12-2020

Deep Reinforcement Learning for Collaborative Edge Computing in Vehicular Networks

Mushu Li

Jie Gao

Lian Zhao

Xuemin Shen

IEEE Transactions on Cognitive Communications and Networking, Vol. 6, No. 4 (December 2020): 1122-1135. DOI. This article is © The Institute of Electrical and Electronics Engineers and permission has been granted for this version to appear in e-Publications@Marquette. The Institute of Electrical and Electronics Engineers does not grant permission for this article to be further copied/distributed or hosted elsewhere without the express permission from The Institute of Electrical and Electronics Engineers.

Deep Reinforcement Learning for Collaborative Edge Computing in Vehicular Networks

Mushu Li
University of Waterloo, Waterloo, ON, Canada
Jie Gao
University of Waterloo, Waterloo, ON, Canada
Lian Zhao
Ryerson University, Toronto, ON, Canada
Xuemin Shen
University of Waterloo, Waterloo, ON, Canada

Abstract:

Mobile edge computing (MEC) is a promising technology to support mission-critical vehicular applications, such as intelligent path planning and safety applications. In this paper, a collaborative edge computing framework is developed to reduce the computing service latency and improve service reliability for vehicular networks. First, a task partition and scheduling algorithm (TPSA) is proposed to decide the workload allocation and schedule the execution order of the tasks offloaded to the edge servers given a computation offloading strategy. Second, an artificial intelligence (AI) based collaborative computing approach is developed to determine the task offloading, computing, and result delivery policy for vehicles. Specifically, the offloading and computing problem is formulated as a Markov decision process. A deep reinforcement learning technique, i.e., deep deterministic policy gradient, is adopted to find the optimal solution in a complex urban transportation network. By our approach, the service cost, which includes computing service latency and service failure penalty, can be minimized via the optimal workload assignment and server selection in collaborative computing. Simulation results show that the proposed AI-based collaborative computing approach can adapt to a highly dynamic environment with outstanding performance.

SECTION I. Introduction

Vehicular communication networks have drawn significant attention from both academia and industry in the past decade. Conventional vehicular networks aim to improve the driving experience and enable safety applications via data exchange in vehicle-to-everything (V2X) communications. In the era of 5G, the concept of vehicular networks has been extended to Internet-of-Vehicle (IoV), in which intelligent and interactive applications are enabled by communication and computation technologies [1]. A myriad of on-board applications can be implemented in the context of IoV, such as assisted/autonomous driving and platooning, urban traffic management, and on-board infotainment services [2], [3].

Although IoV technologies are promising, realizing the IoV applications still faces challenges. One of the obstacles is the limited on-board computation capability at vehicles. For example, a self-driving car with ten high-resolution cameras may generate 2 gigapixels per second of data, while 250 trillion computation operations per second are required to process the data promptly [4]. Processing such computation-intensive applications on vehicular terminals is energy-inefficient and time-consuming. To overcome the limitation, mobile edge computing (MEC) is an emerging paradigm that provides fast and energy-efficient computing services for vehicle users [5]–[6][7]. Via vehicle-to-infrastructure (V2I) communications, resource-constrained vehicle users are allowed to offload their computation-intensive tasks to highly capable edge servers co-located with roadside units (RSUs) for processing. Meanwhile, compared to the conventional mobile cloud computing, the network delay caused by task offloading can be significantly reduced in MEC due to the proximity of the edge server to vehicles [8]. Consequently, some applications that require high computing capability, such as path navigation, video stream analytics, and objective detection, can be implemented in vehicular networks with edge servers [9].

Despite the advantage brought by MEC-enabled vehicular networks, new challenges have emerged in task offloading and computing. One critical problem in MEC is to decide which edge servers should their computing tasks be offloaded to. In vehicular networks, the highly dynamic communication topology leads to unreliable communication links [10]. Due to the non-negligible computing time and the limited communication range of vehicles, a vehicle may travel out of the coverage area of an edge server during a service session, resulting in a service disruption. To support reliable computing services for high-mobility users, a service migration scheme has been introduced in [11]. Under this scope, when a user moves out of the communication area of the edge that the computing task was offloaded, the computing process will be interrupted, and the corresponding virtual machine (VM) will be migrated to a new edge according to the radio association. In the urban area, where highly dense infrastructure are deployed, frequent service interruption would happen due to the dynamically changing radio association, which can significantly increase the overall computing service latency.

Alternatively, computing service reliability can be achieved by cooperation among edge servers. Different from service migration, which achieves service reliability by migrating the computing service according to the vehicle's trajectory, computing cooperation improves the service reliability by accelerating task processing time. The computing task can be divided and computed by multiple servers in parallel or fully offloaded to a server with high computing capability at the cost of communication overhead [12], [13]. In this regard, the computing task can be forwarded to the edge server which is out of the user's communication range. Compared to service migration, in which edge servers only execute the task offloaded by the vehicles under their communication coverage, computing cooperation allows edge servers processing the tasks offloaded by the vehicles out of their coverage for reducing the overall computing time. Nevertheless, multi-hop communications could result in significant transmission delay and waste communication spectrum resources in the task offloading process. The tradeoff between the communication overhead and the computing capability increases the complexity of the server assignment problem. In addition, although computing service latency can be reduced by cooperative computing, it is hard to guarantee service reliability for the vehicles with high mobility. The uncertainty of vehicle moving trajectories poses significant challenges in computing result delivery.

Motivated by the issues in the existing service migration and computing cooperation schemes, we present a computing collaboration framework to provide reliable low-latency computing in an MEC-enabled vehicular network. Once an edge server receives the computing tasks offloaded by a vehicle, it may partially or fully distribute the computing workload to another edge server to reduce computing latency. Furthermore, by selecting proper edge servers to deliver the computing results, vehicle users are able to obtain computing results without service disruption caused by mobility. Under this framework, we propose a novel task offloading and computing approach that reduces the overall computing service latency and improves service reliability. To achieve this objective, we firstly formulate a task partition and scheduling optimization problem, which allows all received tasks in the network to be executed with minimized latency given the offloading strategy. A heuristic task partition and scheduling approach is developed to obtain a near-optimal solution of the non-convex integer problem. In addition, we formulate the radio and computing association problem into a Markov decision process (MDP). By characterizing stochastic state transitions in the network, MDP is able to provide proactive offloading policy for vehicles. An artificial intelligence (AI) approach, deep reinforcement learning (DRL), is adopted to cope with the curse of dimensionality in MDP and unknown network state transitions caused by vehicle mobility. Specifically, a convolutional neural network (CNN) based DRL is developed to handle the high-dimensional state space, and the deep deterministic policy gradient (DDPG) algorithm is adopted to handle the high-dimensional action space in the proposed problem. The major contributions of this paper are:

1. We develop an efficient collaborative computing framework for MEC-enabled vehicular networks to provide low-latency and reliable computing services. To overcome the complexity brought by the dynamic network topology, we propose a location-aware task offloading and computing strategy to guide MEC server collaboration.

2. We devise a task partition and scheduling scheme to divide the computing workload among edge servers and coordinate the execution order for tasks offloaded to the servers. Given the offloading strategy, our scheme can minimize the computing time by finding a near-optimal task scheduling solution with low time-complexity.

3. We further propose an AI-based collaborative computing approach, which utilizes a model-free method to find the optimal offloading strategy and MEC server assignment in a 2-dimensional transportation

system. A CNN based DDPG technique is developed to capture the correlation of the state and action among different zones and accelerate the learning speed.

The remainder of the paper is organized as follows. In Section II, we present the related works. Section III describes the system model. Section IV formulates the service delay minimization problem. In Section V, we present the task partition and scheduling scheme, followed by an AI-based collaborative computing approach in Section VI. Section VII presents simulation results, and Section VIII concludes the paper.

SECTION II. Related Works

A. Mobile Edge Computing

As proposed by ETSI in [14], the main objective of MEC is to reduce the computing task offloading and computing latency via utilizing the computing resources located in edge devices, such as base stations and access points. In the context of edge computing, one of the main problems is to determine the computing task offloading mechanisms. The edge server selection problem has been evaluated in [15] and [16]. In [15], Cheng *et al.* propose a user association strategy to jointly minimize the computing delay, user energy consumption, and the server computing cost under a space-air-ground integrated network. A model-free approach is proposed in the work to deal with the complex offloading decision-making problem. In [6], Liu *et al.* investigate the user-server association policy, which takes into account the communication link quality and server computing capability. In both works, the computing association follows the radio association, i.e., the computing task is processed within the edge server that the task is offloaded. To further reduce the computation time, task partition has been considered in [17]–[18][19]. Computing tasks can be split and computed by multiple servers in parallel. The cooperation computing has been investigated among the works [17]–[18][19] under different network environments, while the impact of user mobility has not been addressed. Additionally, in [12], [20], and [21], task scheduling, i.e., ordering the task execution sequences, is also evaluated in the offloading decision making process. In those works, the task scheduling problem is formulated into a mixed-integer programming problem, and heuristic algorithms are proposed to obtain near-optimal solutions efficiently. Different from the above works, we investigate the task partition and scheduling under the collaborative computing framework, in which the adjustment on workload allocation for a task can affect the performance of other tasks, which makes the problem more complex.

B. MEC-enabled Vehicular Networks

The problem of computing offloading has been investigated in many research works in the context of vehicular networks [22]–[23][24][25]. In those works, the main objective is to minimize service time by selecting the optimal edge server, while service reliability in the presence of vehicle mobility is not taken into account. In [26]–[27][28], machine learning techniques are adopted to obtain the reliable offloading decision for vehicles via predict the vehicle trajectories. In [26], Sun *et al.* focus on task offloading and execution utilizing the computing resources on vehicles, i.e., vehicular edge. An online learning algorithm, i.e., multi-armed bandit, is utilized to determine the computing and communication association among vehicles. In [27], Ning *et al.* apply a DRL approach to jointly allocate the communication, caching, and computing resources in the dynamic vehicular network. Furthermore, to deal with service disruption when the vehicle leaving the server converge, service migration has been firstly proposed in [11]. According to the vehicle moving trajectory, the corresponding computing services can be migrated to another edge server that may associate the vehicle in the future. The proactive service migration strategy has been investigated in [29] and [30], where MDP is utilized to make the migration decision in a proactive manner. To alleviate service interruption and network overheads in virtual machine migration, server cooperation has been studied in [5], [31], and [32]. The works [5] and [31] consider that vehicles divide and offload the computing tasks to multiple servers according to the predicted traveling traces. Vehicle-to-vehicle communication is used to disseminate the computing result if the edge server cannot connect with the vehicle at the end of a service session. In [32], the work utilizes neural networks to predict the computing demand in the vehicular network. MEC servers are clustered to compute the offloaded tasks cooperatively. Different from the above works, our proposed approach achieves service reliability improvement by collaboration and task scheduling among edge servers without cooperative transmission, which reduces the communication overhead of result delivery.

