MESON: A Mobility-aware Dependent Task Offloading Scheme for Urban Vehicular Edge Computing

Liang Zhao, *Member, IEEE,* Enchao Zhang, Shaohua Wan, Ammar Hawbani, Ahmed Y. Al-Dubai, *Senior Member, IEEE,* Geyong Min, and Albert Y. Zomaya, *Fellow, IEEE*

Abstract—Vehicular Edge Computing (VEC) is the transportation version of Mobile Edge Computing (MEC) in road scenarios. One key technology of VEC is task offloading, which allows vehicles to send their computation tasks to the surrounding Roadside Units (RSUs) or other vehicles for execution, thereby reducing computation delay and energy consumption. However, the existing task offloading schemes still have various gaps and face challenges that should be addressed because vehicles with time-varying trajectories need to process massive data with high complexity and diversity. In this paper, a VEC-based computation offloading model is developed with consideration of data dependency of tasks. The minimization of the average response time and average energy consumption of the system is defined as a combinatorial optimization problem. To solve this problem, we propose a Mobility-aware dependent task offloading (MESON) Scheme for urban VEC and develop a DRL-based algorithm to train the offloading strategy. To improve the training efficiency, a vehicle mobility detection algorithm is further designed to detect the communication time between vehicles and RSUs. In this way, MESON can avoid unreasonable decisions by lowering the size of the action space. Moreover, to improve the system stability and the offloading successful rate, we design a task priority determination scheme to prioritize the tasks in the waiting queue. The experimental results show that MESON is superior compared to other task offloading schemes in terms of the average response time, average system energy consumption, and offloading successful rate.

Index Terms—Mobile Edge Computing, Vehicular Edge Computing, Vehicular Networks, Deep Reinforcement Learning, Task Offloading.

1 Introduction

In recent years, Vehicular Edge Computing (VEC) has been considered as a promising computing paradigm that migrates Mobile Edge Computing (MEC) to the road and vehicular scenarios by empowering various computation services for in-vehicle applications [1]. The ubiquitous edge servers deployed on Roadside Units (RSUs) can provide high-Quality of Service (QoS) computing resources for vehicles. In VEC, task offloading allows vehicles to transfer their computation tasks to surrounding RSUs or vehicles for execution. Thus, the execution delay and energy consumption can be reduced [2]. In this way, VEC can improve driving safety and transportation efficiency to better support autonomous driving. Since the offloading decision should be made for each task, the efficiency of the offloading scheme may cause a critical impact on the offloading decision-making. Nowadays, task offloading has attracted extensive attention from both academia and industry. Some existing studies apply heuristic algorithms for task offloading [3–6]. However, they rely heavily on expert knowledge or precise mathematical analytical models. Due to the dynamic nature of VEC, these solutions require constant updating of the existing models, which is time-consuming.

Other existing studies involve Machine Learning (ML) to operate the task offloading, whereas ML demands prior knowledge [16, 17]. Nevertheless, benefiting from the development of vehicular technologies, the data generated by vehicles is with variability and diversity. It is difficult to obtain large amounts of data to train ML models. Other studies believe that Reinforcement Learning (RL) is a potential solution to task offloading [18–20]. RL-based algorithms enable vehicles to learn an optimal offloading strategy by interacting with the environment. Nonetheless, such solutions are only feasible for low-dimensional input decision-making problems, rather than dealing with large-scale task offloading in urban scenarios with high-density vehicles.

To better understand the limitations of the existing studies, we extensively investigate relevant task offloading schemes. Table 1 shows the objectives, mobility of vehicles, application properties, and main algorithm for each scheme. However, the existing task offloading schemes still have various gaps and face challenges that need to be addressed.

- *Liang Zhao and Enchao Zhang are with the School of Computer Science, Shenyang Aerospace University, Shenyang, China. Liang Zhao is also with the Shenzhen Institute for Advanced Study, University of Electronic Science and Technology of China, Shenzhen 518110, China. (e-mail: lzhao@sau.edu.cn, enchaozhang1998@163.com)*
- *Shaohua Wan is with the Shenzhen Institute for Advanced Study, University of Electronic Science and Technology of China, Shenzhen 518110, China. (e-mail: shaohua.wan@ieee.org)*
- *Ammar Hawbani is with University of Science and Technology of China, China. (e-mail: anmande@ustc.edu.cn).*
- *Geyong Min is with the Department of Computer Science, University of Exeter, UK. (e-mail: g.min@exeter.ac.uk).*
- *Ahmed Y.Al-Dubai is with the School of Computing, Edinburgh Napier University, UK. (e-mail: a.al-dubai@napier.ac.uk).*
- *Albert Y. Zomaya is with the School of Computer Science, University of Sydney, Australia (e-mail: albert.zomaya@sydney.edu.au).*
- *Ammar Hawbani and Enchao Zhang are the corresponding authors.*

TABLE 1
A comparison of related studies.

