CoOR: Collaborative Task Offloading and Service Caching Replacement for Vehicular Edge Computing Networks

Zhen Li, Chao Yang, Xumin Huang, WeiLiang Zeng, and Shengli Xie, *Fellow, IEEE*

***Abstract*—In vehicular edge computing networks, edge service caching has emerged as a promising technology that supports delay sensitive applications. When the vehicles pass the coverage areas of roadside units (RSUs), the vehicles can offload part/all of their tasks to the RSUs that had cached the related service data in advance. However, it is difficult to utilize the service caching resources in RSUs efficiently, for the high mobility, variable task computing requests of vehicles and the limited storage and computation capacity of RSUs. In this paper, we propose a collaborative task offloading and service caching replacement scheme from a vehicle perspective, named CoOR, the task processing cooperation between the adjacent RSUs and the service caching replacement for vehicle are mainly considered. We formulate the system computing cost and delay minimization as a mixed integer programming problem, it faces challenges of task offloading and service caching coupling, the heterogeneous computation requests and dynamic data transmission conditions. Thus, we develop an iterative algorithm combined with the Gibbs sampling and deep reinforcement learning (DRL) to find the optimal decisions. Extensive simulation results show that the proposed schemes have good convergence and better performance than the traditional baselines.**

***Index Terms*—Service caching, vehicle edge computing, cache replacement, DRL.**

I. Introduction

With the rapid development of internet of things and AI technologies, the various emerging applications of intelligent connected vehicles and passengers, such as: self-driving, online gaming, have grown significantly. The huge amount of outsourcing data traffic leads to unpredictable task completion delay in intelligent transportation system (ITS). Vehicular edge computing networks (VECNs), integrating the computation and storage resources of both vehicles on road and RSUs, have emerged as the paradigms to cope with the challenges by migrating the delay sensitive and resource intensive tasks to the ITS network edges [1]. In VECNs, the edge nodes (e.g., macro-cell base stations, micro-cell base stations (MBSs) and RSUs), which are endowed with the computation and storage capacities, can execute the tasks offloaded from the vehicles and reduce the cost and delay of task execution significantly.

Normally, the required tasks in VECNs can be divided into direct response tasks (e.g., traffic HD-map download [2]) and computation-intensive ones (e.g., vehicular augmented reality (AR) application [3]). For the former, since the large number of required tasks are likely to be similar and need to be dealt with in a timely manner, *Edge content caching* is recognized as an effective method to reduce the redundant data transmission time and bandwidth cost between the edge nodes and the core cloud center. To deal with the conflict between dynamic computation requests and limited storage capacity, the caching placement and replacement problems are mainly considered and the corresponding task offloading schemes are also studied in [4]–[6]. However, the complexity of pending tasks makes the existing content caching and task offloading schemes inefficient, especially in the fast moving VECNs environment.

For the latter, it is hard to cache the task computing results under the dynamic requests, and the execution of some complex tasks (e.g., pattern recognition) needs the corresponding database support. *Edge service caching*, in which the program data or the related databases are cached in the edge nodes, enables the tasks to be executed at the wireless network edge. Compared with the content caching, the service caching covers a set of similar and interrelated tasks from different users. For a special task, the task execution should be performed on edge nodes which had completed the corresponding service caching. For example, a well-trained deep learning model should be configured at the edge nodes for the recognition task requests. It makes new challenges for the service caching selection, placement and the task offloading decisions. In [7], the service caching placement, task offloading and resource allocation are jointly optimized to minimize the task completed delay and energy consumption of the mobile users. In [8], the authors give an efficient service caching selection and task scheduling strategy in the multi-access edge computing networks. In [9], the cooperative service caching and workload scheduling is investigated. In [10], the joint edge service caching and the storage-computing-communication management scheme for VECNs is analyzed. Moreover, the dynamic service placement and service migration are considered in [11], [12]. However, in the view of prior works, most of them consider the service caching problems at the wireless network side only. Actually, in VECNs, as the vehicles become smarter and more advanced, the optimal collaborative task offloading and service caching between the vehicles on road and RSUs can improve the system performance directly.