SECTION III. System Model

A. Collaborative Edge Computing Framework

An MEC-enabled vehicular network is illustrated in Fig. 1. A row of RSUs, equipped with computing resources, provide seamless communication and computing service coverage for vehicles on the road. An RSU can also communicate with other RSUs within its communication range via wireless links. The set of RSUs is denoted by $\mathcal{R}$, where the index of RSUs is denoted by $r \in \mathcal{R}$. We assume that a global controller has full knowledge of the transportation network and makes offloading and computing decisions for all the vehicles in a centralized manner. In our model, a computing session for a task includes three steps.

1. Offloading: When a computing task is generated at a vehicle, the vehicle selects an RSU, which is under its communication range, and offloads the computing data of the task to the RSU immediately. In the example shown in Fig. 1, RSU r is selected to offload the computing load. Such RSU is referred to as the *receiver RSU* for the task.
2. Computing: After the computing task is fully offloaded, the receiver RSU can process the whole computing task or select another RSU to share the computing load. The RSU, which is selected to process the task collaboratively with the receiver RSU, is referred to as the *helper RSU* for the task.
3. Delivering: A vehicle may travel out of the communication range of its receiver RSU. Therefore, the controller may select an RSU, which could connect with the vehicle at the end of service session, to gather and transmit computing results. The RSU is referred to as the *deliver RSU*. To reduce the overhead, we limit the deliver RSU to be either the receiver RSU or the helper RSU of the task. In the example shown in Fig. 1, RSU $r+\ 1$ behaves as both the helper RSU and the deliver RSU for the computing task offloaded by the vehicle.

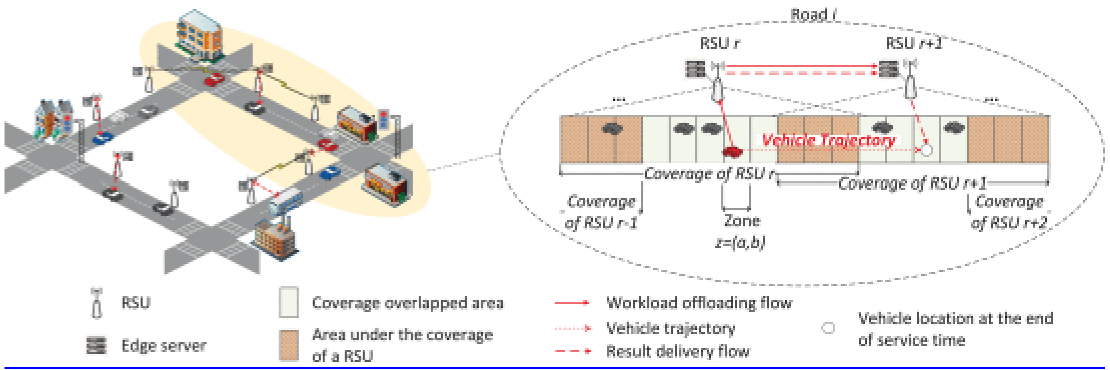


Fig. 1. Network model.

To reduce the decision space in task offloading and scheduling, instead of providing the offloading and computing policy to individual vehicles, we consider location-based offloading and computing policy. We divide each road into several zones with equal length, where the set of zones is denoted by $\mathcal{Z}$. The index of the zones is denoted by $z = (a, b) \in \mathcal{Z}$. The terms a and b represent the index of the roads and the index of the segments on the road, respectively, where $a \in \{1, \dots, A\}$, and $b \in \{1, \dots, B\}$. As the vehicle drives through the road, it traverses the zones consecutively. We assume that all vehicles in the same zone follow the same offloading and computing policy.[1] For simplicity, we evaluate the aggregated tasks for vehicles in each zone at a time slot, and refer to the tasks offloaded by zone z as task z in the remainder of the paper. We suppose that the vehicle will not travel out of a zone during the time duration of a time slot, and vehicles can complete the offloading process of a task generated in a zone before it travels out of the zone. Denote the set of vehicles in zone z and time slot $t \in \mathcal{T}$ as $\mathcal{V}_{z,t}$. The offloading decision for vehicles in zone z and time slot t is represented by a vector $\boldsymbol{\alpha}_{z,t} \in \mathbb{Z}_+^{|\mathcal{R}|}$, where $\sum_{r=1}^{|\mathcal{R}|} \alpha_{z,r,t} = 1$. The element $\alpha_{z,r,t}$ is 1 if RSU r is selected as the receiver RSU for the vehicles in zone z and time slot t, and 0 otherwise. Similarly, the collaborative computing decision for vehicles in zone z and time slot t is represented by a vector $\boldsymbol{\beta}_{z,t} \in \mathbb{Z}_+^{|\mathcal{R}|}$, where $\sum_{r=1}^{|\mathcal{R}|} \beta_{z,r,t} = 1$. The element $\beta_{z,r,t}$ is 1 if RSU r is selected as the helper RSU for the vehicles in zone z and time slot t, and 0 otherwise. In addition, the decision on result delivery is denoted by a binary variable $\gamma_{z,r,t}$, where $\gamma_{z,r,t}$ is 0 if the computing results are delivered by RSU r for task z in time slot t, and $\gamma_{z,r,t}$ is 1 if the computing results are delivered by RSU r.

B. Cost Model

In this paper, the system cost includes two parts: the service delay and the penalty caused by service failure.

1) Service Delay:

We adopt the task partition technique during task processing. Once a receiver RSU receives the offloaded task from vehicles in a zone, it immediately divides the task and offloads a part of the workload to the helper RSU of the corresponding zone. We denote the computing delay of task z corresponding to the receiver or helper RSU r in time slot t as $T^{\mathrm{C}}_{z,r,t}$. As shown in Fig. 2, the computing delay includes task offloading delay, queuing delay, and processing delay. Since the amount of output data is usually much smaller compared to the amount of input data, we neglect the transmission delay in result delivery [19], [24].

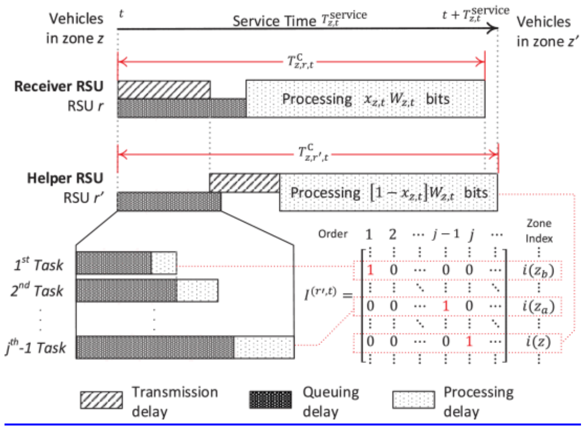


Fig. 2. An example of the task offloading and computing process.

Firstly, task offloading comprises two steps: offloading tasks from vehicles to their receiver RSU and offloading the partial workload from the receiver RSU to the helper RSU. According to the propagation model in 3GPP standards [33], the path loss between a transmitter and a receiver with distance d (km) can be computed as:

$$\begin{aligned} L(d) = \quad & 40(1 - 4 \times 10^{-3} D^{hb}) \log_{10} d - 18 \log_{10} D^{hb} \\ & +21 \log_{10} f + 80(\mathrm{dB}), \end{aligned}$$

(1)

where the parameter f is the carrier frequency in MHz, and the parameter D^{hb} represents the antenna height in meter. We do not consider the shadowing effect of the channel. Denote the distance between the center point of zone z and the location of RSU r as $D_{z,r}$, and the distance between RSU r and r' as $D_{r,r'}$. The data rate for vehicles in zone z offloading task to RSU r is

$$r_{z,r} = B^{\mathrm{Z}} \log_2 \left(1 + \frac{P^{\mathrm{V}} 10^{-\frac{L(D_{z,r})}{10}}}{\sigma_v^2} \right),$$

(2)

where the parameter σ_v^2 denotes the power of the Gaussian noise in the V2I channel, P^{V} represents the vehicle transmit power, and B^{Z} represents the bandwidth reserved for vehicles in a zone. As the receiver RSU for task z, a signal-to-noise ratio threshold should be satisfied, where

$$\frac{P^V 10^{-L(D_{z,r})/10}}{\sigma_v^2} \geq \alpha_{z,r,t} \delta^{\mathrm{O}}, \forall t, z, r,$$

(3)

where δ^{O} is the signal-to-noise ratio threshold for data offloading. Assume that vehicles in a zone are scheduled to offload the tasks successively, and the channel condition is fixed in the duration of any computing task offloading. The transmission delay for offloading the computing data in zone z to the receiver RSU is:

$$T_{z,t}^{\mathrm{T}} = \sum_{r \in \mathcal{R}} \frac{\alpha_{z,r,t} W_{z,t}}{r_{z,r}},$$

(4)

where $W_{z,t}$ represents the overall computing data generated by vehicles in zone z, i.e., task z, and time slot t. In addition, the data rate between RSU r and RSU r' for forwarding the computing data offloaded from a zone is

$$r_{r,r'} = B^{\mathrm{R}} \log_2 \left(1 + \frac{P^{\mathrm{R}} 10^{-\frac{L(D_{r,r'})}{10}}}{\sigma_r^2} \right),$$

(5)

where the parameter σ_r^2 represents the power of the Gaussian noise in the RSU to RSU channel, P^{R} represents the RSU transmit power, and B^{R} represents the bandwidth reserved for forwarding data offloaded from a zone. In data forwarding, the signal-to-noise constraint is also required to be satisfied, where

$$\frac{P^R 10^{-L(D_{r,r'})/10}}{\sigma_r^2} \geq \beta_{z,r',t} \delta^{\mathrm{O}}, \forall t, z, r, r'.$$

(6)

For computing task z in time slot t, the portion of workload to be processed by the receiver RSU and the helper RSU is denoted by $x_{z,t}$ and $1 - x_{z,t}$, respectively. Thus, the delay for forwarding the data to the deliver RSU is:

$$T_{z,t}^{\mathrm{R}} = \sum_{r \in \mathcal{R}} \sum_{r' \in \mathcal{R}} \frac{\alpha_{z,r,t} \beta_{z,r',t} (1 - x_{z,t}) W_{z,t}}{r_{r,r'}}.$$