Reference	Objectives			Mobility	Application Properties			Scheme
	Latency	Joint	Energy		Dependency	Task Number	Priority	
[7]	✓					Single		GT
[8]			✓			Multiple	✓	EECO
[9]	✓					Multiple		MDRCO
[10]		✓				Multiple		DCOR
[11]	✓			✓	✓	Multiple		TESO
[12]	✓					Single		DQN
[13]		✓				Multiple		SARSA
[14]		✓		✓		Multiple		PGA
[15]	✓					Multiple		CCBL
Our Work		✓		✓	✓	Multiple	✓	MESON

First, some studies only consider independent tasks instead of the tasks with data dependency of applications [7–10]. As shown in the table, only one study considered offloading scenarios with task dependency. Although the dependencies of tasks are studied, the characteristics of dependent tasks are not fully considered [11, 21, 22]. As presented in [23], an application is composed of a set of collaborative code units called modules. In our paper, the task is another expression of the module. Generally, a programming module can be executed on various CPUs in a parallel system by applying interfaces. However, tasks in the application are usually data-dependent. When processing tasks with data dependencies, the order of execution of tasks is critical, which has a significant impact on system efficiency. Consequently, the data dependency and the task execution order are also important characteristics of task offloading, which need to be highlighted. The offloading system should consider how to prioritize tasks to maximize the offloading execution efficiency and maintain the stability of the offloading system.

Second, as shown in the table, some existing studies ignored the impact of the time-varying trajectories of vehicles in urban scenarios during the process of task offloading [12, 13]. Wireless communication between vehicles and RSUs plays an important role in task offloading. Considering the dynamic nature of vehicles, broken communication links between vehicles and RSUs caused by vehicles driving away from the coverage of RSUs may further lead to the incompleteness of offloading tasks. Even if the offloading computation results can be sent back via the backbone network, the delay-sensitive service requirement cannot be met in real-time. Others apply ML or Generative Adversarial Networks (GAN) to predict vehicle trajectories by analyzing the movement data of vehicles [24–27]. Paradoxically, such solutions require computing capability with excellent performance to ensure the accuracy of the training model, which ignores the constraints of limited computing capability available at vehicles themselves [14]. Accordingly, the schemes with low complexity are lightweight, which is more appropriate in such scenarios.

Third, the optimization objective of most studies is mainly focused on the task execution delay or energy consumption [7–9], instead of the joint optimization of the overall performance and stability of the system [10, 13, 14, 28]. In urban scenarios, vehicles need to handle massive amounts of data with complexity and diversity generated by applications, and the services need to be provided in real-time to avoid traffic accidents. Consequently, task offloading must take the real-time response for applications and the low energy consumption for vehicles as two crucial optimization objectives.

To fill the aforementioned gaps, we develop a mobility-aware dependent task offloading (MESON) scheme for urban VEC. First, we establish a novel VEC-based computation offloading model, which considers the data dependency of tasks. Second, we leverage the Deep Reinforcement Learning (DRL) algorithm to realize adaptive task offloading without any prior knowledge, which is a good candidate for processing complex applications with data dependency [29–32]. Third, a vehicle mobility detection algorithm is proposed to detect the communication time between vehicles and RSUs for avoiding unreasonable decisions, which decreases the magnitude of the action space. Fourth, we design a task priority determination algorithm to determine the processing order of tasks in the waiting queue, which is aggregated by employing multi-criteria. The main contributions of this paper can be summarized as follows.

