.1.1. State space

According to the proposed system model, the system state includes the following components:

- $\boldsymbol{\delta}^{\mathrm{V}}(t)$: The caching state of vehicles for services at time slot t.
- $\boldsymbol{\delta}^{\mathrm{R}}(t)$: The caching state of edge nodes for services at time slot t.
- $\mathbf{d}(t)$: The task size generated by vehicles at time slot t.
- $\mathbf{b}(t)$: The bandwidth allocated to vehicles at time slot t.
- $\boldsymbol{\gamma}(t)$: The received SINR of edge nodes when communicating with vehicles at time slot t.

At the beginning of the time slot t, the state space of system can be expressed as

$$\begin{aligned}\mathbf{S}(t) = \big\{ &[\boldsymbol{\delta}^{\mathrm{V}}(t)]^{L\times K}, [\boldsymbol{\delta}^{\mathrm{R}}(t)]^{N\times K}, [\mathbf{d}(t)]^{L}, \\ &[\mathbf{b}(t)]^{L}, [\boldsymbol{\gamma}(t)]^{L} \big\}.\end{aligned} \tag{39}$$

4.1.2. Action space

After obtaining the system state, the agent determines the subtask offloading locations and offloading proportion before the end of time slot, while also updating the service deployment of vehicles and edge nodes. The action space at time slot t includes the following components:

- $\boldsymbol{\delta}^{\mathrm{V}}(t)$: The caching decisions of vehicles for services at time slot t.
- $\boldsymbol{\delta}^{\mathrm{R}}(t)$: The caching decisions of edge nodes for all services at time slot t.

- $\boldsymbol{\omega}^{\mathrm{V}}(t)$: The proportion of tasks offloaded to connected edge node at time slot t.
- $\boldsymbol{\omega}^{\mathrm{R}}(t)$: The proportion of tasks transmitted from an edge node to a cooperative edge node at time slot t.

The action space at time slot t can be represented as

$$\begin{aligned}\mathbf{A}(t) = \big\{&[\boldsymbol{\delta}^{\mathrm{V}}(t)]^{\times L\times K}, [\boldsymbol{\delta}^{\mathrm{R}}(t)]^{N\times K} \\ &[\boldsymbol{\omega}^{\mathrm{V}}(t)]^{L}, [\boldsymbol{\omega}^{\mathrm{R}}(t)]^{L}\big\}.\end{aligned} \tag{40}$$

4.1.3. Reward

The reward in RL provides immediate feedback on the agent's actions, guiding its decision-making by evaluating the quality of decisions in a given state and influencing future behavior. Given a state $\mathbf{S}(t)$ and action $\mathbf{A}(t)$, the reward at time slot t is defined as

$$r_t = r(\mathbf{S}(t), \mathbf{A}(t)) = -D^{ave}(t). \tag{41}$$

4.2. DDPG-Based Service Caching and Task offloading Scheme

At each time slot, the environment is characterized by the state space $\mathbf{S}(t)$ and the action space $\mathbf{A}(t)$, which describe all possible states and actions. The agent's policy is continuously updated through iterative interactions with the environment, gradually converging to an optimal strategy. The agent architecture in DRLSCCO—comprising policy and target networks as well as a replay buffer—is designed to support this learning process. The framework of the DRLSCCO scheme is illustrated in Fig. 3.

In the proposed DRLSCCO scheme, a specific state-action pair (s_t, a_t) is sampled from $\mathbf{S}(t) \times \mathbf{A}(t)$, where $s_t \in \mathbf{S}(t)$, $a_t \in \mathbf{A}(t)$. As shown in Fig. 3, the policy actor network implement 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 policy critic network estimate the Q-value (i.e., action-value function) $Q(s_t, a_t|\theta^{Q})$, obtained by taking action a_t in state s_t, which represents the expected cumulative discounted reward and can be expressed as

$$Q(s_t, \mu(s_t|\theta^{\mu})|\theta^{Q}) = \mathbb{E}_{s_{t+1}\sim \mathrm{P}}\left[\sum_{k=0}^{\infty} \alpha^k r_{t+k}\right]. \tag{42}$$

Equation (42) satisfies a recursive relationship called Bellman equation, which can be written as

$$\begin{aligned}Q(s_t, \mu(s_t|\theta^{\mu})|\theta^{Q}) =& \mathbb{E}_{s_{t+1}\sim \mathrm{P}}\big[\alpha Q(s_{t+1}, \mu(s_{t+1}|\theta^{\mu})|\theta^{Q}) \\ &+ r_t\big].\end{aligned} \tag{43}$$