(7)

Furthermore, after the task is offloaded to edge servers, the queuing delay may be experienced. Let set $\mathcal{Z}^{r,t}$ denote the zones which have tasks offloaded to RSU r, i.e., $\{z | \alpha_{z,r,t} = 1\} \cup \{z | \beta_{z,r,t} = 1\}$, and let $i\ (z\ )$ represent the index of zone z in set $\mathcal{Z}^{r,t}$. We denote $N_{r,t}$ as the number of tasks offloaded in time slot t and assigned to the RSU r, where $N_{r,t} = \sum_z \alpha_{z,r,t} + \beta_{z,r,t}$. Then, a matrix, $\mathbb{I}^{(r,t)} \in \mathbb{Z}_+^{N_{r,t} \times N_{r,t}}$, can be defined to imply the processing order of tasks offloaded to RSU r in time slot t, where $I_{i(z),j}^{(r,t)} = 1$ if the task offloaded from zone z is scheduled as the j -th task to be processed among the other tasks offloaded in the same time slot. As shown in Fig. 2, the queuing delay of a task depends on the computing time of the task scheduled priorly. For the first task to be processed among the tasks offloaded in time slot t, the queuing delay stems from the computing time for the tasks offloaded in previous time slots. Thus, the queuing delay of task z in RSU r can be formulated as follows:

$$T_{z,r,t}^{\mathrm{Q}} = \begin{cases} T_{r,t}^{\mathrm{Q0}}, & \mathrm{if} I_{i(z),1}^{(r,t)} = 1, \\ \sum_{z'} \sum_{j} I_{i(z),j}^{(r,t)} I_{i(z'),j-1}^{(r,t)} T_{z',r,t}^{\mathrm{C}}, & \mathrm{otherwise.} \end{cases}$$

(8)

The term $T_{r,t}^{\mathrm{Q0}}$ represents the latency for finishing the tasks offloaded in previous time slots $\{1, \ldots, t-1\}$, where

$$T_{r,t}^{\mathrm{Q0}} = \max\{\sum_{z'} I_{i(z'),N_{r,t-1}}^{(r,t)} T_{z',r,t-1}^{\mathrm{C}} - \epsilon, 0\},$$

(9)

where ϵ is the length of a time slot.

We consider that data transmission and task processing run in parallel. After the task is offloaded and other tasks scheduled priorly are completed, the task can be processed by the dedicated server. The delay for processing task z offloaded to RSU r in time slot t can be formulated as

$$T_{z,r,t}^{\mathrm{P}} = \frac{\chi W_{z,t} \left[\alpha_{z,r,t} x_{z,t} + \beta_{z,r,t} \left(1 - x_{z,t} \right) \right]}{C_r},$$

(10)

where C_r denotes the computing capability (CPU-cycle frequency) of RSU r, and χ denotes the number of computation cycles needed to execute 1 bit of data.

Given the offloading delay, queuing delay, and processing delay, the computing delay for task z on RSU r can be formulated as follows:

$$T_{z,r,t}^{\mathrm{C}}=\max\{T_{z,t}^{\mathrm{T}}+\beta_{z,r,t}T_{z,t}^{\mathrm{R}},T_{z,r,t}^{\mathrm{Q}}\}+T_{z,r,t}^{\mathrm{P}}.$$

(11)

Denote the overall service delay for the task offloaded from zone z in time slot t as $T_{z,t}^{\mathrm{service}}$. As shown in Fig. 2, the overall service delay depends on the longest computing time between the receiver RSU and the helper RSU. Thus, we have

$$T_{z,t}^{\mathrm{service}}=\max\left\{\sum_{r}\alpha_{z,r,t}T_{z,r,t}^{\mathrm{C}},\sum_{r}\beta_{z,r,t}T_{z,r,t}^{\mathrm{C}}\right\}.$$

(12)

2) Service Failure Penalty:

The mobility of vehicles brings uncertainty in result downloading. Service failure may occur if a vehicle is out of the coverage of its deliver RSU during the service session. Denote the zone that vehicle v is located when its computing result is delivered as m_v, i.e., the location of vehicle $v\in\mathcal{V}_{z,t}$ in time slot $t+T_{z,t}^{\mathrm{Service}}$. Also, we denote the signal-to-noise ratio threshold for result delivering as δ^{D}. We introduce a variable $\mathbf{1}_{z,t}$ to indicate whether the computing service for task z offloaded in time slot t is successful or not, where

$$\mathbf{1}_{z,t}=\begin{cases}1, & \mathrm{if} P^{R}10^{-\frac{L(D_{m_v,r})}{10}}\geq\sigma_r^2\gamma_{z,r,t}\delta^{\mathrm{D}},\forall v\in\mathcal{V}_{z,t}\\ 0, & \mathrm{otherwise.}\end{cases}$$

(13)

SECTION IV. Problem Formulation

Our objective is to minimize the weighted sum of the overall computing service delay for vehicle users and service failure penalty. The corresponding objective function can be formulated as follows:

$$\begin{aligned}\min_{\substack{\{\boldsymbol{\alpha},\boldsymbol{\beta},\boldsymbol{\gamma},\mathbf{x},\\ \{\mathbf{I}^{(r,t)},\forall r,t\}\}}}\quad & \lim_{T\to\infty}\frac{1}{T}\sum_{t=0}^{T-1}\sum_{z\in\mathcal{Z}}\{T_{z,t}^{\mathrm{service}}\mathbf{1}_{z,t}+\lambda W_{z,t}(1-\mathbf{1}_{z,t})\}\\ \mathrm{s.t.}\quad & (3),(6),\\ & \sum_{r\in\mathcal{R}}\alpha_{z,r,t}=1,\sum_{r\in\mathcal{R}}\beta_{z,r,t}=1,\sum_{r\in\mathcal{R}}\gamma_{z,r,t}=1\\ & \sum_{i=1}^{N_{r,t}}I_{i,j}^{(r,t)}=1,\sum_{j=1}^{N_{r,t}}I_{i,j}^{(r,t)}=1\\ & 0\leq x_{z,t}\leq 1,\\ & \boldsymbol{\alpha}_{z,t},\boldsymbol{\beta}_{z,t}\in\mathbb{Z}_{+}^{|\mathcal{R}|},\\ & \boldsymbol{I}^{(r,t)}\in\mathbb{Z}_{+}^{N_{r,t}\times N_{r,t}},\end{aligned}$$

(14a)(14b)(14c)(14d)(14e)(14f)(14g)

where λ represents per-unit penalty, for the case when the computing offloading service fails. The optimization variables include three aspects: edge server selection, i.e., $\{\boldsymbol{\alpha},\boldsymbol{\beta},\boldsymbol{\gamma}\}$, task partition, i.e., $\mathbf{x}$, and task scheduling, i.e., $\left\{\mathbf{I}^{(r,t)},\forall r,t\right\}$. It can be seen that Problem (14) is a mixed-integer nonlinear optimization problem. Solving the above problem directly by conventional optimization methods is challenging. Furthermore, the decision dimension of the problem is too large to apply model-free techniques directly. Taking the variable of task execution order as an example, i.e., $\mathbf{I}^{(r,t)}$, there are $N_{r,t}\times N_{r,t}$ number of decisions to be determined for a server in a time slot. The number of combinations of scheduling decisions is at least $(|\mathcal{Z}|/|\mathcal{R}|)!\times|\mathcal{R}|\times|\mathcal{T}|$, in which tasks are evenly assigned to servers and each task is processed by only one server. Thus, to reduce the decision dimension of the problem, we divide Problem (14) into two sub-problems: i) task partition and scheduling problem, and ii) edge server selection problem. In the task partition and scheduling problem, we aim to obtain the optimal task partition ratio and the execution order to minimize the computing latency given the offloading policy $\{\boldsymbol{\alpha},\boldsymbol{\beta}\}$. After that, we re-formulate the edge server selection problem as an MDP and utilize the DRL technique to obtain the optimal offloading and computing policy.

SECTION V. Task Partition and Scheduling

Multiple tasks offloaded from different zones can be received by an edge server in a time slot. The computing tasks can only be processed if the tasks scheduled priorly are executed. As a result, the overall computing time may vary depending on the task execution order in edge servers. In addition, the workload of a task can be divided and offloaded to two edge servers, i.e., receiver and helper RSUs. Workload allocation for a task also affects the overall service time. Therefore, we study task partition and scheduling to minimize the service latency given the offloading policy $\{\boldsymbol{\alpha},\boldsymbol{\beta}\}$. Based on Problem (14), the delay minimization problem can be formulated as follows:

$$\begin{aligned}\min_{\mathbf{x},\{\mathbf{I}^{(r,t)},\forall r,t\}}\quad & \sum_{z\in\mathcal{Z}} T_{z,t}^{\text{service}}\\ \text{s.t.}\quad & (14\text{d}),(14\text{e}),(14\text{g}).\end{aligned}$$

(16a)(16b)

Problem (16) is a mixed-integer programming, which involves a continuous variable x and an integer matrix variable $\left\{\mathbf{I}^{(r,t)},\forall r,t\right\}$. Moreover, even if x is known, the remaining integer problem is a variation of the traveling salesman problem, which is an NP-hard problem. To reduce the time-complexity in problem-solving, we exploit the properties of task partition and scheduling and develop a heuristic algorithm to obtain an approximate result efficiently. To simplify the notations, we eliminate the time index t in the remainder of the section since we consider the scheduling scheme for the tasks offloaded in one time slot. We further denote $r(z)$ and $h(z)$ as the index of receiver and helper RSUs for task z, respectively.

Lemma 1:

If no task is queued after task z for both the receiver RSU and the helper RSU, the optimal partition ratio for the task x_z^* is $\min\left\{\max\left\{0,\hat{x}_z\right\},1\right\}$, where $\hat{x}_z$ can be determined by Eq. (15), as shown at the bottom of the page.