- A VEC-based computation offloading model is proposed with consideration of the data dependency of tasks. The minimization of the average response time and average energy consumption of the system is defined as a combinatorial optimization problem. The proposed offloading system can effectively provide computational services in real-time with low energy consumption.
- A novel mobility-aware task offloading scheme is proposed by employing DRL, which leverages the Deep Deterministic Policy Gradient (DDPG) algorithm to train the offloading strategy. To deal with the time-varying trajectories of vehicles in urban scenarios, we design a mobility detection algorithm to combine with the DDPG algorithm, which ensures stable communication links between vehicles and RSUs. The mobility detection algorithm also decreases the size of the action space, further leading to the improvement of training efficiency.
- To improve the system stability and success rate of tasks, a novel priority determination algorithm is also designed to sort the task queue of RSUs after the offloading process. The priority of the task is aggregated by employing multi-criteria, including the computation capability, the maximum tolerance

time durations, and the features of the dependent task.

The rest of this paper is organized as follows. Section 2 presents the system model and problem formulation. The details of the MESON task offloading scheme are presented in Section 3. Experimental evaluations and discussions are presented in Section 4. Finally, Section 5 concludes this paper.

2 SYSTEM MODEL AND PROBLEM FORMULATION

In this section, we first describe the VEC-based task offloading model, which considers the data dependency of tasks. Second, task offloading is defined as a joint optimization problem of reducing the of average system response time and average system energy consumption.

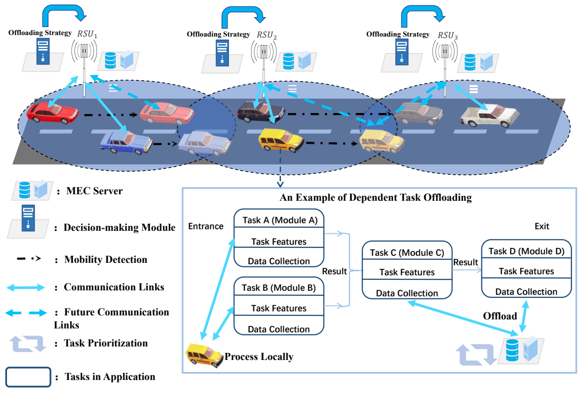


Fig. 1. The VEC network architecture.

2.1 System Overview

This subsection presents the overall system architecture of VEC-based task offloading. As shown in Fig. 1, there are n RSUs and i vehicles. Define $K = \{k_1, \ldots, k_n\}$ and $V = \{v_1, \ldots, v_i\}$ as the sets of RSUs (or the MEC servers) and vehicles, respectively. Vehicles and RSUs are connected via V2I communication. All RSUs are installed with an MEC server to provide computation resource support for surrounding vehicles, and a decision-making module to calculate the offloading strategy for each vehicle within the V2I communication range. In addition, the mobility detection method in the decision-making module predicts the future location of each vehicle and assists the task offloading decision-making, which is proposed in Section 3.2.2. When vehicles offload their tasks to an RSU, vehicles need to wait for the RSU to return the calculation result. At the same time, RSUs prioritize the execution order of tasks and execute them. The detail of task priority determination is proposed in Section 3.2.3. We assume that the generation of a new application in vehicles is a Poisson process, and the application arrival rate is Υ. The application can be divided into several tasks with data dependencies. As shown, each task consists of three parts: the task itself (module or code unit), task features, and data collected by vehicle sensors (presented in Section 2.2). An example is shown in Fig. 1.

TABLE 2
Main notations.

Notation	Description
K	set of RSUs
V	set of vehicles
$A_j^{v_i}$	application generated by vehicle v_i
$\mathcal{B}^j$	set of tasks in application $A_j^{v_i}$
H^j	matrix of task dependency of application $A_j^{v_i}$
$\Psi_{i,n}$	communication rate between vehicle v_i and RSU k_n
p_z	processing time of task b_z^j
τ_z	waiting time of task b_z^j
$o_{z,n,x}$	task offloading indicator
c_z	communication time of task b_z^j
T_{res}^j	response time of application $A_j^{v_i}$
T_x	average application response time at the time of the x-th slot
E_z^{proc}	processing energy consumption of task b_z^j
E_z^{comm}	communication energy consumption of task b_z^j
E_z^{total}	total energy consumption of task b_z^j
E^{A_j}	total energy consumption of application $A_j^{v_i}$
E_x	average energy consumption at the x-th time slot
$P_{b_z^j}$	priority of task b_z^j
$\triangle_{i,n}(x)$	distance between vehicle v_i and RSU k_n at the time slot of x
$T_{i,n}^{\text{con}}(x)$	communication time between vehicle v_I and RSU k_n
$G^{A_j}(x)$	connectivity matrix for the application $A_j^{v_i}$ at the x-th time slot
$\mho$	the number of tasks in the task queue
(lt^{v_i}, gt^{v_i})	the location of vehicle v_i

