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

**Abstract**

In recent years, vehicular edge computing (VEC) has become 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 scheme offers three main advantages: 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. In order to thoroughly assess the effectiveness of DRLSCCO, 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 computation 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

computation 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 vehicular applications with computational intensity and strict latency requirements. [1, 2]. 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 [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 task 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 nodes with the corresponding service program can process a task. 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 how services are cached and how tasks are offloaded to 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 computation delay. This paper makes the following primary contributions:

1. We propose a 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 average 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 type, 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 offload-

ing 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 computation delay across diverse scenarios and exhibits minimum growth rate of energy consumption as edge server 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 DRLSCCO scheme, with experimental results and analysis provided in Sec. 5. Finally, Section 6 concludes the paper and suggests future work.

## 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 task computation 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] 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 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 computation 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 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 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 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. [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 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. [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 an algorithm which integrates dynamic programming based service caching with game theory for computation offloading. This algorithm jointly considers caching preferences 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. The goal of this paper is to minimize the average task computation delay and maintain a reasonable and acceptable growth rate of MNO's energy consumption.

## 3. System Model

First, we provide an overview of our system, and then, we elaborate on the system from four perspectives: the communication model, the service generation model, the joint service caching and computation 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 paper are summarized in Table 1.

### *3.1. System Overview*

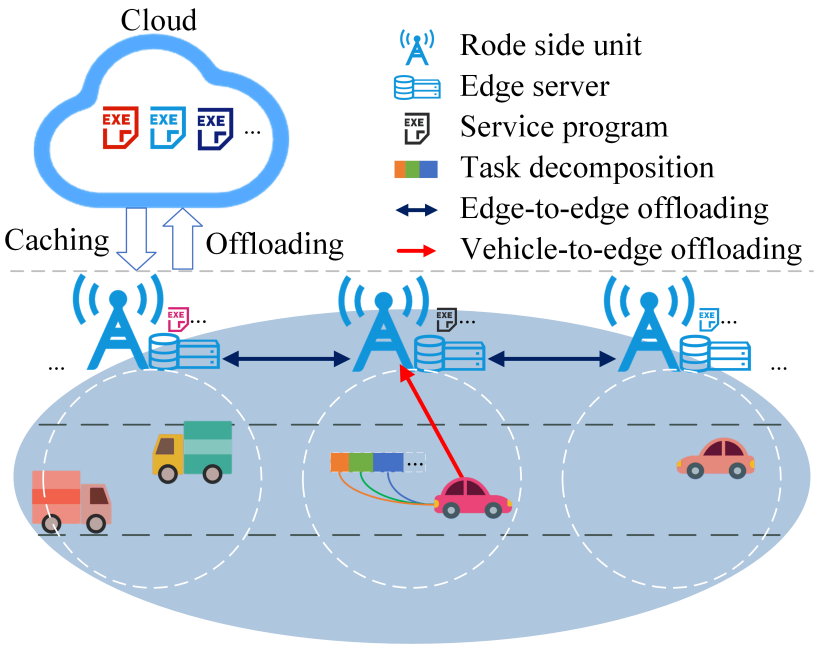


Figure 1: System model.

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 |
| $d_j$ | 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 CPU frequency of a vehicle |
| $f^R$ | The CPU frequency of an edge node |
| $\omega^V_{l,j,k}$ | The offloading proportion of subtask $s_{j,k}$ from vehicle $l$ to edge node |
| $\omega^R_{n,j,k}$ | The offloading proportion of subtask $s_{j,k}$ from edge node $n$ to cooperative edge node |

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. 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. The RSUs is denoted by $\mathcal{N} = \{n \mid n = 1, 2, \ldots, N\}$ and the vehicles is denoted by $\mathcal{L} = \{l \mid l = 1, 2, \ldots, L\}$. Here, $L = \sum_{n=1}^{N} \rho_n(t)$ and $\rho_n(t)$ denotes the vehicle density 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 $\Delta t$.

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 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 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 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 $\sigma^2$ 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 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 with a skewness parameter $z > 0$ [30]. 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 positive 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 task generation patterns in VEC networks.