Proof:

Without considering the tasks queued later, the service time of task z can be minimized by solving the following problem:

$$\min \max\left\{T^{\mathrm{C}}_{z,r(z)}, T^{\mathrm{C}}_{z,h(z)}\right\} \mathrm{s.t.}\ (14\mathrm{e}).$$

(17)

Given that $0 < x_z < 1$, the optimal task partition strategy exists when $T^{\mathrm{C}}_{z,r(z)} = T^{\mathrm{C}}_{z,h(z)}$. The optimal task partition ratio is $x^*_z = \hat{x}_z$. In addition, $x^*_z = \max\{0, \hat{x}_z\} = 0$ when the helper RSU can fully process task z in a shorter service time comparing to the queuing time in the receiver RSU, i.e., $\max\left\{T^{\mathrm{Q}}_{z,r(z)}, T^{\mathrm{T}}_z\right\} \geq \max\left\{T^{\mathrm{Q}}_{z,h(z)}, T^{\mathrm{T}}_z + \frac{\chi W_z}{R_{r(z),h(z)}}\right\} + \frac{\chi W_z}{C_{h(z)}}$. Otherwise, $x^*_z = \min\{1, \hat{x}_z\} = 1$, when the receiver RSU can process task z by itself in a shorter service time comparing to the queuing time in the helper RSU, i.e., $\max\left\{T^{\mathrm{Q}}_{z,r(z)}, T^{\mathrm{T}}_z\right\} \leq T^{\mathrm{Q}}_{z,h(z)} - \frac{\chi W_z}{C_{r(z)}}$.

Lemma 1 shows the optimal partition ratio from the individual task perspective. However, multiple tasks could be offloaded from different zones to an RSU, where the role of the RSU could be different for those tasks. The task partition strategy for a single task could affect the computing latency for the task queued later. Therefore, we will investigate the optimality of the task partition scheme in Lemma 1 in terms of minimizing the overall service time for all tasks $z \in \mathcal{Z}$.

Lemma 2:

Assume that the following conditions are met:

- The computing capability C_r is identical for all edge servers.
- The receiver RSU and helper RSU are different for each task, i.e., $r(z) \neq h(z)$.
- For the helper RSUs for all tasks, the queuing time is not shorter than the offloading time, i.e., $T^{Q}_{z,h(z)} \geq T^{T}_{z,r(z)} + T^{R}_{r(z),h(z)}, \forall z, r$.

Then, given the execution order of tasks, the optimal solution of Problem (16) follows the results shown in Lemma 1, i.e., $x^*_z = \min\{\max\{0, \hat{x}_z\}, 1\}, \forall z$.

Proof:

See Appendix A.

We have proved that, given the task execution order, the partition ratio in Lemma 1 is the optimal solution for Problem (16) under certain assumptions. Next, we will explore the optimal scheduling order given the workload allocation policy.

Lemma 3:

Consider only one available RSU in the system, i.e., $r(z) = h(z)$. Under the assumption in which the offloading time is proportional to the size of the task, the optimal task execution order is to schedule the task with the shortest service time first.

Proof:

See Appendix B.

According to the properties provided in Lemmas 1–3, we design a heuristic algorithm to schedule the task execution order and allocate workload among RSUs. The full algorithm is presented in Algorithm 1. In the algorithm, we allocate the task that has the shortest service time first. For each task, we divide the workload between the receiver RSU and helper RSU according to the optimal partition ratio in Lemma 1. In the worst case, in which all zones have tasks to offload in a time slot, the algorithm requires $|\mathcal{Z}|(|\mathcal{Z}|+1)/2$ iterations to compute the task partition and scheduling results, which can still provide fast responses in the dynamic environment.

SECTION Algorithm 1 Task Partition and Scheduling Algorithm (TPSA)

```
1.  At time slot $t$, initialize set $\mathcal{S} = \{z|W_{z,t} \neq 0\}$.
2.  Initialize $\psi_r = T^{\mathrm{Q0}}_{r,t}$, $\mathbf{I}^{(r,t)} = 0$, and $j_r = 1, \forall r$.
3.  **while** $|\mathcal{S}| \neq 0$ **do**
4.    Initialize $Q_z = 0, \forall z \in \mathcal{S}$.
5.    **for** Task $z = 1{:}\,|\mathcal{S}|$ **do**
6.        Update $r(z) = \{r|\alpha_{z,r,t} = 1\}$ and $= \{r|\beta_{z,r,t} = 1\}$.
7.        Update partition ratio $x_z = \min\{\max\{0, \hat{x}_z\},1\}$,where $\hat{x}$ is obtained by (15).
8.        Update $\hat{\psi}_{z,r(z)} = \psi_{r(z)} + T^{\mathrm{C}}_{z,r(z)}$.
9.        Update $\hat{\psi}_{z,h(z)} = \psi_{h(z)} + T^{\mathrm{C}}_{z,h(z)}$.
10.       If $x_z = 1$, then $Q_z = \hat{\psi}_{z,h(z)}$.
11.       If $x_z = 0$, then $Q_z = \hat{\psi}_{z,r(z)}$.
12.       If $0 < x_z < 1$, then $Q_z = (\hat{\psi}_{z,r(z)} + \hat{\psi}_{z,h(z)})/2$.
13.   **end for**
14.   Find $z^* = \mathrm{argmin}_z Q_z$.
15.   Update $\psi_{r(z^*)} = \hat{\psi}_{z^*,r(z^*)}$ and $\psi_{h(z^*)} = \hat{\psi}_{z^*,h(z^*)}$.
16.   Update order matrix $I^{r(z^*),t}_{z^*,j_{r(z^*)}} = 1$, and $I^{h(z^*),t}_{z^*,j_{h(z^*)}} = 1$.
17.   Update $j_{r(z^*)} = j_{r(z^*)} + 1$, and $j_{h(z^*)} = j_{h(z^*)} + 1$.
18.   $\mathcal{S} = \mathcal{S} \setminus \{z^*\}$.
19. **end while**
20. $T^{\mathrm{Q0}}_{r,t+1} = \psi_r - \epsilon, \forall r$.
```

SECTION VI. AI-Based Collaborative Computing Approach

To deal with the server selection problem, we utilize a DRL technique to conduct the complex decision-making problem in a dynamic environment. To implement the DRL method, we first re-formulate the problem into an MDP. An MDP can be defined by a tuple $(\mathbb{S}, \mathbb{A}, \mathbb{T}, \mathbb{C})$, where $\mathbb{S}$ represents the set of system states; $\mathbb{A}$ represents the set of actions; $\mathbb{T} = \{p(s_{t+1}|s_t, a_t)\}$ is the set of transition probabilities; and $\mathbb{C}$ is the set of real-value cost functions. The term $C\,(s, a)$ represents the cost when the system is at state $s \in \mathbb{S}$ and an action $a \in \mathbb{A}$ is taken. A policy π represents a mapping from $\mathbb{S}$ to $\mathbb{A}$. In our problem, the state space, action space, and cost model in an MDP are summarized as follows:

1. *State space:* In time slot t, the network state, s_t, includes the computing data amount in zones, i.e., $\{W_{z,t}, \forall z\}$, the average vehicle speed, i.e., $\{v_{z,t}, \forall z\}$, and the delay for edge servers to finish the tasks offloaded in previous time slots $\{1, \ldots, t-1\}$, i.e., $\{T_{r,t}^{\mathrm{Q0}}, \forall r\}$.
2. *Action space:* For zone z and time slot t, the action taken by the network includes three elements: the index of receiver RSU, helper RSU, and deliver RSU, which can be represented by $\{a_{z,t}^1, a_{z,t}^2, a_{z,t}^3\}$, respectively.
3. *Cost model:* Given the state-action pair, the overall service time can be available by the TPSA algorithm. Thus, according to the objective function (14), the cost function can be formulated as

$$C(s_t, a_t) = \sum_{z \in \mathcal{Z}} \left\{ T_{z,t}^{\mathrm{service}} \mathbf{1}_{z,t} + \lambda W_{z,t} \left(1 - \mathbf{1}_{z,t}\right) \right\}.$$

(18)

Then, to obtain the expected long-term discounted cost, the value function V of state s is

$$V(s, \pi) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t C(s_t, a_t) | s_0 = s, \pi\right],$$

(19)

where the parameter γ is a discount factor. By minimizing the value function of each state, we can obtain the optimal offloading and computing policy π^*; that is,

$$\pi^*(s) = \mathrm{argmin}_a \sum_{s'} p(s'|s, a) \left[C(s, a) + \gamma V(s', \pi^*)\right].$$

(20)

Due to the limited knowledge on transition probability between the states and the sizeable state-action space in the network, the traditional dynamic programming is not able to find the optimal policy efficiently. Therefore, we adopt DRL to solve the proposed server selection problem. There are three common DRL algorithms: deep Q network (DQN), actor-critic (AC), and DDPG. DQN is a powerful tool to obtain the optimal policy with a high dimension in the state space. Besides an online neural network (evaluation network) to learn the Q value, a frozen network (target network) and the experience replay technique are applied to stabilize the learning process. However, the method shows the inefficiency on the network with a high dimension in the action space, while in our problem, the large number of zones leads the high dimension in both state and action spaces. On the other hand, both AC and DDPG tackle the problem with a high action dimension by the policy gradient technique. Two networks, i.e., actor and critic networks, are adopted, in which the critic evaluates the Q value, and the actor updates policy parameters in the direction suggested by the critic. Moreover, DDPG combines the characteristics of DQN on top of the AC algorithm to learning the Q value and the deterministic policy by the experience relay and the frozen network, thereby helping reach the fast convergence [34]. In this paper, we exploit the DDPG algorithm to obtain the optimal collaborative computing policy in vehicular networks.

The illustration of our AI-based collaborative computing approach is shown in Fig. 3. The system states are observed from the MEC-enabled vehicular network. After state s_t is obtained, the optimal server selection policy can be computed by the DDPG algorithm. According to the server selection results, the corresponding task partition and scheduling policy can be obtained by the proposed TPSA algorithm. Then, the cost of the corresponding state-action pair and the next system state can be observed from the environment. The state transition set (s_t, a_t, r_t, s_{t+1}) is stored in the replay memory for training the neural networks. In DDPG, four neural networks are employed. Two of the four networks are evaluation networks, where the weights are updated when the neural network is trained, and the other two networks are target networks, where the weights are replaced periodically from the evaluation network. For both evaluation and target networks, two neural networks, i.e., actor and critic networks, are adopted to evaluate the optimal policy and Q value, respectively. The weights in evaluation and target critic networks are denoted by θ^Q and $\theta^{Q'}$, and the weights in evaluation and target actor networks are denoted by θ^μ and $\theta^{\mu'}$, respectively.