In this paper, we consider a scenario that a vehicle with a sequence of inter-dependent computation tasks moves pass the coverage areas of RSUs. Both the vehicle and RSUs have pre-cached a set of service data. The vehicle can offload part/all of tasks to nearby RSUs for processing. We analyze the service caching replacement and task offloading in VECNs from a vehicle perspective. The vehicle does not have the rights to ask the RSUs update their service caching data immediately, it can replace its own caching data. The main objective is to minimize the total system cost of task execution for the

This work was supported by National Natural Science Foundation of China under Grants 62001125, 62273102, 62003094 and U1911401. (*Corresponding author: Chao Yang.*)

Zhen Li, Chao Yang are with the School of Automation, Guangdong University of Technology, and also with the Guangdong Key Laboratory of IoT Information Technology, Guangzhou, China. Email: z.li2281@qq.com, yangchaoscut@aliyun.com

Xuming Huang and Weiliang Zeng are with the 111 Center for Intelligent Batch Manufacturing Based on IoT Technology, Guangdong University of Technology, and also with the Key Laboratory of Intelligent Detection and the Internet of Things in Manufacturing, Ministry of Education, Guangzhou, China. Email: huangxu_min@163.com, weiliangzeng@gdut.edu.cn

Shengli Xie is with the Guangdong-HongKong-Macao Joint Laboratory for Smart Discrete Manufacturing, and also with the Joint International Research Laboratory of Intelligent Information Processing and System Integration of IoT, Guangzhou, China. Email: shlxie@gdut.edu.cn.

vehicle, which includes the time and computing costs. There are three critical problems need to be addressed. 1) *How to perform task offloading for VECNs with service caching*? The service caching conditions on RSUs and the mobility of vehicles affect the offloading decision directly. In addition, the cooperation between the two adjacent RSUs makes the task offloading complexity. 2) *How to perform caching replacement for the vehicle*? The high mobility and variable task computing requests of the vehicle require the replacement of service caching data to be performed more precisely and carefully. 3) *How to find the optimal solution of the proposed problem*? The service caching and the task offloading are coupled to each other, and the dynamic traffic network conditions make the solution of problem difficult.

To address the above problems, we propose a CoOR scheme to jointly orchestrate the task offloading and service caching replacement decisions. The main contributions of this paper are summarized as follows.

- We construct an analytical model of service caching enabled VECNs from the vehicle perspective, the service caching replacement for vehicle, the collaborative computation task offloading between RSUs, and the inter-dependency of tasks are mainly considered.
- We formulate a mixed integer non-linear programming problem for the service caching replacement and task offloading decisions, to minimize the total system cost of vehicle's task execution.
- We design an iterative algorithm combined with the Gibbs sampling and DRL to find the proposed problem solutions, under the task offloading and service caching coupling, the heterogeneous computation requests and dynamic data transmission conditions.

Moreover, extensive simulation results are proposed to verify that our proposed algorithm is superior to the baselines under different scenarios.

II. System Model and Problem Formulation

A. VECN Model

As shown in Fig. 1, we consider an urban VECN scenario, including one MBS and two adjacent RSUs, to provide services for the covered vehicles. The LTE-V2X is used to support the V2I and V2V communications [13]. Denote the computation and storage capacity of the RSU m as F_m and C_m. Meanwhile, the local ones of vehicles are set as F_0 and C_0, and $F_m \gg F_0$, $C_m \gg C_0$. A single vehicle executes a sequence of I tasks for an application (e.g., vehicular AR). Formulate the task set as $\mathcal{W} = \{w_1, \ldots, w_i, \ldots, w_I\}$, where each task w_i is characterized by a tuple (ρ_i, ϕ_i, d_i). ρ_i denotes the data size, ϕ_i and d_i denote the computation requirement and the maximum latency of the task, respectively. The I tasks are inter-dependent that the execution of task w_{i+1} requires the result of task w_i [14], [15].

Considering the service caching framework, all the task executions require the specific services support. These tasks can only be offloaded to the edge nodes cached the corresponding services. Set the ITS provides a library of services $\mathcal{S} = \{s_1, \ldots, s_k, \ldots, s_K\}$ and η_k is the required storage

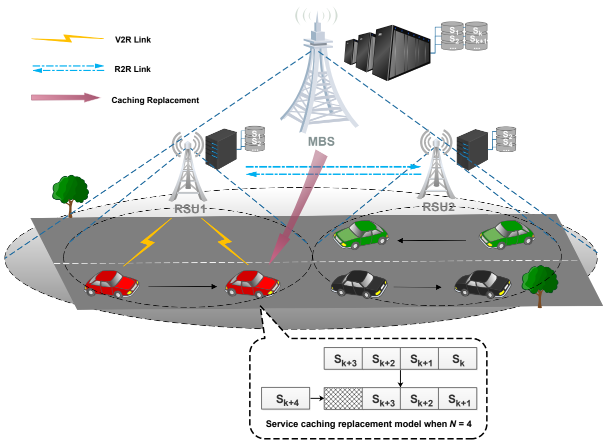

Fig. 1. System model.

capacity to cache service s_k. $v_{i,k} = 1$ means that the task w_i can be processed by service s_k, $v_{i,k} \in \{0, 1\}$. In this paper, we set the services had been cached at RSUs according to the long-term historical data collection, and wouldn't be updated frequently [14]. Denote a matrix $E^{(2,K)}$ as the RSU's service caching status, $e_{m,k} = 1$ means RSU m had cached service s_k. Specially, connected with the centralized cloud server via wired communication links, we assume the MBS can provide all services for the covered users.