### *3.4. Service Caching and Task Offloading Model*

We assume that a task can be processed within its originating time slot. An agent is deployed on the cloud server, and it determines the caching strategy for each type of service at both vehicles and edge nodes before 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 $\beta$ represents the size of each 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 vector is a binary variable indicating the caching status of a specific service: 1 means the service is cached, while 0 indicates 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 continuous variables $\omega_{l,j,k}^V(t)$ and $\omega_{n,j,k}^R(t)$ denote offloading proportion of subtask $s_{j,k}(t)$ of vehicle-to-edge and edge-to-edge, respectively. The offloading proportion 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 further verifies whether the associated edge node $n$ also caches the service, then determines the optimal execution location for each subtask, choosing between local or partial offloading to an edge node. If the service is not cached at vehicle $l$, the subtask is fully uploaded to the associated edge node $n$. Based on service caching states of edge node $n$ and its cooperative edge nodes, the agent then decides

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 |

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 requirements, one of them is randomly selected for collaboration. If neither the vehicle, the edge node, nor the cooperative edge nodes cache 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 $\lambda$ 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 $r^{\text{e2e}}$ represents data transmission rate.

The execution delay at the cooperative edge node is expressed as

$$D^{\mathrm{co}}_{m,j,k}(t) = (1 - \delta^{V}_{l,k}(t))\delta^{C}_{m,k}(t)\omega^{V}_{l,j,k}(t)\omega^{R}_{n,j,k}(t)\frac{\lambda d_{j,k}(t)}{f^{R}}. \tag{10}$$

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

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

where $r^{\mathrm{e2c}}$ denotes the data transmission rate.

Since the computational results are substantially smaller than the raw input data, and cloud server possess abundant computing resource, the delay caused by result transmission and execution on cloud server is negligible [35].

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

$$\begin{aligned} D^{\mathrm{total}}_{j,k}(t) = \max\Bigg\{ & D^{\mathrm{local}}_{l,j,k}(t), D^{\mathrm{v2e}}_{l,j,k}(t) + \max\Big\{ D^{\mathrm{edge}}_{n,j,k}(t), \\ & D^{\mathrm{e2e}}_{n,j,k}(t) + D^{\mathrm{co}}_{m,j,k}(t), D^{\mathrm{cloud}}_{n,j,k}(t) \Big\} \Bigg\}. \end{aligned} \tag{12}$$

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

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

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

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

Referring to [25], we also consider MNOs' energy consumption in our

study. The energy consumption for processing subtask $s_{j,k}(t)$ is given by

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

Here, parameter $\kappa$ represents the processor energy efficiency parameter of edge server, which typically depends on hardware conditions. Parameters $p^{\text{e2e}}$ and $p^{\text{e2c}}$ 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

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

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

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

*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*

To minimize task average computation delay, the problem of service caching and task offloading is formulated as a MINLP optimization problem. Let $\mathbf{Z}(t)$ denote 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.

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

s.t.

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

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

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

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

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

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

$$\beta \cdot \boldsymbol{\delta}^R_n(t)\mathbf{e} \leq S^R. \tag{38f}$$

$$E^{\text{total}}_j(t) < E^{\max}, \tag{38g}$$

In Eq. (38), constraints (38a) and (38b) ensure that the offloading proportions of a subtask are continuous values within the range of $[0, 1]$. Parameters $X^R_{l,k}(t)$ 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 (38c) and (38d) specify that the caching states of type-$k$ service at a vehicle or an edge node

are binary. Constraints (38e) and (38f) ensure that the cached services do not exceed the storage capacity of a vehicle or an edge node. Constraint (38g) restricts the energy consumption to be less than $E^{\max}$, where $E^{\max}$ is the maximum value of energy consumption under service different caching states.

## 4. The Proposed DRLSCCO Scheme

Traditional model-based approaches struggle to adapt to dynamic environments and heuristic algorithms lack the generality for unpredictable VEC scenarios. To overcome this, we employ the DDPG algorithm, an actor-critic method that integrates deep neural networks with reinforcement learning. The algorithm enables an agent to interact with the environment and make decisions based on experiences, especially suited for high-dimensional systems like VEC [36]. Based on this formulation, we model the optimization problem presented in (38) as a Markov Decision Process (MDP) and propose DRLSCCO, a DDPG-based scheme for joint service caching and computation offloading, designed to dynamically optimize decision.