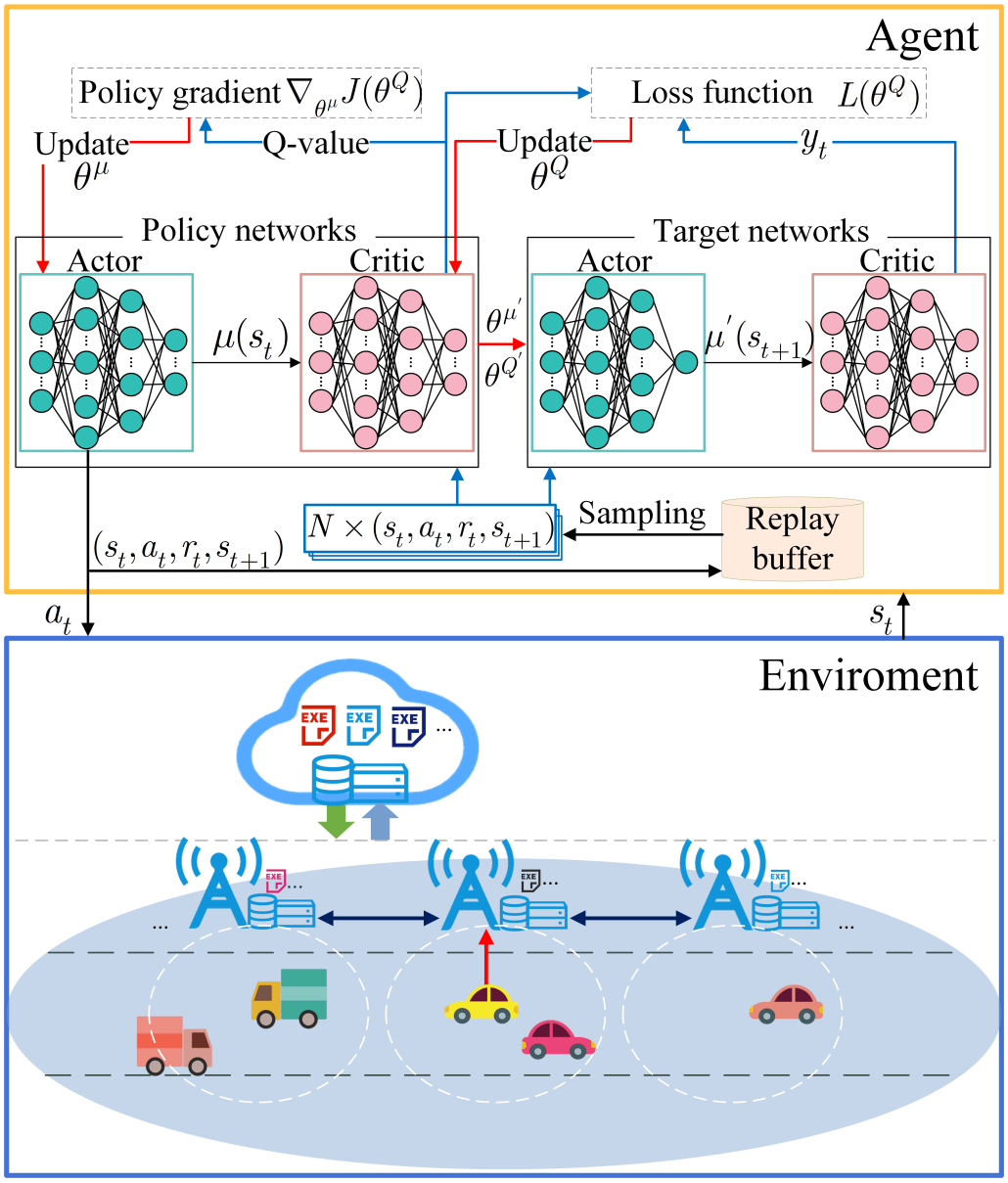

Figure 3: The framework of the DRLSCCO.

Here, α is the discount factor; θ^μ and θ^Q denote the parameters of the actor and critic networks in the policy networks, respectively. 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 ensure stable training, the target Q-value y_t is constructed by replacing the Q-value and policy with their target networks' counterparts. It is defined as

$$y_t = r_t + \alpha Q'(s_{t+1}, \mu(s_{t+1}|\theta^{\mu'})|\theta^{Q'}). \tag{44}$$

The target networks are maintained with parameters $\theta^{\mu'}$ and $\theta^{Q'}$, which 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, yielding the modified action a_t.

$$a_t = \mu(s_t|\theta^{\mu}) + n_t. \tag{45}$$

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}). At each time slot, after the agent selects an action a_t and receives the corresponding reward r_t and next state r_{t+1}, the transition (s_t, a_t, r_t, s_{t+1}) is stored in the experience replay buffer. The data randomly sampled from the replay buffer is used for updating the parameters of the policy networks.

Now that experience replay buffer provides samples of mini-batch of transitions, the critic network θ^Q is updated by minimizing the mean squared error (MSE) between the Q-value and the target Q-value. The loss function is defined as

$$L(\theta^Q) = \frac{1}{N}\sum_{i=1}^{N}\left(y_i - Q\left(s_i, \mu\left(s_i|\theta^{\mu}\right)|\theta^Q\right)\right)^2, \tag{46}$$

where N is the mini-batch size sampled from the experience replay buffer. The actor network parameter θ^{μ} is updated using the sampled deterministic policy gradient, which aims to maximize the Q-value and can be expressed as

$$\begin{aligned}\nabla_{\theta^{\mu}} J(\theta^Q) =& \frac{1}{N}\sum_{i=1}^{N}\left[\nabla_a Q\left(s, a|\theta^Q\right)|_{s=s_i, a=\mu(s_i|\theta^{\mu})}\right. \\ & \left.\cdot \nabla_{\theta^{\mu}}\mu(s|\theta^{\mu})|_{s=s_i}\right].\end{aligned} \tag{47}$$