Similarly, denote $u_k \in \{0, 1\}$ as the caching policy of the vehicle, $u_k = 1$ means it had cached the service s_k, and

$$\sum_{s_k \in S} u_k \cdot \eta_k \leq C_0. \tag{1}$$

B. Vehicular Service Caching Replacement

Differing from the RSUs, the vehicle can replace its local caching service data when the corresponding task execution is finished. However, it cannot ask the RSU to change its current caching strategy immediately. For simple, we set that the local service caching replacement of vehicle follows the sequential method *FIFO* (First In and First Out). As shown in Fig. 1, the vehicle caches a set of different services. After the execution of task k, it decides whether to free the space of service s_k and download other services. Let $o_i^N \in \{0, 1\}$ as the caching replacement decision, $o_i^N = 1$ means the service s_i is replaced by service s_{i+N}. N is the maximum number of services that the vehicle can cache, and

$$\sum_{k=1}^{N} \eta_{s_{i+k}} \leq C_0. \tag{2}$$

Meanwhile, downloading service data from MBS is payable, the cost is related to the data size of the services. Thus,

$$L_{i+N}^{download} = \Gamma_{cloud} \cdot \eta_{i+N}, \tag{3}$$

where $\Gamma_{cloud} > 0$ is the cost parameter. Specifically, the vehicle can cache the service data that the RSUs hadn't cached. For the services cached at the RSUs, the vehicle may also cache part of them optimally. Although the payment for MBS is needed, the task transmission delay and the RSU handover cost for task calculation can also be reduced, when the vehicle caches the corresponding services locally. Thus, the optimal service replacement and task offloading decisions are given for the tradeoff between the system cost and the task completion delay.

C. Computation Model

1) Local Computing: The vehicle can execute some of tasks locally without the cost, that is attainable only when the following conditions hold: it cached the corresponding services before, it had received the results of the previous tasks, and the local computing capacity is enough. Equivalently, the first condition is expressed as

$$\sum_{j=1}^{K} v_{i,j} \cdot u_j \geq 1. \tag{4}$$

Let $\alpha_i \in \{0,1\}$ as the task offloading decision of vehicle. $\alpha_i = 0$ if it decides to execute task w_i locally, otherwise, $\alpha_i = 1$. The local computing delay of task w_i is

$$T_i^{local} = \frac{\phi_i}{F_0}. \tag{5}$$

2) Task Offloading: The vehicle can also offload part/all of tasks to the RSU through V2I communication link. When $\alpha_i = 1$, similar to the local computing, the service caching constraint is given as

$$\sum_{j=1}^{K} e_{m,j} \cdot v_{i,j} \geq 1. \tag{6}$$

The LTE-V2X technology is used for the V2I transmission. In this paper, following the Third Generation Partnership Project (3GPP) cellular V2X resource allocation model 3 [13], we set that the vehicles obtain their own transmission channel via the schedule of RSU. For the vehicle moves pass the coverage of RSUs, the V2I transmission data rate between RSU m and the vehicle with task w_i, $r_{V2R,m,i}$ is

$$r_{V2R,m,i} = B_m log_2\left(1 + \frac{p h_m (l_i)^{-r}}{\sigma_m^2}\right), \tag{7}$$

where h_m and B_m denote the V2R channel gain and bandwidth. σ_m^2 is power of the Gaussian noise, p denotes the transmission power. l_i is the distance between the position where vehicle starts to transmit task w_i and the RSU, which is generated as ref. [16]. Hence, the uploading time of task w_i to RSU m is

$$T_{m,i}^{upload} = \frac{\rho_i}{r_{V2R,m,i}}. \tag{8}$$

In this paper, we set that all of the covered vehicles can offload computation tasks to the RSU. When the objective vehicle sends the task, a queuing delay may be needed at the RSU. Assume the tasks arrival RSU m is a Poisson process with parameter λ_m [9]. The RSU performs the computation service for the arriving tasks sequentially with *FCFS* (First Come and First Service) scheme, and it can obtain the average service rate via long-term observations and statistics, as μ_m. For each RSU m , the queuing process of tasks is modelled as M/D/1 [17], and the expected queuing delay is

$$T_m^{queue} = \frac{\lambda_m}{2\mu_m(\mu_m - \lambda_m)}. \tag{9}$$

Then, the processing delay of the task w_i in RSU m is expressed as

$$T_{m,i}^{process} = \frac{\phi_i}{F_m}. \tag{10}$$

The total execution delay of the task w_i served by RSU m, including the queuing delay and processing delay, is given as