### *4.1. Problem Formulation Based on RL*

The Markov property fundamentally defines 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 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 at time slot $t$ can be expressed as

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

*4.1.2. Action space*

With the state observed at the beginning of a time sloths, the agent determines offloading locations and offloading proportion, while also updating the service caching states. The action 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 at time slot $t$ can be expressed as

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

*4.1.3. Reward*

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 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

$$r_t = -D^{ave}(t). \tag{41}$$

*4.2. The DRLSCCO Scheme*

To effectively learn and converge to the optimal policy, we propose the DRLSCCO scheme. The scheme incorporates policy networks, target networks and an experience replay buffer. These components work in concert

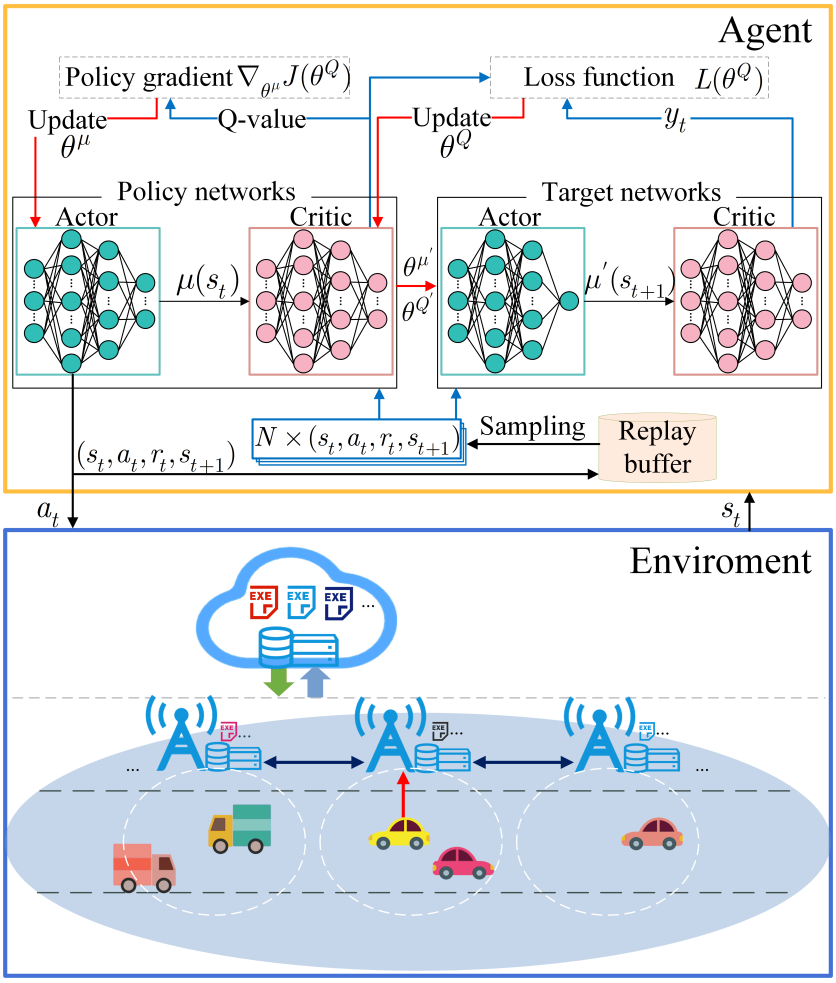


Figure 2: The framework of the DRLSCCO.

to stabilize the training process and ensure the agent's policy is progressively converged through iterative interactions with the environment. The framework of the DRLSCCO scheme is illustrated in Fig. 2.

As shown in Fig. 2, the actor networks of policy networks 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 critic networks of policy

networks estimate 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

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

Here, parameter $\alpha$ is the discount factor, while $\theta^\mu$ and $\theta^Q$ represent the actor and critic networks of the policy networks, respectively.

Eq. (42) adheres to a recursive relationship known as the Bellman equation, which can be written as

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

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 $y_t$ is constructed by replacing the Q-value and the policy with their target networks' counterparts. The target Q-value is defined as

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

Here, parameters $\theta^{\mu'}$ and $\theta^{Q'}$ represent the actor and critic networks 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 action $a_t$. The modified action $a_t$ is defined as

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

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

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

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

where $I$ is the minimum batch size.