Therefore, the critic and actor parameters are updated by gradient descent and gradient ascent, which as follows:

$$\theta^Q \leftarrow \theta^Q - \eta \cdot \nabla_{\theta^Q}(L(\theta^Q)) \tag{48}$$

and

$$\theta^{\mu} \leftarrow \theta^{\mu} + \eta \cdot \nabla_{\theta^{\mu}}(J(\theta^{\mu})), \tag{49}$$

where η denote learning rate. The target networks are not directly copied from the policy networks but updated using a soft update mechanism, which slowly tracks the policy networks' parameters. At each iteration, the target networks' parameters $\theta^{\mu'}$ and $\theta^{Q'}$ are updated according to

$$\theta^{Q'} \leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q'} \tag{50}$$

and

$$\theta^{\mu'} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'}. \tag{51}$$

Here, $0 < \tau \ll 1$ is a smoothing factor, which prevent excessive fluctuations in the target Q-value during learning. Through the above process, the agent gradually optimizes decision-making strategy, enabling it to efficiently perform service caching and offloading decisions in complex VEC environment. The complete DRLSCCO scheme is detailed in Algorithm 1.

5. Simulation Experiments and Performance Analysis

In this section, we analyze the performance of the proposed DRLSCCO scheme by conducting simulation experiments under different parameter settings to emulate various scenarios. We use Python 3.8 and PyTorch 2.1.2 to simulate the proposed system model in our study and implement the DRLSCCO scheme. The simulation environment considers a three-layer VEC system, consisting of a cloud server, edge nodes and vehicles. The primary parameters used in the experiments are summarized in Table 3, with some parameter settings referenced from previous studies [37, 38].

To comprehensively evaluate the proposed DRLSCCO scheme, we compare it with four service caching and computation offloading benchmark methods:

1. No comparative edge nodes (NCE): Subtasks are restricted to three execution options: local execution, (partial) offloading to directly connected edge nodes, or offloading to the cloud server. The cooperative interactions between edge nodes are disabled.
2. LRU-based service caching on vehicles and edge nodes (LRUSC): If the agent attempts to cache a new service, but the cache capacity is insufficient, it replaces the least recently used service.

Algorithm 1 The DRLSCCO Scheme for Service Caching and Computation Offloading.

Input: $\mathcal{N}$, $\mathcal{L}$, $\mathcal{T}$, $\mathcal{K}$, $\mathcal{J}_t$, $\mathcal{K}_j$, $\mathcal{C}_n$.

Output: The optimal policy actor network parameter θ^{μ}.

Initialize networks : the policy actor network parameter θ^{μ}; the policy critic network parameter θ^{Q}; the target actor network parameter $\theta^{\mu'}$; the target critic network parameter $\theta^{Q'}$; the soft update coefficient τ; the number of episodes I; the batch sample size e; the replay buffer size E; the random noise n_t.
Initialize empty experience replay buffer.
for $i = 1, 2, \ldots, I$ **do**
  Initialize the observation state s_0 of time slot 0 with the input.
  for $t = 0, 1, \ldots, T-1$ **do**
    The agent selects an action a_t according to (45).
    Vehicles and edge servers execute the selected 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 replay buffer size exceeds a predefined threshold **then**
      Randomly sample a mini-batch of e transitions $(s, a, r, s_{\text{next}})$ from the replay buffer.
      Calculate the Q-value by (42).
      Calculate the target Q-value y by (44).
      Update the critic network parameter θ^{Q} by minimizing the loss function defined in (46).
      Update the actor network parameter θ^{μ} (47).
      Perform a soft update on the target networks' parameters $\theta^{Q'}$ and $\theta^{\mu'}$ according to (50) and (51), respectively.
    end if
  end for
end for

Table 3: Parameter Setting

Parameter	**Meaning**	**Value**
lr_a	The learning rate of actor network	0.001
lr_c	The learning rate of critic network	0.002
γ	The discount factor	0.99
e	The sample batch size	256
ρ^{max}	The max density of vehicles within range of each edge node	10
N	The number of edge nodes	3
K	The number of service types	3
β	The data size of each service	30 Mb
$d_j(t)$	The data size of each task	20 Mb
S^V	The storage capacity of each vehicle	30 Mb
S^R	The storage capacity of each edge server	50 Mb
r^{e2e}	The transmission rate between edge nodes	10 Mbps
r^{e2c}	The transmission rate from edge node to cloud server	8 Mbps
p^{edge}	The transmission power between edge nodes	1 W
p^{e2c}	The transmission rate from edge node to cloud server	2 W

3. Random service caching and computation offloading (RSCCO): The caching behavior of vehicles and edge nodes is randomized. Additionally, if a subtask is partially offloaded, its offloading ratio is also determined randomly.
4. No edge nodes (NE): In this case, edge nodes are removed from the system, and subtasks can only be executed locally or offloaded to the cloud server.