$$T_i^m = T_m^{queue} + T_{m,i}^{process}. \tag{11}$$

We consider the collaboration between the two adjacent RSUs. When the task queue is long, the RSU m can share the computing load with the adjacent RSU via wired links [18]. Denote the collaborative computing decision is $\beta_i \in \{0,1\}$, $\beta_i = 1$, if the RSU m offloads the task w_i to RSU m'. The RSU m' also has the constraint as follows

$$\sum_{j=1}^{K} e_{m',j} \cdot v_{i,j} \geq \beta_i. \tag{12}$$

Denote $r_{m,m'}$ as the data rate between RSUs, the transmission delay for offloading task w_i data to RSU m' is

$$T_i^{trans} = \frac{\rho_i}{r_{m,m'}}. \tag{13}$$

Similarly, we have

$$T_{m'}^{queue} = \frac{\lambda_{m'}}{2\mu_{m'}(\mu_{m'} - \lambda_{m'})}. \tag{14}$$

$$T_{m',i}^{process} = \frac{\phi_i}{F_{m'}}. \tag{15}$$

The execution delay of task w_i served by RSU m' is

$$T_i^{m'} = T_i^{trans} + T_{m'}^{queue} + T_{m',i}^{process}. \tag{16}$$

In summary, if the vehicle selects the task offloading scheme, the total execution delay for task w_i is

$$T_i^{offloading} = T_i^{upload} + (1-\beta_i) T_i^m + \beta_i T_i^{m'} + T^{back}, \tag{17}$$

where T^{back} is the delay for RSU result return, which is fixed as the effect on the overall latency is negligible.

On the other hand, denote the price of RSU computing resources as Γ_{RSU} RMB/(M cycle), the execution cost of task w_i is formulated as

$$L_i^{offloading} = \Gamma_{RSU}\phi_i. \tag{18}$$

D. Problem Formulation

According to the VECN framework, the total task execution delay and cost of the vehicle are summarized as

$$T^{sys} = \sum_{i=1}^{I}\left[(1-\alpha_i)T_i^{local} + \alpha_i T_i^{offloading}\right], \tag{19}$$

$$L^{sys} = \sum_{i=1}^{I}\left[\alpha_i L_i^{offloading} + o_i^N L_{i+N}^{download}\right]. \tag{20}$$

This paper jointly optimizes the task offloading and service caching replacement decisions of the vehicle, as well as the collaboration decisions of RSUs, aims to minimize the total system cost of task execution, including time and computing costs. The problem is shown as follows

$$(\boldsymbol{P1}): \underset{\{\boldsymbol{\alpha},\boldsymbol{\beta},\boldsymbol{o}\}}{min} \quad \zeta T^{sys} + L^{sys}$$

$$\begin{aligned}
s.t. \quad & C1: \sum_{s_k \in S} u_k \cdot \eta_k \le C_0, \\
& C2: \sum_{k=1}^{N} \eta_{s_{i+k}} \le C_0, \quad \forall i \in \{1,2,\cdots,I\}, \\
& C3: \sum_{j=1}^{K} v_{i,j} \cdot u_j \ge 1, \quad \forall i \in \{1,2,\cdots,I\}, \\
& C4: \sum_{j=1}^{K} e_{m,j} \cdot v_{i,j} \ge 1, \quad \forall i \in \{1,2,\cdots,I\}, \\
& C5: \sum_{j=1}^{K} e_{m',j} \cdot v_{i,j} \ge \beta_i, \quad \forall i \in \{1,2,\cdots,I\}.
\end{aligned}$$

Here ζ represents the price of unit delay. Constraints (C1) and (C2) ensure that the services which vehicle had cached are less or equal to the storage capacity. Constraints (C3), (C4) and (C5) guarantee that the tasks should be executed with the corresponding services correctly.

III. Algorithm Design

P1 is a mixed-integer programming problem, which is hard to solve directly. However, since the effect of decision $\boldsymbol{o}$ is reflected in the next few steps, while the caching replacement and current task execution can be regarded as two parallel processes due to they wouldn't interfere with each other at the same time. We decompose problem **P1** into two sub-problems, and design a two-layer iterative algorithm, named Gibbs Sampling and DRL method (**G+DQN method**). For the outer layer, we obtain the approximate optimal caching replacement policy based on Gibbs sampling (Algorithm 1). In the inner layer, we propose a DRL based algorithm (Algorithm 2) to minimize the total system cost (**P2**), based on the obtained vehicle caching replacement policy.

$$(\boldsymbol{P2}): \underset{\{\boldsymbol{\alpha},\boldsymbol{\beta}\}}{min} \quad \zeta T^{sys} + L^{sys}$$
$$s.t. \quad (C3), (C4), (C5).$$