$\theta^\mu$ is updated by following the deterministic policy gradient to maximize the Q-value. The gradient update can be expressed as

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

At each iteration, $\theta^Q$ is updated according to

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

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

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

where $\eta$ 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 $\theta^{Q}$ and $\theta^{\mu}$. At each iteration, $\theta^{Q'}$ is updated according to

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

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

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

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

The DRLSCCO scheme is detailed in Algorithm 1.

## 5. Simulation Experiments and Performance Analysis

We conduct simulation experiments under different parameter settings to analyze the performance of the proposed scheme DRLSCCO. The main parameters used in the experiments are summarized in Table 3, the others refer to [37, 38].

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

1. No comparative edge nodes (NCE): This scheme disables cooperation among edge nodes, restricting subtask execution to three locations: the local vehicle, the edge node and the cloud server.

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

**Input:** $\mathcal{L}$, $\mathcal{N}$, $\mathcal{T}$, $\mathcal{K}$.

**Output:** $\theta^{\mu}$.

Initialization: $\theta^{\mu}$, $\theta^{Q}$, $\theta^{\mu'}$, $\theta^{Q'}$, the soft update coefficient $\tau$, the number of episodes, the batch sample size $i$, the experience replay buffer size $I$, the random noise $n_t$, an empty experience replay buffer.
**for** each episode **do**
  Initialize the state $s_0$ of time slot 0.
  **for** $t = 0, 1, \ldots, T-1$ **do**
    The agent selects action $a_t$ according to Eq. (45).
    Vehicles and edge nodes execute the action.
    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 $(s, a, r, s_{\text{next}})$ from the replay buffer.
      Calculate the Q-value by Eq. (42).
      Calculate the target Q-value by Eq. (44).
      Update $\theta^{Q}$ by Eq. (46).
      Update $\theta^{\mu}$ by Eq. (47).
      Perform a soft update on $\theta^{Q'}$ and $\theta^{\mu'}$ according to Eqs. (50) and (51), respectively.
    **end if**
  **end for**
**end for**

Table 3: Main parameters

| **Parameter** | **Meaning** | **Value** |
|---|---|---|
| $\eta^{\text{actor}}$ | The learning rate of actor networks | 0.001 |
| $\eta^{\text{critic}}$ | The learning rate of critic networks | 0.002 |
| $\alpha$ | The discount factor | 0.99 |
| $\rho^{\max}$ | The max density of vehicles within range of an edge node | 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 |
| $r^{\text{e2e}}$ | The transmission rate between adjacent edge nodes | 10 Mbps |
| $r^{\text{e2c}}$ | The transmission rate from an edge node to cloud server | 8 Mbps |
| $p^{\text{2e2}}$ | The transmission power between adjacent edge nodes | 1 W |
| $p^{\text{e2c}}$ | The transmission power from an edge node to cloud server | 2 W |

2. Least Recently Used (LRU)-based service caching replacement on vehicles and edge nodes (LRUSC): The caching decisions for both vehicles and edge nodes are made by the agent. The LRU policy is used to updating cache when the storage capacity is insufficient.
3. Random service caching and task offloading (RSCCO): The caching decisions for both vehicles and edge nodes, along with the offloading proportions for subtasks, are all generated through a stochastic process.
4. No edge nodes (NE): In the absence of edge nodes, each subtask must be processed either locally within the vehicle or remotely through cloud server offloading.