First, we compare the average task processing delay per training episode for schemes involving network participation in Fig. 4. Since MNOs' energy consumption is closely related to task processing delay, we further compare the average energy consumption per episode as training progresses in Fig. 5. Among these, the average delay of schemes decreases and gradually

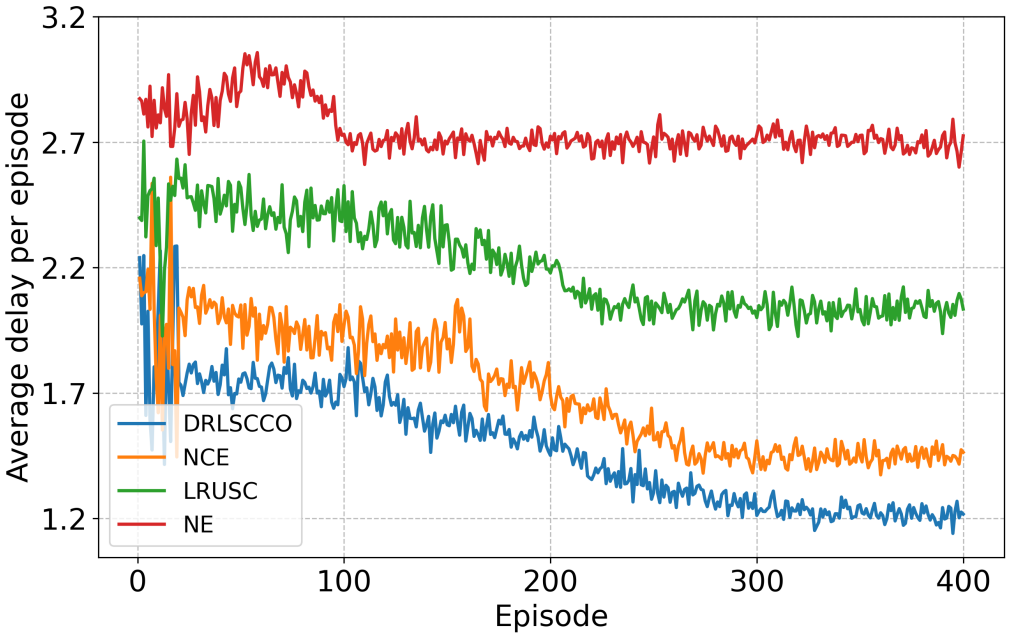

Figure 4: The performance of average delay per episode

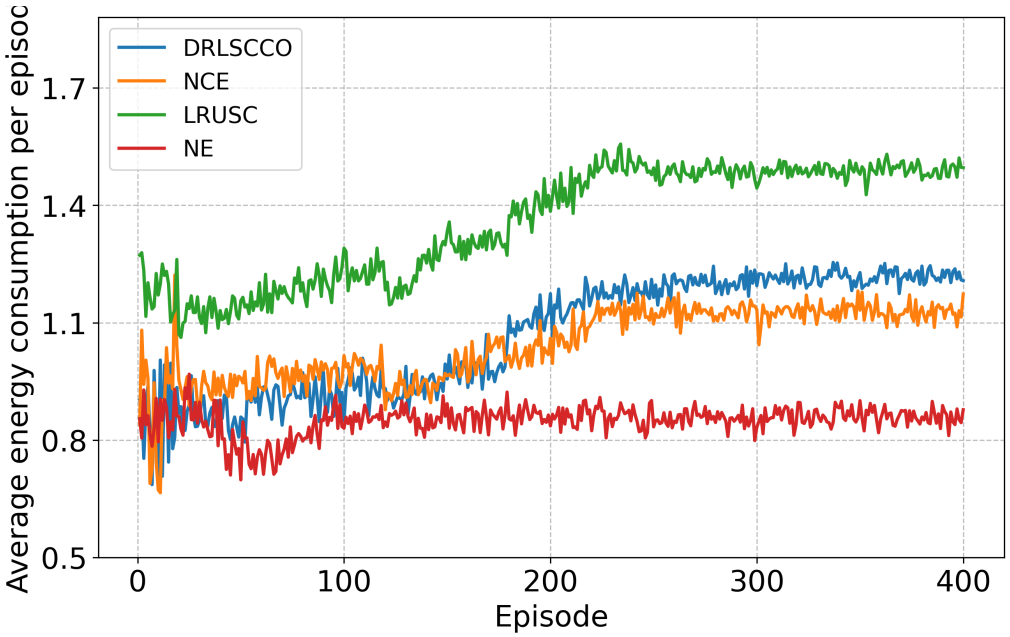

Figure 5: The performance of average energy consumption per episode

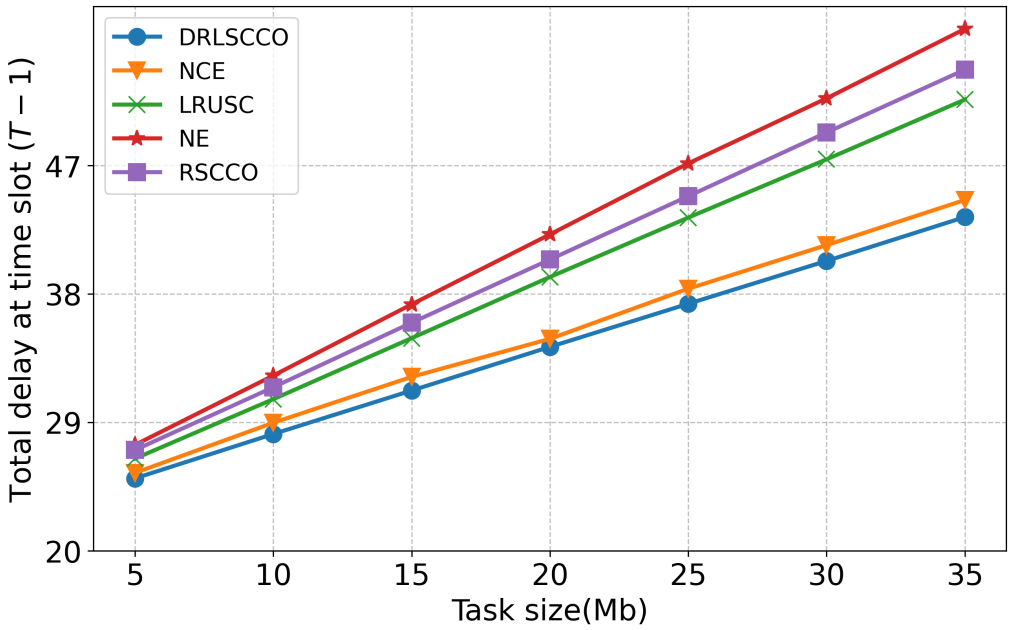

Figure 6: The influence of task size on delay

stabilizes as the number of episodes increases, while their average energy consumption increases and reach a steady state. This is because the participation of edge servers and cloud servers deployed by the MNOs reduces task execution delay, while simultaneously increasing the MNOs' energy consumption. Lacking cooperative caching and offloading, the NCE forces subtasks to be offloaded to the cloud when edge nodes are unavailable, increasing average transmission delay. The LRUSC, relying solely on past access records for cache replacement, operates more greedily than DRLSCCO and NCE, thus impacting offloading decisions and resulting in higher delay. In the NE, the absence of edge servers means that task execution delay is primarily determined by computation on vehicle and transmission latency. As a result, it exhibits the highest delay, highlighting the advantages of edge computing. Overall, these results demonstrate that the proposed scheme, DRLSCCO, continuously optimizes both service caching allocation and subtask offloading scheduling, achieving the lowest delay while keeping the increase in energy consumption within a relatively small range.