A. Gibbs Sampling for Service Caching Replacement

In this section, we elaborate the Gibbs sampling in detail, as Algorithm 1. For each iteration, one of the caching replacement decisions o_n is selected randomly and changed to $\tilde{o}_n$ virtually. Then, the new optimal offloading decisions $\{\tilde{\boldsymbol{\alpha}}, \tilde{\boldsymbol{\beta}}\}$ are obtained by solving **P2**, and the objective values change from f to $\tilde{f}$ (Here, f is defined as the objective function $\zeta T^{sys} + L^{sys}$). Afterward, the decision is updated to $\tilde{o}_n$ with probability $\psi = \frac{1}{1+e^{(\tilde{f}-f)/\tau}}$ $(\tau > 0)$ and keeps unchanged (i.e., o_n) with probability $1-\psi$. The parameter τ, referred as the smooth parameter, is used to control exploration versus exploitation (i.e., the degree of randomness). When τ is small, the decision $\tilde{o}_n$ that yields lower system cost will be picked with a higher probability. Finally, the iteration ends until the stop criterion is satisfied.

Algorithm 1 Gibbs Sampling for Service Caching Replacement

```
Input: Task set $\mathcal{W}$, service set $\mathcal{S}$, and matrix $E^{(2,K)}$
Output: Caching replacement policy $\boldsymbol{o}$, and task offloading policy $\{\boldsymbol{\alpha}, \boldsymbol{\beta}\}$.
1:  Randomly initialize $\boldsymbol{o}^0 = \{o_1^0, o_2^0, \ldots, o_{\mathcal{N}}^0\}$.
2:  for $i = 1, 2, \ldots$ do
3:      Randomly select $n \in \mathcal{N}$, make a decision $\tilde{o}_n$,
4:      if $\tilde{o}_n$ is feasible then
5:          Based on the policy $(o_1^{i-1}, o_2^{i-1}, ..., o_n^{i-1}, ..., o_{\mathcal{N}}^{i-1})$, obtain $\{\boldsymbol{\alpha}, \boldsymbol{\beta}\}$ and $f$ by minimizing $\boldsymbol{P2}$,
6:          Based on the policy $(o_1^{i-1}, o_2^{i-1}, ..., \tilde{o}_n, ..., o_{\mathcal{N}}^{i-1})$, obtain $\{\tilde{\boldsymbol{\alpha}}, \tilde{\boldsymbol{\beta}}\}$ and $\tilde{f}$ by minimizing $\boldsymbol{P2}$,
7:          $\psi \leftarrow \frac{1}{1+e^{(\tilde{f}-f)/\tau}}$,
8:          Set $o_n^i \leftarrow \tilde{o}_n$ with probability $\psi$, Set $o_n^i \leftarrow o_n^{i-1}$ with probability $(1-\psi)$.
9:      end if
10:     if the stopping criterion is satisfied then,
11:         return $\boldsymbol{o}^i$ and $\{\boldsymbol{\alpha}^i, \boldsymbol{\beta}^i\}$ .
12:     end if
13: end for
```

Lemma 1: *As $\tau > 0$ decreases, the Algorithm 1 converges with an increasing probability to the global optimal solution of **P1**. When $\tau \to 0$, the algorithm can converge to the globally optimal solution.*

Proof: See refs. [9], [19].

B. DRL Based Algorithm for VECN

In this section, we develop a DQN model to solve the complex task offloading and collaboration decision-making in **P2**. The **P2** is re-formulated as a Markov Decision Process (MDP), three main elements, i.e., state space, action space, and reward function are summarized as follows.

1) State Space: Since the location of vehicle (i.e., l_i) affects the task transmission delay directly. Thus, combined with the data size and computation requirement of task w_i (i.e., ρ_i and ϕ_i), the state vector is formulated as $s_i = [\rho_i, \phi_i, l_i]$.

2) Action Space: In **P2**, the caching replacement decisions of vehicle are given by Algorithm 1. Thus, we focus on two task offloading decisions in vehicle and RSU, the action vector is described as $a_i = [\alpha_i, \beta_i]$.

3) Reward Function: Given the state-action pairs, the reward function $r_i = -(\zeta T_i^{sys} + L_i^{sys})$ is set as the inverse of the overall cost in VECNs.