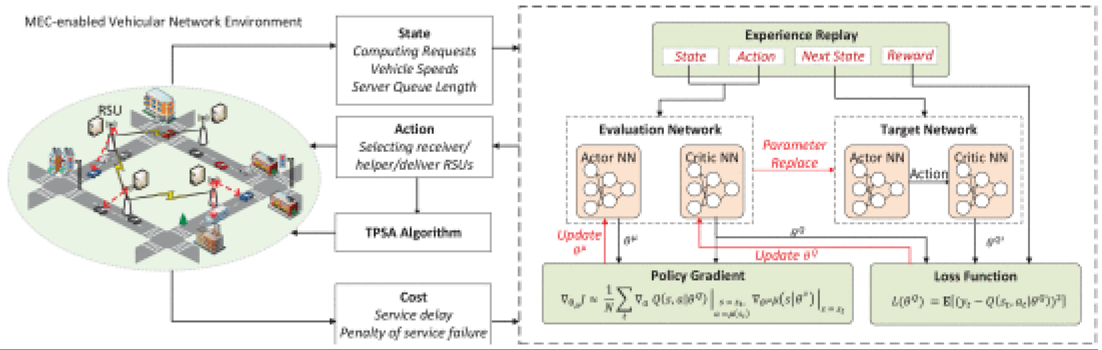


Fig. 3. AI-based collaborative computing approach.

In each training step, a batch of experience tuples are extracted from the experience replay memory, where the number of tuples in a mini-batch is denoted by N. The critics in both evaluation and target networks approximate the value function and compute the loss function L, where

$$L(\theta^Q) = \mathbf{E}\left[\left(y_t - Q(s_t, a_t|\theta^Q)\right)^2\right].$$

(21)

The term $Q(s_t, a_t|\theta^Q)$ represents the Q function approximated by the evaluation network. The value of y_t is obtained from the value function approximated by the target network, where

$$y_t = C(s_t, a_t) + \gamma Q\left(s_{t+1}, \mu'\left(s_{t+1}\middle|\theta^{\mu'}\right)\middle|\theta^{Q'}\right).$$

(22)

The term $\mu'(s_{t+1}|\theta^{\mu'})$ represents the action taken at s_{t+1} given by the target actor network. By minimizing the loss function (21), the weights in the evaluation critic, i.e., θ^Q, can be updated. On the other hand, to update the weights of the evaluation actor network, the policy gradient can be represented as

$$\nabla_{\theta_\mu} J \approx \frac{1}{N}\sum_t \nabla_a Q(s, a|\theta^Q)|_{\substack{s=s_t,\\ a=\mu(s_t)}} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s=s_t}.$$

(23)

From (23), it can be seen that actor weights are updated in each training step according to the direction suggested by the critic.

Although the DDPG algorithm is able to tackle the problem with a high dimension of state and action spaces, it is inefficient to apply the DDPG algorithm directly in our problem due to the 2-dimensional transportation network and the multiple dimensions of the input. A huge number of neurons in a network will be deployed if the conventional neural network with fully connected layers is adopted. To improve the algorithm efficiency, we utilize CNN in both actor and critic networks to exploit the correlation of states and actions among different zones. The structure of actor and critic networks is shown in Fig. 4. Before fully connected layers, convolution layers and pooling layers are applied to learn the relevant features of the inputs among zones. Due to the weight sharing feature of CNN filters, the number of training parameters can be significantly reduced compared to the network with fully connected layers [35]. After several convolution and pooling layers, the output of the CNN combines the state of edge servers and forwards to fully connected layers.

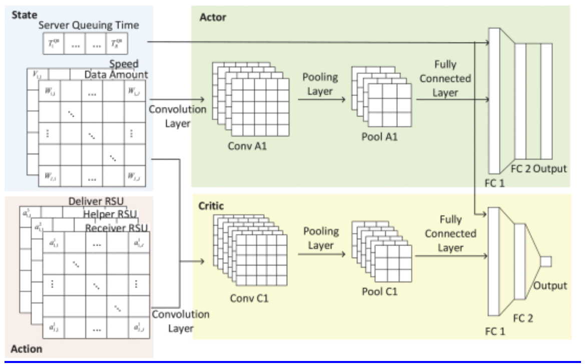


Fig. 4. The structure of actor and critic neural networks.

The proposed AI-based collaborative computing approach is provided in Algorithm 2, where τ is a small number less than 1. In our algorithm, to learn the environment efficiently, the system will continuously train the parameter by N_t times after N_e time step, where $N_e > N_t$.

SECTION Algorithm 2 AI-Based Collaborative Computing Approach

```
1.  Initialize critic network $Q(s_0, a_0|\theta^Q)$ and actor network $\mu(s_0|\theta^\mu)$ with weights $\theta^Q$ and $\theta^\mu$.
2.  Initialize target network with weights $\theta^{Q'} = \theta^Q$ and $\theta^{\mu'} = \theta^\mu$.
3.  Initialize the experience replay buffer.
4.  Initialize a random vector $\mathcal{N}$ as the noise for action exploration.
5.  **for** episode = 1: G **do**
6.      Initialize environment, and observe the initial state $s_0$.
7.      **for** time slot $t = 1{:}T$ **do**
8.          Select action $a_t = \mu(s|\theta^\mu) + \mathcal{N}$.
9.          Let $\alpha_{z,a^1_{z,t},t}$, $\beta_{z,a^2_{z,t},t}$, and $\gamma_{z,a^3_{z,t},t}$ equal to 1.
10.         Compute the task partition and scheduling results by Algorithm 1.
11.         Observe next state $s_{t+1}$ and cost $C(s_t, a_t)$.
12.         Store transition $(s_t, a_t, r_t, s_{t+1})$ into the experience replay buffer. Delete the oldest
            transition set
            if the buffer is full.
13.         **if** $k \bmod N_e == 0$ **then**
14.             **for** $j = 1{:}N_t$ **do**
15.                 Sample a mini-batch of $N$ samples.
16.                 Update $y_t$ by (22).
17.                 Update the weights in the evaluation criticnetwork by minimizing the loss in (21).
```

```
18.             Update the weights in the evaluation actornetwork using sampled policy
                gradientpresented in (23).
19.             Update target networks: $\theta^{Q'} = \tau\theta^{Q} + (1-\tau)\theta^{Q'}$; $\theta^{\mu'} = \tau\theta^{\mu} + (1-\tau)\theta^{\mu'}$.
20.         end for
21.       end if
22.   end for
23. end for
```

SECTION VII. Performance Evaluation

In this section, we first present the efficiency of the proposed TPSA algorithm in task partition and scheduling. Then, we evaluate the performance of the proposed AI-based collaborative computing approach in a vehicular network simulated by VISSIM [36], where TPSA is applied to schedule computing tasks according to the policy given by the DDPG algorithm.

A. Task Partition and Scheduling Algorithm

We first evaluate the performance of the proposed TPSA algorithm. In the simulation, we consider that tasks can be offloaded to five edge servers with an identical offloading rate of 6 Mbits/s. The communication rate among the servers is 8 Mbits/s. We set that the computing capability of the servers is 8 GC/s, and the number of computation cycles needed for processing 1 Mbit is 4 GC. The computing data amount of tasks is uniformly distributed in the range of [1, 21] Mbits. For each task, the receiver and helper RSUs are randomly selected from the five servers. We compare the proposed TPSA algorithm with *brute-force* and *random* schemes. In the brute-force scheme, we utilize an exhaustive search for finding the optimal scheduling order. In the random scheme, we randomly assign the scheduling order of the tasks. Note that, for both *brute-force* and *random* schemes, we adopt the optimal task partition ratio in workload allocation. The simulation results presented in Figs. 5(a) and 5(b) are averaged over 200 rounds of Monte Carlo simulations.

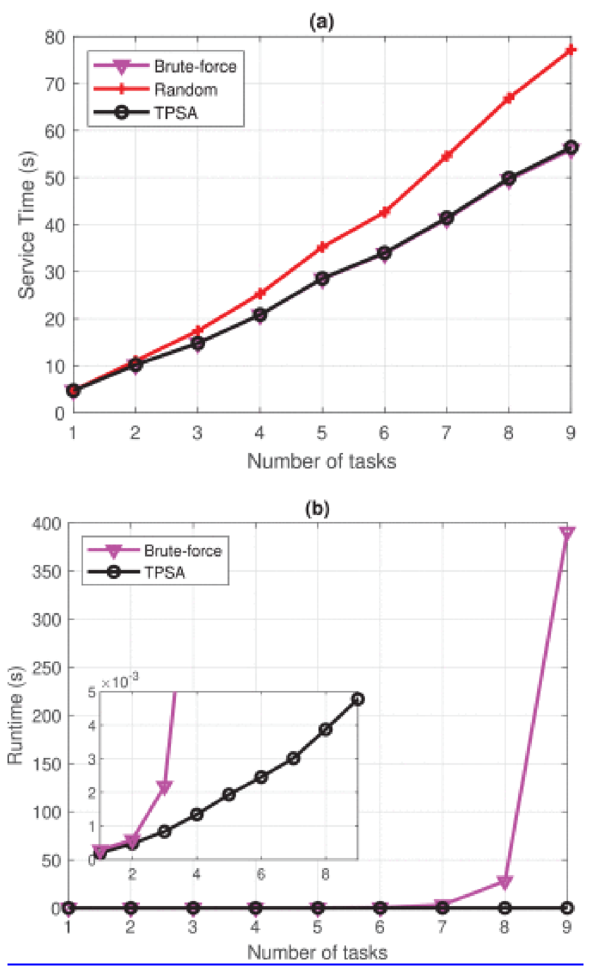


Fig. 5. (a) Average service delay among the three task partition and scheduling schemes with respect to the number of tasks. (b) Average computation runtime among the three task partition and scheduling schemes with respect to the number of tasks.

The service delay performance of the proposed algorithm is shown in Fig. 5(a). It can be seen that an increase in the task number leads to increasing overall service time, and the increasing rate of the random scheme is the highest among the three schemes. The proposed TPSA algorithm can achieve a performance very close to the brute-force scheme. Moreover, we compare the runtime between the proposed TPSA and the brute-force scheme. As shown in Fig. 5(b), as the number of the task increases, the runtime of brute-force scheme increases exponentially, while the proposed TPSA algorithm has imperceptible runtime to compute the scheduling result that is close to the optimal one. In summary, the proposed TPSA algorithm can achieve a near-optimal performance for task partition and scheduling with low computation complexity.