Then the vehicle density is fixed at 10 to explore the impact of varying task size on the total delay under all schemes, as shown in Fig. 6. As the task size increases from 5 Mb to 35 Mb, the total delay increases

steadily across all schemes, as expected, due to the positive correlation between delay and subtask size $d_{j,k}(t)$ in all delay-related formulations. When the task size is small, the delays of all schemes are relatively close. As the task size increases, the delay differences among the schemes become more pronounced. The NE scheme, which lacks edge servers, exhibits the most significant increase in delay, followed by RSCCO and LRUSC. Compared to DRLSCCO, the NCE scheme offloads more subtasks to the cloud server—corresponding to the service deployment mode M6—thereby resulting in higher delays. Among all the schemes, the proposed scheme DRLSCCO consistently achieves the lowest total delay under all task sizes. This demonstrates the effectiveness of the scheme in adapting to varying task loads and optimizing offloading decisions dynamically.

Subsequently, the task size is fixed at 20 Mb to investigate the impact of vehicle density on task execution delay in Fig. 7. The results demonstrate that as vehicle density increases, the average delay of all schemes rises due to increased demand on the communication channel, which reduces per-vehicle bandwidth and leads to higher transmission delays. When the vehicle density is relatively low, i.e., $\rho^{\max} = 4$, 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 overall processing delay. Consequently, in low-density scenarios, the NE scheme exhibits a lower average delay compared to LRUSC and RSCCO, both of which involve edge participation, but this advantage vanishes as vehicle density increases. Notably, the proposed DRLSCCO scheme consistently achieves the lowest task execution delay across varying vehicle densities, highlighting its strong adaptability and effectiveness in dynamic environments.