Via employing a neural network to approximate the Q function, i.e., $Q(s_t, a_t) \approx Q(s_t, a_t|\theta)$, where θ is the parameter of the network, the DQN model is used here. The training loss function is defined as the mean square error (MSE), as

$$Loss(\theta) = \mathbf{E}[(y_t - Q(s_t, a_t|\theta))^2], \tag{21}$$

where the value of y_t represents the Q values approximated by the target network,

$$y_t = r_t + \gamma \underset{a_{t+1}}{max} Q(s_{t+1}, a_{t+1}|\bar{\theta}). \tag{22}$$

Then, we use gradient descent method to update the weights of the Q network. θ is updated in the direction of the minimized loss function, as

$$\theta_{t+1} = \theta_t + \xi_\theta [y_t - Q(s_t, a_t|\theta_t)] \nabla_{\theta_t} Q(s_t, a_t|\theta_t), \qquad (23)$$

where $\xi_\theta \in (0, 1)$ is the learning rate. The primary steps of DQN based algorithm for **P2** is illustrated in Algorithm 2.

Algorithm 2 DQN Based Algorithm for VECNs

```
1: Initialize capacity of replay memory
2: Initialize function $Q(s_t, a_t|\theta)$ with random weights $\theta$,
   target function $Q(s_t, a_t|\bar{\theta})$ with weights $\bar{\theta} = \theta$.
3: for each episode do
4:     Initialize state $s_0$ randomly,
5:     for $t = 0, 1, 2, \ldots, I$ do
6:         With probability $\epsilon$ select $a_t$ randomly, otherwise
           select $a_t = \max_a Q(s_t, a|\theta)$,
7:         Execute action $a_t$ in VECN, obtain next state $s_{t+1}$
           and reward $r_t$.
8:         Store $(s_t, a_t, r_t, s_{t+1})$ into replay memory,
9:         Sample mini-batch of transitions randomly from
           replay memory and calculate loss function via Eq. (21),
10:        Update $\theta$ via Eq. (23),
11:        Reset network parameters $\bar{\theta} = \theta$ with a fixed steps.
12:    end for
13: end for
```

IV. Performance Evaluation

In this section, we use Python 3.7 to build a CoOR simulation environment, and extensive simulations are conducted to evaluate the performance of proposed G+DQN method. We consider the caching capacity of vehicle is $C_0 = 200$ GB, and the transmission power of vehicle is $p = 30$ dBm. The RSUs are located at the center of coverage area, and the antenna heights are the same, as $l_m = l_{m'} = 30$ m. Other evaluation parameters are listed in Table I. The neural network structures of G+DQN are implemented by PyTorch. We use the following parameters: the learning rate $\xi_\theta = 0.01$, the size of mini-batch is 128, the reward discount $\gamma = 0.5$ and the capacity of replay memory is 6×10^4. Meanwhile, the parameter τ in Algorithm 1 is set as 0.001, then $\zeta = 1$ RMB/s.

For performance comparison, we present five baseline schemes:

TABLE I
Simulation Parameters

Parameter	Value	Definition
ρ_i	Data size of task	$[1-30]M$ bits
η_i	Data size of service	$[20-80]$ GB,
ϕ_i	CPU cycles of task execution	$[50-200]M$ cycles
d_i	Maximal task execution delay	0.5 sec
T_{back}	Result backhaul time	0.005 sec
B_m	Channel bandwidth of V2R	$8M$ Hz
r	Path loss efficient	2.5
σ^2	Background noise of V2R	-100 dBm
$r_{m,m'}$	Data rate of R2R	$800M$ bps
Γ_{cloud}	Cost parameter for downloading	2.5×10^{-3} RMB/GB

- *LocalOnly*: the tasks only be processed locally by the vehicle.
- *RSUOnly*: the tasks only be offloaded to RSUs for processing.
- *NonCo*: the vehicle makes decisions based on Gibbs Sampling and DQN, but no collaboration exists between RSUs.
- *Q-learning Method* [20]: the Q-function in Q-learning directly approximates the optimal function $Q^*(s, a)$, as

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \xi[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]. \qquad (24)$$

  Then, the agent selects the actions using ϵ-greedy.
- *DQN Method*: the task offloading, caching replacement, and collaboration policies are all generated by DQN method, the system state is set as $[\rho_i, \phi_i, l_i]$ and the action is $[\alpha_i, \beta_i, o_i]$.

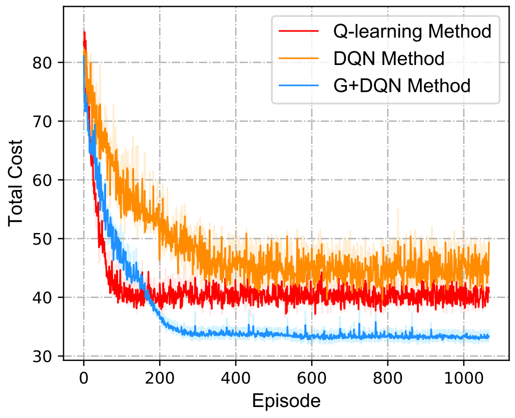

Fig. 2. Convergence of the proposed G+DQN method.