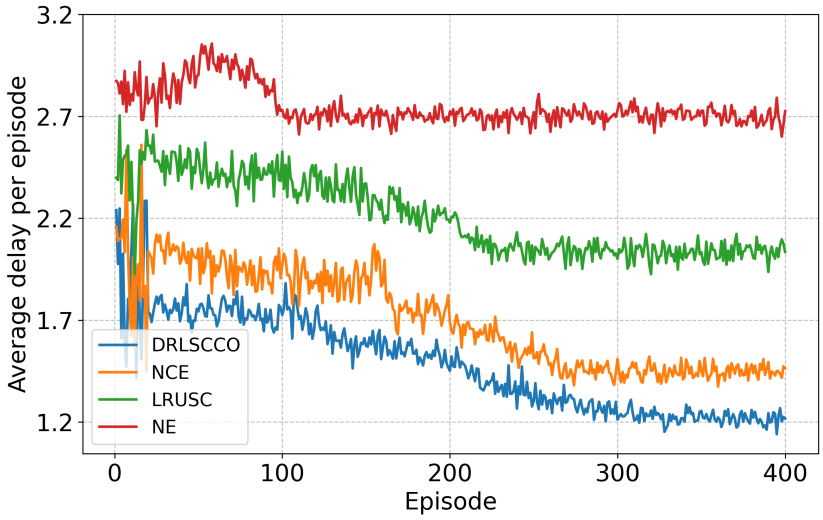


Figure 3: The average computation delay per episode

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 the schemes.

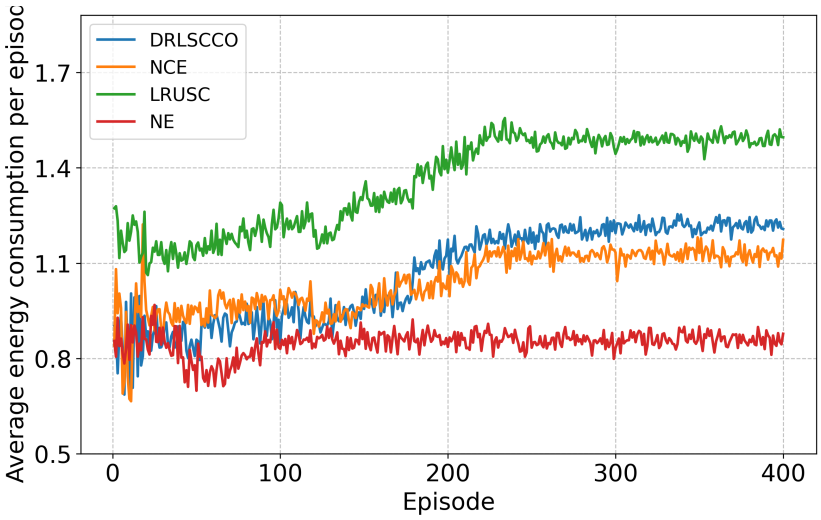


Figure 4: The average energy consumption per episode

As shown in the 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. This trend occurs because that deploying edge nodes and cloud server reduces execution delay but consumes more energy. Among all benchmark schemes, 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 of DRL-based approaches. This limitation in caching strategy consequently leads to inferior offloading decisions and higher transmission delay. In the case of NE, where edge nodes are absent, computation delay is dominated by local execution delay and cloud transmission delay, resulting in the highest computation delay and revealing the critical role of

edge computing. These results confirm that the proposed DRLSCCO 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.

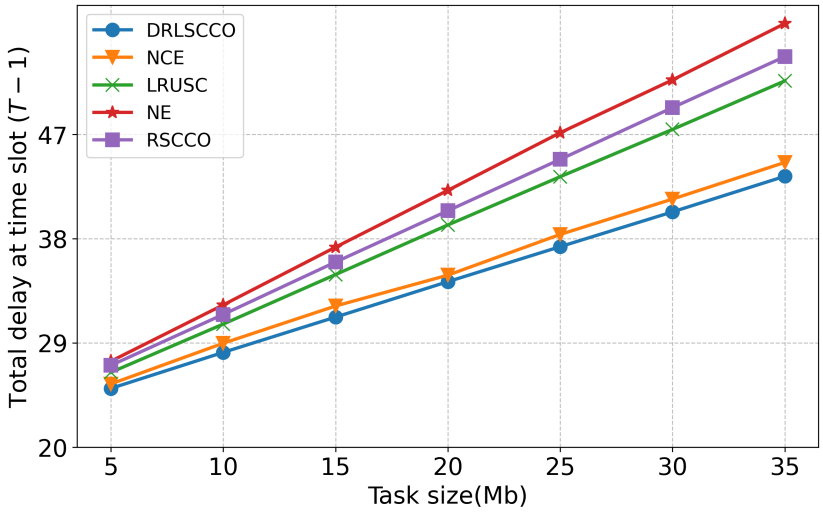


Figure 5: The impact of task data size on total computation delay

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

From Fig. 5, it can be clearly observed that as the data size increases from 5 Mb to 35 Mb, the 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 delays differences among the schemes become more pronounced. The underlying reason for this phenomenon lies in the shift from resource-abundant to resource-constrained as the data size grows. For smaller data size, the computing resource is underutilized, masking the deficiencies of LRUSC, NE and RSCCO. Under heavy load, however, intelligent resource management becomes important. DRLSCCO excels in this context by continuously learning to optimize caching and offloading decisions. For example, NE and NCE are forced to use the cloud server too often, and LRUSC or RSCCO make short-sighted or random choices. By making globally efficient decisions, DRLSCCO achieves the lowest computation delay under different levels of computational demand.