The results in Fig. 8 and Fig. 9 align with theoretical expectations, demonstrating that higher CPU frequency enable lower task execution delay but also lead to increased energy consumption. Since the NE scheme does not utilize edge servers, its delay and energy consumption remain constant regardless of CPU frequency changes. The LRUSC scheme, which only implements the LRU strategy for service caching replacement without jointly optimizing service caching and task offloading, is less sensitive to variations in CPU frequency. Meanwhile, LRUSC generally exhibits lower delay and energy consumption compared to RSCCO, which lacks clear caching and offloading strategies and thus shows the most unstable performance with the highest delay. Notably, when the CPU frequency is relatively low, the

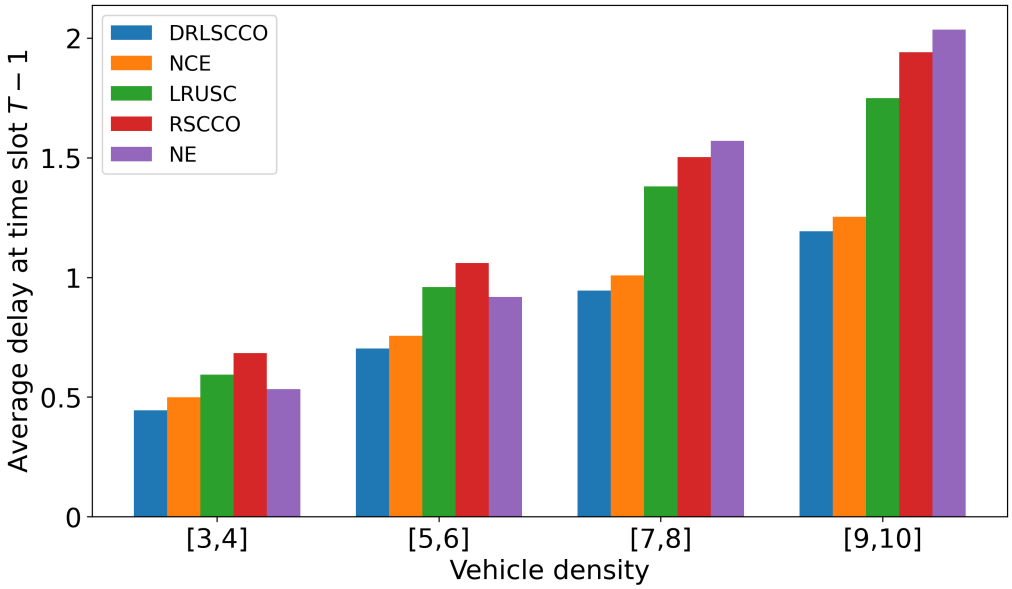

Figure 7: The influence of vehicle density on delay

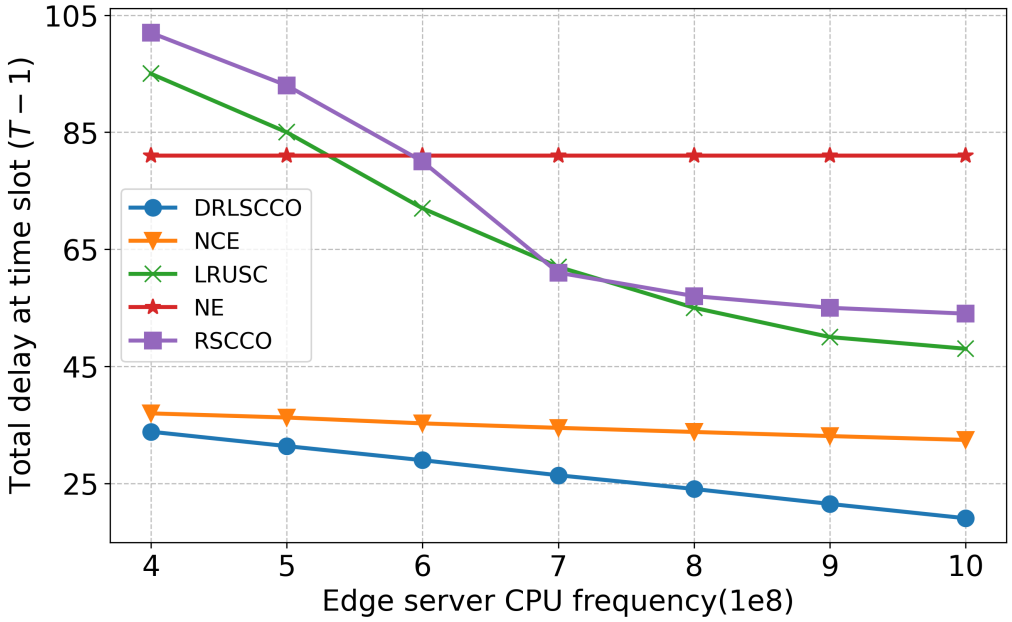

Figure 8: The influence of edge server's CPU frequency on delay

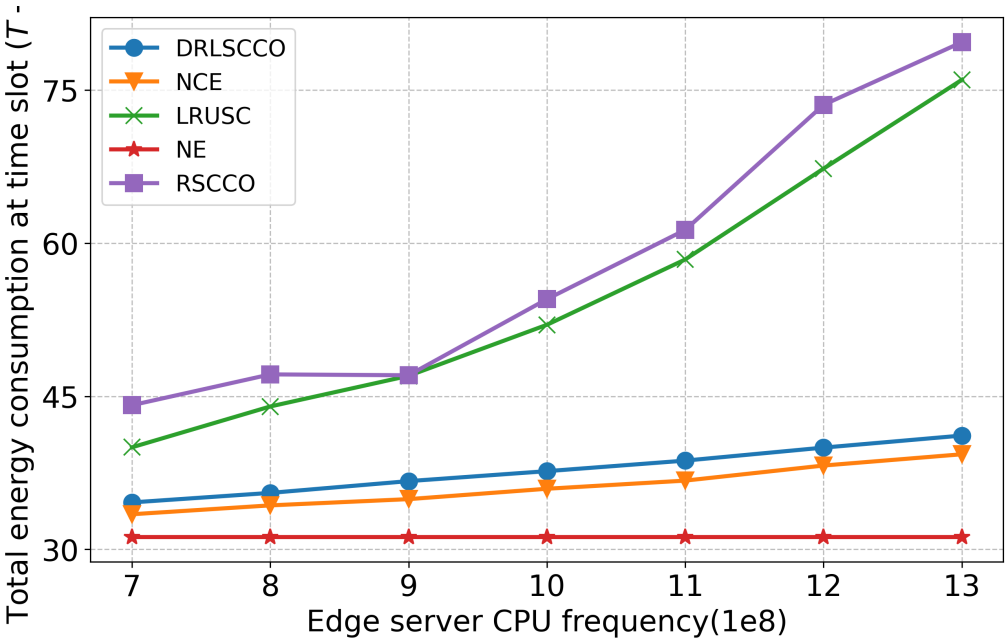

Figure 9: The influence of edge server's CPU frequency on energy

delays of LRUSC and RSCCO even exceed that of NE. This indicates that, under the same task size and vehicle density, the delay incurred by offloading subtask to cloud server may be lower than executing them on edge server with low computational capacity. However, as the CPU frequency increases to 6×10^8, the delays of LRUSC and RSCCO become lower than that of NE. Although the MNOs' energy consumption of LRUSC and RSCCO remains consistently higher than NE, this highlights that appropriate deployment of edge computing capability by MNOs can significantly enhance user experience compared to relying solely on cloud computing. In Fig. 8, it can be observed that for the proposed DRLSCCO scheme and the NCE scheme, the delays are relatively close when the CPU frequency is 4×10^8. As the CPU frequency increases, although the delay gap between DRLSCCO and NCE gradually widens, the growth rates of energy consumption for the two schemes remain nearly the same. This phenomenon arises because when the edge computing capability is low, the subtask offloading among edge nodes in DRLSCCO causes additional transmission delays. Once the computational capacity becomes sufficient, this disadvantage diminishes. Moreover, since the MNO's energy consumption is closely related to computational delay, the energy consumption growth rate of DRLSCCO does not signifi-

cantly exceed that of NCE.

6. Conclusion