Fig. 2 presents the convergence performance of G+DQN method, compared with the Q-learning and DQN methods. It shows that the total cost in VECNs decreases when the number of episodes increases, until it attains a stable value, which means that the proposed method is convergent. Intuitively, we can observe that the G+DQN achieves a lower final cost, as compared with other two baseline schemes. The reason is that the G+DQN optimizes the task offloading, collaboration and service caching replacement policies separately, while the Q-learning and DQN don't. Furthermore, the larger size of the action space and the complexity of the couple problem, degrade the performance of neural network in DQN method.

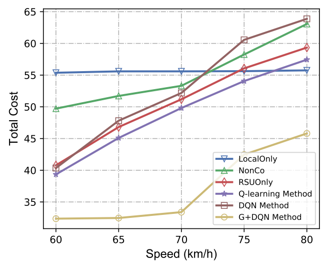

Fig. 3. Performance on total cost under different vehicle speeds.

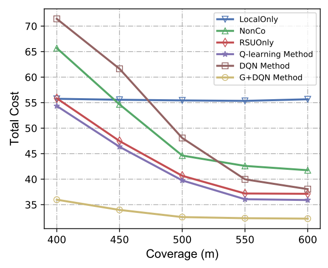

Fig. 4. Performance on total cost under different RSU coverage areas.

Fig. 3 illustrates the total cost comparison under different vehicle speeds. It is observed that the system cost of these five RSU-assisted methods increases as the increasing of vehicle speed, due to the potential handover delay and computing cost when vehicle travels out of the coverage area of RSU. More importantly, the G+DQN achieves a lower overall cost, as compared with all the baseline schemes. Specifically, when the $Speed = 60$ km/h, the total cost incurred by G+DQN is approximately 20% less than that obtained by the best other baseline scheme, which means our proposed method is beneficial.

Fig. 4 shows the effects of different RSU coverage areas. We can find that the cost of five schemes decrease when the coverage area of RSU is wider, as more task offloading and caching replacement can be finished under one RSU coverage area. Compared to other five comparison schemes, our proposed method can achieve the lowest total cost. Further, when $Coverage > 500$ m, the objective value become minimum in this scenario based on G+DQN method, while the Q-learning and RSUOnly schemes reach that until $Coverage > 550$ m.

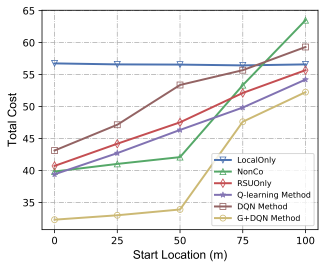

Fig. 5. Performance on total cost under different start locations.

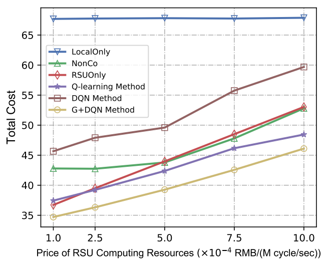

Fig. 6. Performance on total cost under different price of RSU computing resources.

Fig. 5 shows the total cost of task execution with different start locations. Obviously, G+DQN still can outperform the other schemes in terms of the total cost. Moreover, we can find that when $StartLocation > 75$ m, the performances of G+DQN and Q-learning schemes are close to the RSUOnly. The reason is that the initial locations affect the distances between the vehicle and RSU, as well as the handover cost of task execution. When the distance is reducing, the agents tend to offload more tasks to the RSU for obtaining a lower delay.

Finally, Fig. 6 shows the total cost comparison under different prices of RSU computing resources. As expected, the higher price results in more total cost all around, which is easy to explain. Getting benefits from the tradeoff between computing efficiency and cost, the proposal G+DQN performs the best regardless of the price is in the scenario.

V. Conclusion

In this paper, we jointly optimize task offloading and collaboration decisions, as well as the service caching replacement policy in VECNs from the vehicle perspective. The total delay and cost of a sequence of inter-dependent tasks execution are minimized. We propose a CoOR scheme and design a G+DQN method that combines the advantages of Gibbs sampling and DRL to solve the formulated joint optimization problem. Evaluation results show that our proposed scheme can significantly improve the service and computing resource utility, compared to the traditional baselines.

References

[1] H. Guo, J. Liu, J. Ren, and Y. Zhang, "Intelligent task offloading in vehicular edge computing networks," *IEEE Wir. Commun.*, vol. 27, no. 4, pp. 126–132, 2020.

[2] H. Tian, X. Xu, L. Qi, X. Zhang, W. Dou, S. Yu, and Q. Ni, "Copace: Edge computation offloading and caching for self-driving with deep reinforcement learning," *IEEE Trans. on Veh. Technol.*, vol. 70, no. 12, pp. 13 281–13 293, 2021.