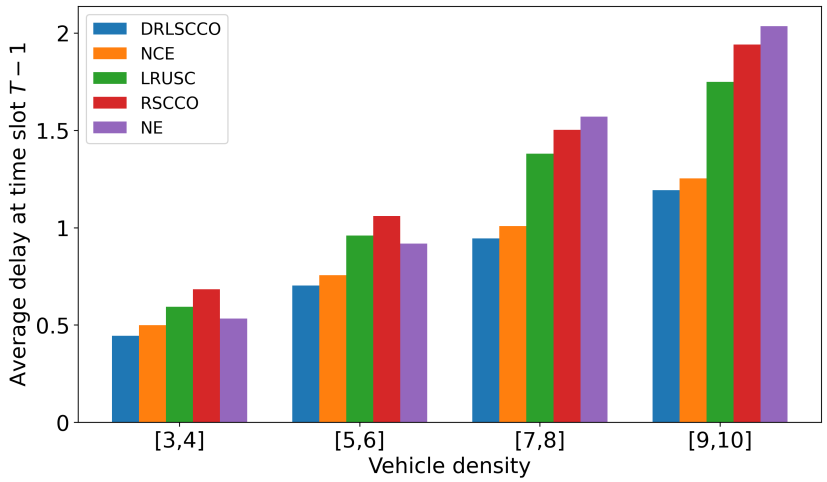


Figure 6: The impact of vehicle density on average computation delay

To isolate the impact of network scale, Figure 6 present an investigation

into how vehicle density affects computation delay by fixing the task data size at 20 Mb. This controlled 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 in all schemes rises due to increased demand on the communication channel, which reduces bandwidth of a vehicle and leads to higher transmission delay. When the vehicle density is relatively lower, 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 computation delay. Consequently, in low-density scenarios, NE exhibits a lower average computation delay compared to LRUSC and RSCCO, but this advantage vanishes as vehicle density increases. Notably, DRLSCCO consistently achieves the lowest computation delay as vehicle density varies, highlighting its strong adaptability and effectiveness at different network scales.

With the vehicle density and task data size fixed as 10 and 20 Mb, respectively, we investigate the trends of total computation delay and total energy consumption as the CPU frequency of edge nodes increased 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 computation delay but leads to increased energy consumption. Since NE does not utilize edge nodes, its computation delay and energy consumption remain constant. Both LRUSC and RSCCO exhibit considerable fluctuations in computation delay and energy consumption. However, as the offloading decisions of LRUSC are governed by an agent, its