This paper investigates a VEC system with limited edge caching resources, where each task depends on multiple types of services and tasks are decomposed into subtasks based on service requirements, resulting in a tightly coupled process between service caching and fine-grained task offloading. We design the communication, caching, and offloading mechanisms, and model the execution delay and MNOs' energy consumption. The joint service caching and subtask offloading problem is formulated as a MINLP problem, which is addressed using a DRL-based scheme, referred to as DRLSCCO. Extensive simulation results demonstrate that DRLSCCO significantly reduces task execution delay while effectively balancing MNO energy consumption compared to several baseline methods. For future work, potential research directions include addressing heterogeneous communication constraints, such as intermittent connectivity, and exploring decentralized or multi-agent learning frameworks to enhance system scalability and robustness.

References

[1] L. Xing, Reliability in internet of things: Current status and future perspectives, IEEE INTERNET OF THINGS JOURNAL 7 (8) (2020) 6704–6721, 2020-08-01 Review. doi:10.1109/JIOT.2020.2993216.

[2] 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-AN INTERNATIONAL JOURNAL-JESTECH 54, 2024-06-15 Review (2024). doi:10.1016/j.jestch.2024.101699.

[3] A. Waheed, M. A. Shah, S. M. Mohsin, A. Khan, C. Maple, S. Aslam, S. Shamshirband, A comprehensive review of computing paradigms, enabling computation offloading and task execution in vehicular networks, IEEE ACCESS 10 (2022) 3580–3600, 2022-01-19 Review. doi:10.1109/ACCESS.2021.3138219.

[4] L. Liu, C. Chen, Q. Q. Pei, S. Maharjan, Y. Zhang, Vehicular edge computing and networking: A survey, MOBILE NETWORKS & APPLICATIONS 26 (3) (2021) 1145–1168, times Cited in Web of Science Core Collection: 292 Total Times Cited: 312 Cited Reference Count: 142 ER -. doi:10.1007/s11036-020-01624-1.

[5] H. H. Wu, B. B. Wang, H. H. Ma, X. H. 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, times Cited in Web of Science Core Collection: 1 Total Times Cited: 1 Cited Reference Count: 37 ER -. doi:10.1109/JIOT.2024.3392329.

[6] L. M., H. S., G. Z., T. Y., H. L., D. H., Allocation of computation-intensive graph jobs over vehicular clouds in iov, IEEE Internet of Things Journal 7 (1) (2020) 311–324, iEEE Internet of Things Journal. doi:10.1109/JIOT.2019.2949602.

[7] 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, 2023-09-02 Review (2023). doi:10.1016/j.iot.2023.100690.