B. AI-Based Collaborative Computing Approach

In this subsection, we evaluate the performance of the proposed AI-based collaborative computing approach. In the simulation, we consider an 800 m ×800 m transportation system, where the transportation topology is shown in Fig. 6. Nine RSUs with edge servers are deployed, as indicated in the figure. We generate vehicle traffic by VISSIM [36], where 200 vehicles are traveling in the area. The speed of vehicles depends on the speed limit on the road and the distance to the vehicle ahead. For each vehicle, the computing tasks are generated using a Poisson process, and the input data amount of each task is uniformly distributed in the range of [2, 5] Mbits. The length and width of a zone are 40 m and 10 m (2 driving lanes), respectively. Other network parameter settings are presented in Table I. We test the system performance within a duration of 20 seconds.

TABLE I Network Parameters

P^V	P^R	σ_r^2, σ_v^2	λ	ϵ
27 dBm	37 dBm	-93 dBm	50	1 s
f	χ	N_e, N_t	δ^o	δ^D
2800 MHz	1200 C/bits	80, 25	7 dB	7 dB

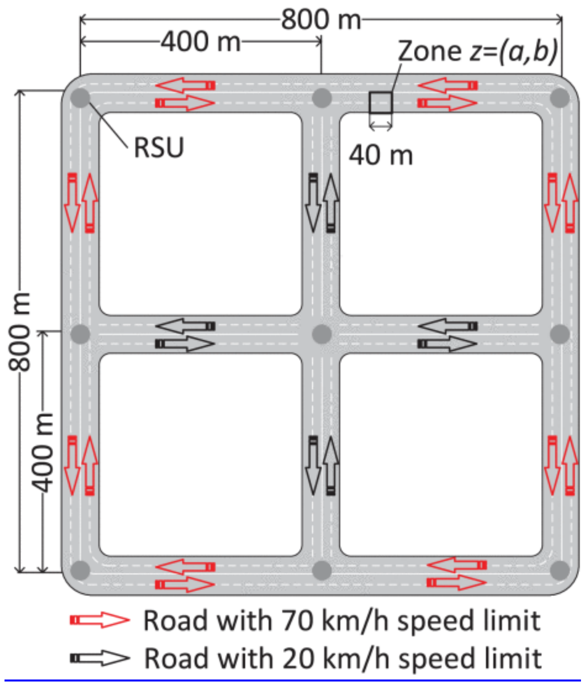


Fig. 6. The transportation network topology for simulation.

The neural network structures of the DDPG algorithm are presented in Table II. The initial learning rates of the actor and critic networks are 1e-5 and 1e-4, respectively, and the learning rates are attenuated by 0.991 in every 500 training steps. The experience replay buffer can adopt 8,000 state-action transitions, and in each training step, the number of transition tuples selected for training, i.e., the batch size, is 128. We adopt a soft parameter replacement technique to update the parameters in the target network, where τ is 0.01. We compare the performance of the proposed AI-based collaborative computing approach with three approaches. In the *Greedy* approach, vehicles always offload their tasks to the RSU with the highest SNR, and the received computing tasks will not be collaboratively computed with other RSUs. In the *Greedy* + *TPSA* approach, a vehicle offload their tasks to the RSU with the highest SNR, and the RSU randomly selects another RSU to compute the task collaboratively. The task partition and scheduling policy follows the TPSA algorithm, and the computing results are delivered by the receiver RSU. In the *Random+TPSA* approach, the receiver, helper, and deliver RSUs are selected randomly, and the TPSA algorithm is applied to determine the task partition ratio and the execution order.

TABLE II Neural Network Structure

Actor Network		
Layer	Number of neurons	Activation function
CONV1	$5 \times 1 \times 2 \times 10$, stride 1	relu
POOL1	2×1	none

Data Concatenation and Batch Normalization Layer		
FC1	1400	tanh
FC2	1400	tanh
FC3	$5 \times A \times B$	tanh
Critic Network		
Layer	Number of neurons	Activation function
CONV1	$5 \times 1 \times 2 \times 40$, stride l	relu
POOL1	2×1	none
CONV2	$3 \times 1 \times 40 \times 10$, stride 1	relu
POOL2	2×1	none
Data Concatenation and Batch Normalization Layer		
FC1	640	relu
FC2	512	relu
FC3	128	none
FC4	1	relu

The overall weighted computing cost with respect to task arrival rates is shown in Fig. 7. Our proposed approach can achieve the lowest computing cost compared to the other three approaches. The random approach suffers the highest cost compared to others due to the inefficient server selection in the scheme. Moreover, the greedy+TPSA approach achieves a lower cost compared to the greedy approach. The reason is that parallel computing is able to reduce the overall service time, and the proposed TPSA is able to achieve near-optimal task partition and scheduling results. However, the greedy approach selects the servers according to the instantaneous cost of the network rather than the value in the long term. Therefore, the greedy + TPSA approach cannot attain a lower cost compared to the proposed AI-based approach.

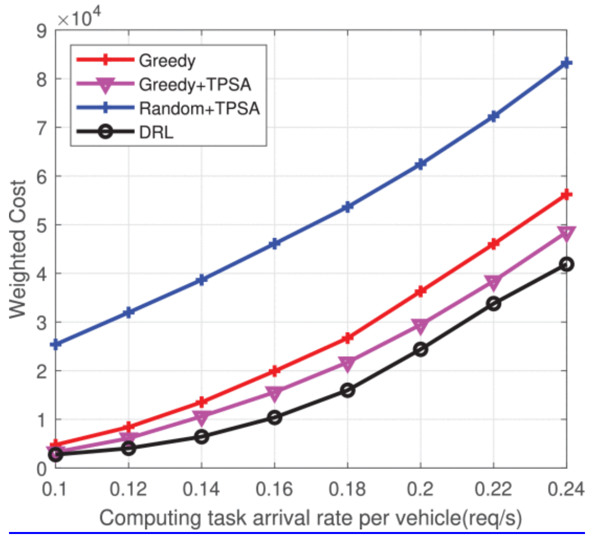


Fig. 7. Average weighted computing delay cost versus computing task arrive rate per vehicle.

As indicated in Eq. (18), the service cost consists of the service delay and the failure penalty. The results of the service failure percentage is shown in Fig. 8. Similar to the service cost, the proposed AI-based approach achieves the lowest failure percentage among the four approaches. Correspondingly, as shown in Fig. 9, the proposed approach can successfully process the highest amount of data among the four approaches. On the other hand, the results of the average service delay for 1 Mbits successful computed data are shown in Fig. 10. Compared to the other three approaches, the proposed scheme reduces the service delay significantly.

Furthermore, the delay of the random approach increases exponentially since less amount of data can be successfully computed when the task arrival rate is high.

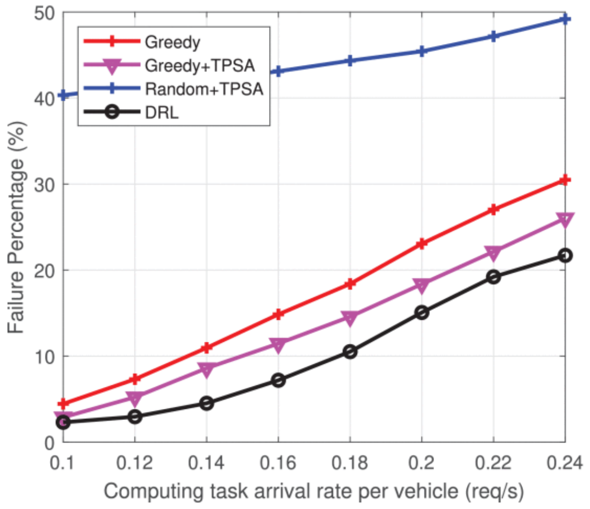


Fig. 8. Average percentage of service failure versus computing task arrive rate per vehicle.

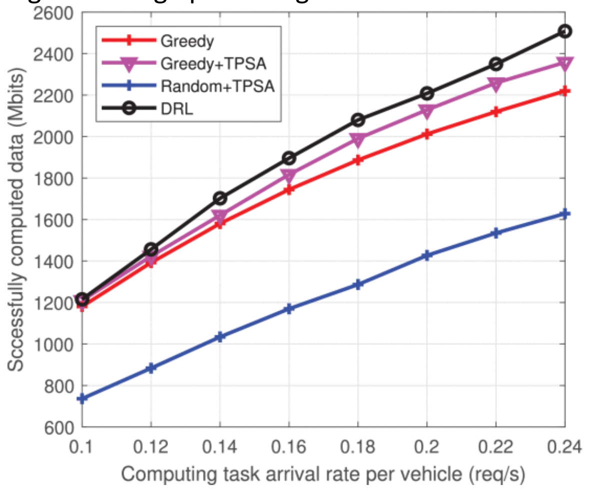


Fig. 9. Average computing data amount which is successfully computed versus computing task arrive rate per vehicle.

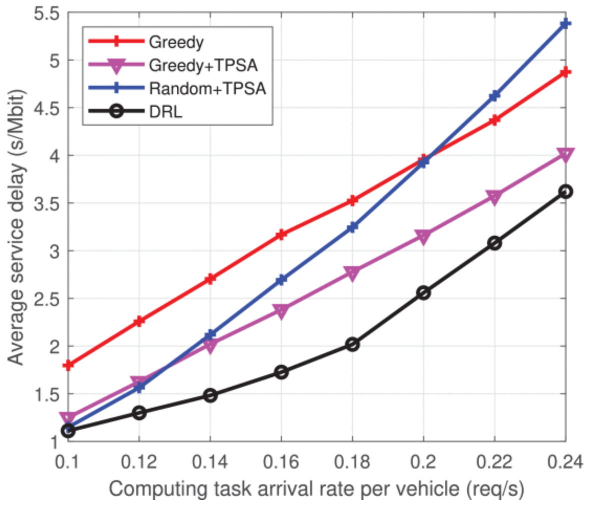


Fig. 10. Average service delay for 1 Mbits successful computed data versus computing task arrive rate per vehicle.

The convergence performance of the proposed AI-based approach is shown in Fig. 11, where the highlighted line represents the moving average from 50 samples around the corresponding point. Note that in our algorithm, we explore multiple times in each training step. It can be seen that our approach converges after 10,000 episodes, or equivalently, after the network being trained by around 3,000 episodes, i.e., 60,000 training steps.