[3] Y. Liu, S. Wang, Q. Zhao, S. Du, A. Zhou, X. Ma, and F. Yang, "Dependency-aware task scheduling in vehicular edge computing," *IEEE Internet of Things J.*, vol. 7, no. 6, pp. 4961–4971, 2020.

[4] G. Qiao, S. Leng, S. Maharjan, Y. Zhang, and N. Ansari, "Deep reinforcement learning for cooperative content caching in vehicular edge computing and networks," *IEEE Internet of Things J.*, vol. 7, no. 1, pp. 247–257, 2020.

[5] K. Jiang, H. Zhou, D. Zeng, and J. Wu, "Multi-agent reinforcement learning for cooperative edge caching in internet of vehicles," in *Proc. IEEE 17th MASS*, 2020, pp. 455–463.

[6] K. Zhang, J. Cao, H. Liu, S. Maharjan, and Y. Zhang, "Deep reinforcement learning for social-aware edge computing and caching in urban informatics," *IEEE Trans. on Ind. Inform.*, vol. 16, no. 8, pp. 5467–5477, 2020.

[7] S. Bi, L. Huang, and Y.-J. A. Zhang, "Joint optimization of service caching placement and computation offloading in mobile edge computing systems," *IEEE Trans. on Wir. Commun.*, vol. 19, no. 7, pp. 4947–4963, 2020.

[8] X.-Q. Pham, T.-D. Nguyen, V. Nguyen, and E.-N. Huh, "Joint service caching and task offloading in multi-access edge computing: A QoE-based utility optimization approach," *IEEE Commun. Letters*, vol. 25, no. 3, pp. 965–969, 2021.

[9] X. Ma, A. Zhou, S. Zhang, and S. Wang, "Cooperative service caching and workload scheduling in mobile edge computing," in *Proc. IEEE INFOCOM*, 2020, pp. 2076–2085.

[10] J. Zhao, X. Sun, Q. Li, and X. Ma, "Edge caching and computation management for real-time internet of vehicles: An online and distributed approach," *IEEE Trans. on Intell. Transp. Syst.*, vol. 22, no. 4, pp. 2183–2197, 2021.

[11] Z. Ning, P. Dong, X. Wang, S. Wang, X. Hu, S. Guo, T. Qiu, B. Hu, and R. Y. K. Kwok, "Distributed and dynamic service placement in pervasive edge computing networks," *IEEE Trans. on Parallel and Distrib. Syst.*, vol. 32, no. 6, pp. 1277–1292, 2021.

[12] Q. Yuan, J. Li, H. Zhou, T. Lin, G. Luo, and X. Shen, "A joint service migration and mobility optimization approach for vehicular edge computing," *IEEE Trans. on Veh. Technol.*, vol. 69, no. 8, pp. 9041–9052, 2020.

[13] K. Sehla, T. Nguyen, G. Pujolle, and P. Velloso, "Resource allocation modes in C-V2X: From LTE-V2X to 5G-V2X," *IEEE Internet of Things J.*, vol. 9, no. 11, pp. 8291–8314, 2022.

[14] G. Zhao, H. Xu, Y. Zhao, C. Qiao, and L. Huang, "Offloading tasks with dependency and service caching in mobile edge computing," *IEEE Trans. on Parallel and Distrib. Syst.*, vol. 32, no. 11, pp. 2777–2792, 2021.

[15] J. Yan, S. Bi, Y. J. Zhang, and M. Tao, "Optimal task offloading and resource allocation in mobile-edge computing with inter-user task dependency," *IEEE Trans. on Wir. Commun.*, vol. 19, no. 1, pp. 235–250, 2020.

[16] C. Yang, Y. Liu, X. Chen, W. Zhong, and S. Xie, "Efficient mobility-aware task offloading for vehicular edge computing networks," *IEEE Access*, vol. 7, pp. 26 652–26 664, 2019.

[17] F. Chiariotti, O. Vikhrova, B. Soret, and P. Popovski, "Peak age of information distribution for edge computing with wireless links," *IEEE Trans. on Commun.*, vol. 69, no. 5, pp. 3176–3191, 2021.

[18] M. Li, J. Gao, L. Zhao, and X. Shen, "Deep reinforcement learning for collaborative edge computing in vehicular networks," *IEEE Trans. on Cog. Commun. and Net.*, vol. 6, no. 4, pp. 1122–1135, 2020.

[19] L. P. Qian, Y. J. A. Zhang, and M. Chiang, "Distributed nonconvex power control using gibbs sampling," *IEEE Trans. on Commun.*, vol. 60, no. 12, pp. 3886–3898, 2012.

[20] Y. Li and S. Xu, "Collaborative optimization of edge-cloud computation offloading in internet of vehicles," in *Proc. IEEE ICCCN*, 2021, pp. 1–6.