[8] Y. Khan, S. Mustafa, R. W. Ahmad, T. Maqsood, F. Rehman, J. Ali, J. J. P. C. Rodrigues, Content caching in mobile edge computing: a survey, CLUSTER COMPUTING-THE JOURNAL OF NETWORKS SOFTWARE TOOLS AND APPLICATIONS 27 (7) (2024) 8817–8864, 2024-05-30 Review. doi:10.1007/s10586-024-04459-7.

[9] 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, 2024-08-23 Article. doi:10.1109/JIOT.2023.3288200.

[10] X. Ren, X. Chen, L. Jiao, X. Dai, Z. Dong, IEEE, Joint optimization of trajectory, caching and task offloading for multi-tier uav mec networks, 2024-09-21 ER - Proceedings Paper (2024-01-01 2024).

[11] Y. Y. Mao, C. S. You, J. Zhang, K. B. Huang, K. B. Letaief, A survey on mobile edge computing: The communication perspective, IEEE COMMUNICATIONS SURVEYS AND TUTORIALS 19 (4) (2017) 2322–2358, times Cited in Web of Science Core Collection: 2049 Total Times Cited: 2083 Cited Reference Count: 237 ER -. doi:10.1109/COMST.2017.2745201.

[12] W. S. Shi, J. Cao, Q. Zhang, Y. Li, L. Y. Xu, Edge computing: Vision and challenges, IEEE INTERNET OF THINGS JOURNAL 3 (5) (2016) 637–646, times Cited in Web of Science Core Collection: 3812 Total Times Cited: 4048 Cited Reference Count: 32 ER -. doi:10.1109/JIOT.2016.2579198.

[13] M. Raeisi-Varzaneh, O. Dakkak, A. Habbal, B. S. Kim, Resource scheduling in edge computing: Architecture, taxonomy, open issues and future research directions, IEEE ACCESS 11 (2023) 25329–25350, times Cited in Web of Science Core Collection: 33 Total Times Cited: 33 Cited Reference Count: 186 ER -. doi:10.1109/ACCESS.2023.3256522.

[14] 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, 2022-04-27 Article (2022). doi:10.1016/j.comnet.2021.108710.

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

[16] R. Zhang, H. Xia, Z. J. 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.

[17] C. Zeng, X. W. Wang, R. F. Zeng, Y. Li, J. Z. 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-THE INTERNATIONAL JOURNAL OF ESCIENCE 155 (2024) 121–

131. doi:10.1016/j.future.2024.01.025.

[18] R. T. Xie, J. H. Fang, J. M. Yao, X. H. Jia, K. S. 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.

[19] M. Tang, V. 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.

[20] 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, 2024-03-26 Article (2024). doi:10.1016/j.iot.2024.101081.

[21] 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, 2023-07-27 Article. doi:10.1109/TSC.2022.3197881.

[22] 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, 2025-02-27 ER - Proceedings Paper (2024-01-01 2024).

[23] Y. Mao, B. He, S. Zhou, C. Ma, Z. Wang, IEEE, Collaborative edge caching: a meta reinforcement learning approach with edge sampling, 2023-11-05 ER - Proceedings Paper (2023-01-01 2023).

[24] M. Zhao, M. R. Nakhai, IEEE, Deep reinforcement learning based two-phase proactive caching for collaborative edge networks, 2024-09-21 ER - Proceedings Paper (2024-01-01 2024).

[25] X. J. Kong, G. H. Duan, M. L. Hou, G. J. Shen, H. Wang, X. R. 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, times Cited in Web of Science Core Collection: 64 Total Times Cited: 65 Cited Reference Count: 30 ER -. doi:10.1109/TII.2022.3155162.

[26] 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, times Cited in Web of Science Core Collection: 0 Total Times Cited: 0 Cited Reference Count: 30 ER -. doi:10.1109/TVT.2023.3312340.

[27] 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, 2023-08-26 Article. doi:10.1109/JIOT.2022.3229009.

[28] 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, 2024-07-28 Article. doi:10.1109/JIOT.2024.3392329.

[29] 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, journal Article http://www.syndetics.com/index.aspx?isbn=/sc.gif& issn=2327-4662&

client=summontrial. doi:10.1109/JIOT.2021.3052542.

[30] 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, 2024-08-22 Article. doi:10.1109/TITS.2024.3429305.

[31] Z. Xue, C. Liu, C. L. Liao, G. J. Han, Z. G. 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, times Cited in Web of Science Core Collection: 22 Total Times Cited: 22 Cited Reference Count: 41 ER -. doi:10.1109/TVT.2023.3234336.

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

[33] 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, 2024-06-25 Article. doi:10.1016/j.comcom.2024.05.020.

[34] 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, 2024-06-27 Article. doi:10.1109/JIOT.2024.3382723.

[35] W. J., W. J., C. Q., Y. Z., Z. P., W. X., F. C., 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, iEEE Transactions on Network Science and Engineering. doi:10.1109/TNSE.2021.3075530.

[36] S. David, L. Guy, H. Nicolas, D. Thomas, W. Daan, R. Martin, Deterministic policy gradient algorithms.

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

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