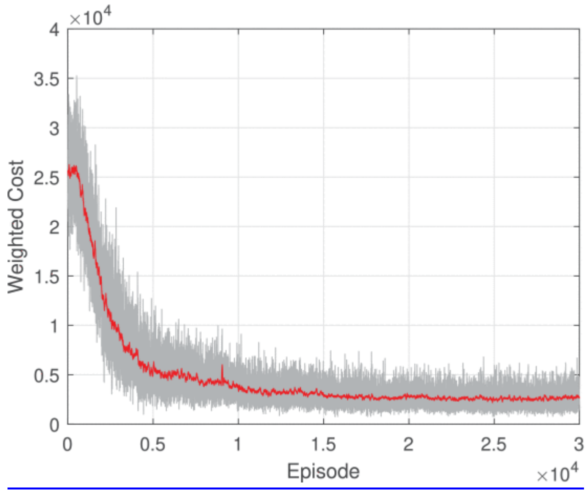


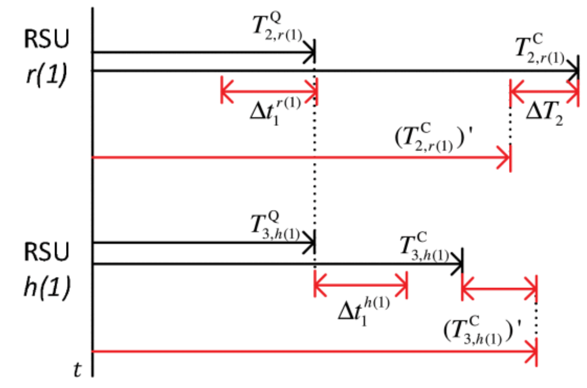
Fig. 11. Convergence performance of the proposed algorithm, where the task arrival rate is 0.1 request/sec.

SECTION VIII. Conclusion

We have introduced a novel collaboration computing framework to reduce computing service latency and improve service reliability in MEC-enabled vehicular networks. The proposed framework addresses the challenge of maintaining computing service continuity for vehicle users with high mobility. As a result, our collaborative computing approach is able to support proactive decision making for computation offloading through learning the network dynamics. Our work can be applied to offer low-latency and high-reliable edge computing services to vehicle users in a complex network environment, such as urban transportation systems. In the future, we will investigate multi-agent learning approach to compute the optimal computing strategy with the limited information collected by the edge servers.

Appendix A Proof of Lemma 2

An illustration of task partition is shown in Fig. 12. Consider that the workload of all tasks are divided and shared among RSUs following the results in Lemma 1. We focus on a single task which is numbered as task 1 as shown in the figure. As indicated in the second and the third assumptions in Lemma 2, the computing load of task 1 is shared between RSUs r (1) and h (1). Tasks 2 and 3 are scheduled after task 1 in RSUs r (1) and h (1), respectively. In addition, $T^{Q}_{2,h(2)} \geq T^{T}_{2,r(2)} + T^{R}_{r(2),h(2)}$. We then prove that, under the assumption in Lemma 2, the overall service time will be increased if the partition ratio of task 1 does not follow the policy presented in Lemma 1.



RSU
r(1)
$T^{\mathrm{Q}}_{2,r(1)}$
$T^{\mathrm{C}}_{2,r(1)}$
$\Delta t_1^{r(1)}$
ΔT_2
$(T^{\mathrm{C}}_{2,r(1)})'$
RSU
h(1)
$T^{\mathrm{Q}}_{3,h(1)}$
$T^{\mathrm{C}}_{3,h(1)}$
$\Delta t_1^{h(1)}$
$(T^{\mathrm{C}}_{3,h(1)})'$
t

Fig. 12. An illustration of task partition.

Consider that, for task 1, the workload assigned to RSU r (1) is decreased by Δx. Correspondingly, the computing time of task 1 in server r (1) is reduced by $\Delta t_1^{r(1)} = \Delta x / C_{r(1)}$, while the computing time of task 1 in server h (1) is increased by $\Delta t_1^{h(1)} = \Delta x / C_{h(1)}$. Thus, the service time of task 1 is increased by $\Delta T_1 = \Delta x / C_{h(1)}$. Denote the new partition ratio of task 2, after task partition ratio x_1 is decreased by Δx, as $\hat{x}_2$. We then list following cases to analyze the time deduction from the tasks queued after the task 1:

- *Case 1:* Task 2 regards RSU r (1) as the receiver RSU, i.e., r (1) =r (2), and $\hat{x}_2 < 1$. According to Eq. (12) and Lemma 1, the optimal service time of task 2 is

$$T_2^{\text{service}} = \max\left\{T_2^{\mathrm{T}}, T_{2,r(1)}^{\mathrm{Q}}\right\} + \frac{\left(T_{2,h(2)}^{\mathrm{Q}} - \max\left\{T_2^{\mathrm{T}}, T_{2,r(1)}^{\mathrm{Q}}\right\}\right) C_{h(2)} + \chi W_2}{C_{r(1)} + C_{h(2)}}.$$

(24)

After task partition ratio x_1 is decreased by Δx, task 2 can be processed by RSU r (1) in advance by $\Delta t_1^{r(1)}$. The new optimal service time of task 2 is

$$(T_2^{\text{service}})' = \max\left\{T_2^{\mathrm{T}}, T_{2,r(1)}^{\mathrm{Q}} - \Delta t_1^{r(1)}\right\} + \frac{\left(T_{2,h(2)}^{\mathrm{Q}} - \max\left\{T_2^{\mathrm{T}}, T_{2,r(1)}^{\mathrm{Q}} - \Delta t_1^{r(1)}\right\}\right) C_{h(2)} + \chi W_2}{C_{r(1)} + C_{h(2)}}.$$

(25)

The service time deduction on task 2 can be obtained by subtracting Eq. (24) by Eq. (25). We found the reduced service time $\Delta T_2 \leq \Delta t_1^{r(1)} C_{r(1)} / (C_{r(1)} + C_{h(2)})$, where equality can be reached when $T_2^{\mathrm{T}} \leq T_{2,r(1)}^{\mathrm{Q}} - \Delta t_1^{r(1)}$.

- *Case 2:* Task 2 regards RSU r (1) as the receiver RSU, i.e., r (1) =r (2), and $\hat{x}_2 = 1$. In this case, the new optimal service time of task 2 is

$$(T_2^{\text{service}})' = \max\left\{T_2^{\text{T}}, T_{2,r(1)}^{\text{Q}} - \Delta t_1^{r(1)}\right\} + \frac{\chi W_2}{C_{r(1)}}.$$

(26)

Via subtracting Eq. (24) by Eq. (26), we have

$$\begin{aligned}\Delta T_2 \leq &\quad \Delta t_1^{r(1)} - \frac{\left(\chi W_z/C_{r(1)} - T_{2,h(2)}^{\text{Q}} + T_{2,r(1)}^{\text{Q}}\right)C_{h(2)}}{C_{r(1)} + C_{h(2)}} \\ \leq &\quad \frac{\Delta t_1^{r(1)} C_{r(1)}}{\left(C_{r(1)} + C_{h(2)}\right)},\end{aligned}$$

(27)

where equality can be achieved when $T_2^{\text{T}} \leq T_{2,r(1)}^{\text{Q}} - \Delta t_1^{r(1)}$.

- *Case 3:* Task 2 regards RSU r (1) as the helper RSU, i.e., r (1) =h (2). In this case, the new optimal service time of task 2 is

$$\begin{aligned}(T_2^{\text{service}})' &= \max\left\{T_2^{\text{T}}, T_{2,r(2)}^{\text{Q}} - \Delta t_1^{r(2)}\right\}\} \\ &+ \frac{\left(T_{2,r(1)}^{\text{Q}} - \Delta t_1^{r(2)} - \max\left\{T_2^{\text{T}}, T_{2,r(2)}^{\text{Q}}\right\}\right) C_{r(1)} + \chi W_2}{C_{r(2)} + C_{r(1)}}.\end{aligned}$$

(28)

Similar as case 1, the reduced service time for task 2 is $\Delta T_2 = \Delta t_1^{r(1)} C_{r(1)}/(C_{r(1)} + C_{r(2)})$.

Considering that the computing capabilities Cr are identical for all servers (the first assumption in Lemma 2), the maximum service time deduction for task 2 is $\Delta t_1^{r(1)}/2$. For all tasks queued after task 1 in RSU r (1), the overall service time deduction is less than $\Delta t_1^{r(1)}[1/2 + (1/2)^2 + (1/2)^3 + \cdots]$, which is always less than $\Delta t_1^{r(1)}$. We omit the proof for the case when the workload assigned in RSU h (1) is decreased by Δx due to the similarity. Therefore, we obtain that, under the assumptions presented in Lemma 2, the overall service time will be increased if the workload allocation does not follow the task partition ratio presented in Lemma 1.

Appendix B Proof of Lemma 3

Suppose the tasks in edge server r are scheduled by the shortest-task-first rule, and task 2 is queued after the task 1. Then, we have

$$\max\left\{T_{1,r}^{\text{Q}}, T_{1,r}^{\text{T}}\right\} + T_{1,r}^{\text{P}} \leq \max\left\{T_{1,r}^{\text{Q}}, T_{2,r}^{\text{T}}\right\} + T_{2,r}^{\text{P}}.$$

(29)

If the order of task 1 and task 2 are switched with each other, the service time of task 2 will be decreased by

$$D = \max\left\{\max\left\{T_{1,r}^{\text{Q}}, T_{1,r}^{\text{T}}\right\} + T_{1,r}^{\text{P}}, T_{2,r}^{\text{T}}\right\} - \max\left\{T_{1,r}^{\text{Q}}, T_{2,r}^{\text{T}}\right\}.$$

(30)

On the other hand, the service time of task 1 will be increased by

$$I = \max\left\{T_{1,r}^{\mathrm{Q}}, T_{2,r}^{\mathrm{T}}\right\} + T_{2,r}^{\mathrm{P}} - \max\left\{T_{1,r}^{\mathrm{Q}}, T_{1,r}^{\mathrm{T}}\right\}.$$

(31)

From (29), we can derive that $I \geq T_{1,r}^{\mathrm{P}}$. Then, the overall service time of tasks 1 and 2 will be increased by

$$\begin{aligned} I - D \geq \quad & T_{1,r}^{\mathrm{P}} - \max\left\{\max\left\{T_{1,r}^{\mathrm{Q}}, T_{1,r}^{\mathrm{T}}\right\} + T_{1,r}^{\mathrm{P}}, T_{2,r}^{\mathrm{T}}\right\} \\ & + \max\left\{T_{1,r}^{\mathrm{Q}}, T_{2,r}^{\mathrm{T}}\right\}. \end{aligned}$$

(32)

We then list the three scenarios on $T_{2,r}^{\mathrm{T}}$:

- *Case 1:* $T_{2,r}^{\mathrm{T}} \geq \max\{T_{1,r}^{\mathrm{Q}}, T_{1,r}^{\mathrm{T}}\} + T_{1,r}^{\mathrm{P}}$. In this case, $I - D \geq T_{1,r}^{\mathrm{P}} \geq 0$.
- *Case 2:* $T_{1,r}^{\mathrm{Q}} \leq T_{2,r}^{\mathrm{T}} \leq \max\left\{T_{1,r}^{\mathrm{Q}}, T_{1,r}^{\mathrm{T}}\right\} + T_{1,r}^{\mathrm{P}}$. In this case, $I - D \geq T_{2,r}^{\mathrm{T}} - \max\left\{T_{1,r}^{\mathrm{Q}}, T_{1,r}^{\mathrm{T}}\right\}$. According the assumption, where $T_{1,r}^{\mathrm{T}} \leq T_{2,r}^{\mathrm{T}}$, we then have $I - D \geq 0$.
- *Case 3:* $T_{2,r}^{\mathrm{T}} \leq T_{1,r}^{\mathrm{Q}}$. In this case, $I - D \geq T_{1,r}^{\mathrm{Q}} - \max\{T_{1,r}^{\mathrm{Q}}, T_{1,r}^{\mathrm{T}}\} = 0$.