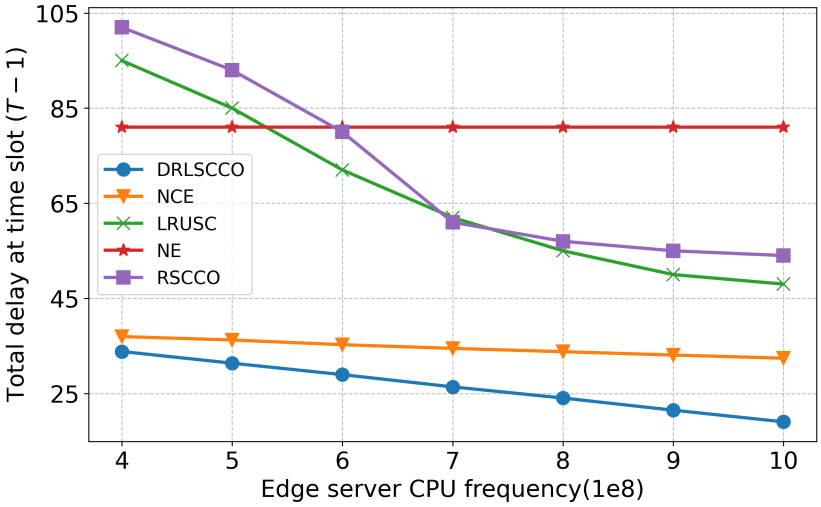


Figure 7: The influence of edge nodes CPU frequency on computation delay

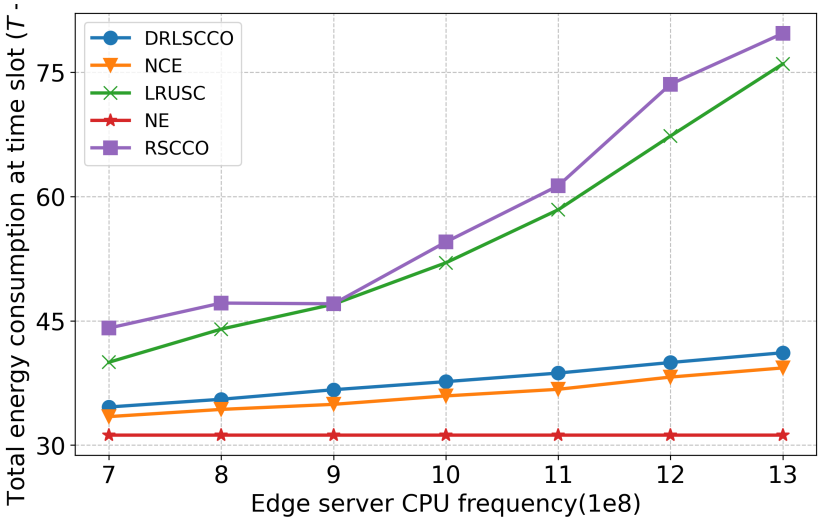


Figure 8: The influence of edge node CPU frequency on energy consumption

performance trend demonstrates relative stability compared to RSCCO. An observation is that LRUSC and RSCCO exhibit higher computation delays than NE at a lower CPU frequency. This phenomenon indicates that the transmission delay for offloading a subtask to the cloud server could be less than that for executing a subtask on an edge node with constrained computing resource. This also suggests that in order to fully realize the advantages of edge computing, MNOs need to deploy edge nodes with rich computing resource. The computation delays of DRLSCCO and NCE are relatively close at a lower CPU frequency. As the CPU frequency increases, the delay gap between DRLSCCO and NCE gradually widens. This is because that with limited computing resource, the cooperative offloading in DRLSCCO introduces extra transmission delay. When computing resource becomes sufficient, the delay is increasingly offset by the growing advantages of cooperative offloading. Although DRLSCCO 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, improving user experience quality. The results show that as the CPU frequency increases, DRLSCCO consistently maintains the lowest total computation delay across all schemes while managing to restrict the energy consumption within a reasonable range.

## 6. Conclusion

In this paper, we investigate a joint service caching and task offloading problem in a VEC system. To minimize task computation delay, we formu-

lated the joint optimization problem as a MINLP problem and proposed a DRL-based scheme named DRLSCCO to solve it. First, the scheme supports heterogeneous service requirements, i.e., a task request multiple types of services simultaneously. Second, the scheme enables fine-grained task decomposition based on service types and distributed offloading of subtasks. Third, we introduce six caching and offloading states and conduct extremum analysis of computation delay and energy consumption for each, providing a theoretical foundation for agent decision-making. Training results show that both computation delay and energy consumption gradually converge as caching and offloading decisions are updated. We further evaluated the performance of DRLSCCO under various scenarios: 1) with fixed vehicle density, we assessed the total computation delay under increasing computational demands, 2) with fixed task size, we analyzed the average computation delay as the number of communication links increases, and 3) we examined the trends of total computation delay and total energy consumption as the CPU frequency of edge nodes increases. Experimental results demonstrate that DRLSCCO achieves the lowest computation delay over all compared schemes while maintaining reasonable and acceptable energy consumption growth.

While DRLSCCO operates within a dynamic vehicular network environment, certain idealized conditions remain, such as the assumption of continuous communication between vehicles and edge nodes. Future work will focus on overcoming intermittent communication to enable the continuous transmission of tasks.

## 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 TRANSAC-

TIONS 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.