Therefore, we can obtain the conclusion that the service time will be increased if the task execution order does not follow a shortest-task-first rule under the assumptions.

References

1. N. Cheng, N. Zhang, N. Lu, X. Shen, J. W. Mark and F. Liu, "Opportunistic spectrum access for CR-VANETs: A game-theoretic approach", *IEEE Trans. Veh. Technol.*, vol. 63, no. 1, pp. 237-251, Jan. 2014.

2. H. Peng and X. Shen, "Deep reinforcement learning based resource management for multi-access edge computing in vehicular networks", *IEEE Trans. Netw. Sci. Eng.*, Mar. 2020.

3. J. Gao, M. Li, L. Zhao and X. Shen, "Contention intensity based distributed coordination for V2V safety message broadcast", *IEEE Trans. Veh. Technol.*, vol. 67, no. 12, pp. 12288-12301, Dec. 2018.

4. *Self Driving Safety Report*, Jun. 2018, [online] Available: https://www.nvidia.com/content/dam/en-zz/Solutions/self-driving-cars/safety-report/auto-print-safety-report-final-web.pdf.

5. K. Zhang, Y. Zhu, S. Leng, Y. He, S. Maharjan and Y. Zhang, "Deep learning empowered task offloading for mobile edge computing in urban informatics", *IEEE Internet Things J.*, vol. 6, no. 5, pp. 7635-7647, Oct. 2019.

6. J. Liu, J. Wan, B. Zeng, Q. Wang, H. Song and M. Qiu, "A scalable and quick-response software defined vehicular network assisted by mobile edge computing", *IEEE Commun. Mag.*, vol. 55, no. 7, pp. 94-100, Jul. 2017.

7. N. Zhang, S. Zhang, P. Yang, O. Alhussein, W. Zhuang and X. Shen, "Software defined space-air-ground integrated vehicular networks: Challenges and solutions", *IEEE Commun. Mag.*, vol. 55, no. 7, pp. 101-109, Jul. 2017.

8. J. Xu, L. Chen and S. Ren, "Online learning for offloading and autoscaling in energy harvesting mobile edge computing", *IEEE Trans. Cogn. Commun. Netw.*, vol. 3, no. 3, pp. 361-373, Sep. 2017.

9. J. Zhang and K. B. Letaief, "Mobile edge intelligence and computing for the Internet of Vehicles", *Proc. IEEE*, vol. 108, no. 2, pp. 246-261, Feb. 2020.

10. F. Lyu et al., "MoMAC: Mobility-aware and collision-avoidance MAC for safety applications in VANETs", *IEEE Trans. Veh. Technol.*, vol. 67, no. 11, pp. 10590-10602, Nov. 2018.

11. T. Taleb, A. Ksentini and P. A. Frangoudis, "Follow-me cloud: When cloud services follow mobile users", *IEEE Trans. Cloud Comput.*, vol. 7, no. 2, pp. 369-382, Apr. 2019.

12. J. Cao, L. Yang and J. Cao, "Revisiting computation partitioning in future 5G-based edge computing environments", *IEEE Internet Things J.*, vol. 6, no. 2, pp. 2427-2438, Apr. 2019.

13. L. Lin, X. Liao, H. Jin and P. Li, "Computation offloading toward edge computing", *Proc. IEEE*, vol. 107, no. 8, pp. 1584-1607, Aug. 2019.

14. Y. C. Hu, M. Patel, D. Sabella, N. Sprecher and V. Young, Mobile edge computing—A key technology towards 5G, Sophia Antipolis, France, 2014.

15. N. Cheng et al., "Space/aerial-assisted computing offloading for IoT applications: A learning-based approach", *IEEE J. Sel. Areas Commun.*, vol. 37, no. 5, pp. 1117-1129, May 2019.

16. C. Liu, M. Bennis, M. Debbah and H. V. Poor, "Dynamic task offloading and resource allocation for ultra-reliable low-latency edge computing", *IEEE Trans. Commun.*, vol. 67, no. 6, pp. 4132-4150, Jun. 2019.

17. L. Chen, S. Zhou and J. Xu, "Computation peer offloading for energy-constrained mobile edge computing in small-cell networks", *IEEE/ACM Trans. Netw.*, vol. 26, no. 4, pp. 1619-1632, Aug. 2018.

18. C. You and K. Huang, "Exploiting non-causal CPU-state information for energy-efficient mobile cooperative computing", *IEEE Trans. Wireless Commun.*, vol. 17, no. 6, pp. 4104-4117, Jun. 2018.

19. M. Li, N. Cheng, J. Gao, Y. Wang, L. Zhao and X. Shen, "Energy-efficient UAV-assisted mobile edge computing: Resource allocation and trajectory optimization", *IEEE Trans. Veh. Technol.*, vol. 69, no. 3, pp. 3424-3438, Mar. 2020.

20. H. A. Alameddine, S. Sharafeddine, S. Sebbah, S. Ayoubi and C. Assi, "Dynamic task offloading and scheduling for low-latency IoT services in multi-access edge computing", *IEEE J. Sel. Areas Commun.*, vol. 37, no. 3, pp. 668-682, Mar. 2019.

21. J. Feng, Z. Liu, C. Wu and Y. Ji, "AVE: Autonomous vehicular edge computing framework with ACO-based scheduling", *IEEE Trans. Veh. Technol.*, vol. 66, no. 12, pp. 10660-10675, Dec. 2017.

22. Y. He, N. Zhao and H. Yin, "Integrated networking caching and computing for connected vehicles: A deep reinforcement learning approach", *IEEE Trans. Veh. Technol.*, vol. 67, no. 1, pp. 44-55, Jan. 2018.

23. Q. Qi et al., "Knowledge-driven service offloading decision for vehicular edge computing: A deep reinforcement learning approach", *IEEE Trans. Veh. Technol.*, vol. 68, no. 5, pp. 4192-4203, May 2019.

24. X. Wang, Z. Ning and L. Wang, "Offloading in Internet of Vehicles: A fog-enabled real-time traffic management system", *IEEE Trans. Ind. Informat.*, vol. 14, no. 10, pp. 4568-4578, Oct. 2018.

25. M. Li, L. Zhao and H. Liang, "An SMDP-based prioritized channel allocation scheme in cognitive enabled vehicular ad hoc networks", *IEEE Trans. Veh. Technol.*, vol. 66, no. 9, pp. 7925-7933, Sep. 2017.

26. Y. Sun et al., "Adaptive learning-based task offloading for vehicular edge computing systems", *IEEE Trans. Veh. Technol.*, vol. 68, no. 4, pp. 3061-3074, Apr. 2019.

27. Z. Ning et al., "Joint computing and caching in 5G-envisioned Internet of Vehicles: A deep reinforcement learning-based traffic control system", *IEEE Trans. Intell. Transp. Syst.*, Feb. 2020.

28. K. A. Hafeez, L. Zhao, Z. Liao and B. N. Ma, "A fuzzy-logic-based cluster head selection algorithm in VANETs", *Proc. IEEE Int. Conf. Commun. (ICC)*, pp. 203-207, Jun. 2012.

29. S. Wang, R. Urgaonkar, M. Zafer, T. He, K. Chan and K. K. Leung, "Dynamic service migration in mobile edge computing based on Markov decision process", *IEEE/ACM Trans. Netw.*, vol. 27, no. 3, pp. 1272-1288, Jun. 2019.

30. H. Liang, X. Zhang, J. Zhang, Q. Li, S. Zhou and L. Zhao, "A novel adaptive resource allocation model based on SMDP and reinforcement learning algorithm in vehicular cloud system", *IEEE Trans. Veh. Technol.*, vol. 68, no. 10, pp. 10018-10029, Oct. 2019.

31. J. Zhou, D. Tian, Y. Wang, Z. Sheng, X. Duan and V. C. M. Leung, "Reliability-optimal cooperative communication and computing in connected vehicle systems", *IEEE Trans. Mobile Comput.*, vol. 19, no. 5, pp. 1216-1232, May 2020.

32. Z. Xiao et al., "Vehicular task offloading via heat-aware MEC cooperation using game-theoretic method", *IEEE Internet Things J.*, vol. 7, no. 3, pp. 2038-2052, Mar. 2020.

33. J. Chen, W. Xu, N. Cheng, H. Wu, S. Zhang and X. Shen, "Reinforcement learning policy for adaptive edge caching in heterogeneous vehicular network", *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, pp. 1-6, Dec. 2018.

34. T. P. Lillicrap et al., "Continuous control with deep reinforcement learning" in, 2015, [online] Available: http://arXiv:1509.02971.

35. A. Krizhevsky, I. Sutskever and G. E. Hinton, "ImageNet classification with deep convolutional neural networks" in Advances in Neural Information Processing Systems 25, Red Hook, NY, USA:Curran Assoc., Inc, pp. 1097-1105, 2012.

36. *VISSIM*, Jun. 2020, [online] Available: http://vision-traffic.ptvgroup.com/.

Footnotes

1. The accuracy of vehicle locations will be improved when the length of the zone is reduced. In consideration of the length of a car, the length of a zone is larger than 5 m.