The vehicle has generated a new application, which contains four tasks. Task A, task B, and task D are the entrance and exit of the application program, task C requires task A and task B as the input data and task D requires task C as the input data. The vehicle can offload these tasks to RSUs or execute them locally by the decisions of the decision-making module. The detailed description of the decision-making module is proposed in Section 3. For instance, task A and task B are scheduled to execute locally, and task C and task D are offloaded to an MEC server within the communication range. If the communication link is interrupted due to the mobility of the vehicle during the offloading process, the offloading task is considered incomplete.

2.2 Application Model

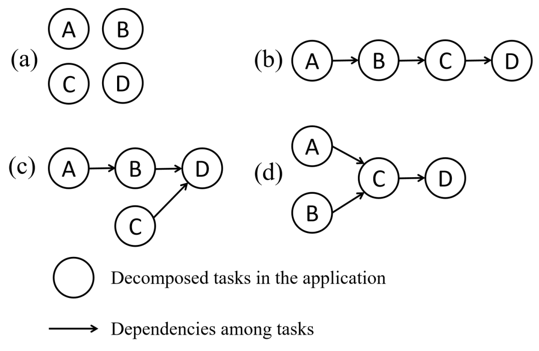


Fig. 2. The example of different dependencies between tasks.

This subsection proposes the application model with task dependency. In order to match the practical environment, we consider different types of applications by defining different parameters such as the maximum tolerance time durations, architecture of applications, and number of tasks in applications. Then, we define various types of tasks by using different parameters, such as the input/output data size, local data size, and computation load.

We define $A_j^{v_i} = \{v_i, m, H^j, d_{max}\}$ as the application generated by vehicle v_i, where m indicates the number of tasks in application $A_j^{v_i}$, H^j represents the task interdependency of these tasks, and d_{max} indicates the maximum tolerant response time of application $A_k^{v_i}$, respectively. The application $A_j^{v_i}$ can be divided into several tasks $\mathcal{B}^j = \left\{b_1^j, \ldots, b_z^j\right\}$. Matrices $H^j = \left(H_{l,r}^j\right)_{m \times m}$ are used to represent the dependency of tasks in application $A_j^{v_i}$, where $H_{l,r}^j = \{0,1\}$, $1 \leq l < r \leq m$. $H_{l,r}^j = 1$ means that task r needs the result of task l. As shown in Fig. 2 (c), task B needs to wait for the result of task A, while task D needs the results of task B and task C. Therefore, the adjacency matrix for the direct graph in Fig. 2 can be given by

$$\mathrm{H}^1 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, H^2 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix},$$

$$\mathrm{H}^3 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}, H^4 = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

We define the task b_z^j of application $A_j^{v_i}$ as a tuple $\varphi_z = (\omega_z, \lambda_z, \gamma_z, \delta_z)$, where ω_z represents the computational load, λ_z represents the size of local data collected by its own sensors of vehicles, γ_z represents the size of input data, and δ_z represents the size of output data of task b_z^j in application $A_j^{v_i}$. Thereby, the computation load of the application is $\sum_1^m \omega_z$. If task φ_z is the entry point of this application, then $\lambda_z = \gamma_z$. The correlation between input data and output data between tasks is defined as

$$\gamma_r = \sum_{l=1}^{r} H_{l,r+1}^j \cdot \delta_l \tag{1}$$

The attributes of each task $(\omega_z, \lambda_z, \gamma_z, \delta_z)$ are determined by the applications generated by vehicles, and different applications have various task dependencies and attributes. Only when all tasks in the application are executed successfully in sequence, the entire application is considered complete. Therefore, the offloading decision of each task is critical.

2.3 Communication Model

The communication model of the VEC network is presented in this subsection. The system is divided into X time slots. During each time slot, the whole system is quasi-static. During each time slot, the positions of vehicles are stable, but the positions of vehicles may vary along with the changes of different time slots. Each vehicle in the system has a communication link with the RSU by adopting V2I communication. The maximum V2I communication range is $\mathcal{R}_0$. In the proposed VEC network, the achievable communication rate of V2I communication is defined as:

$$\Psi_{i,n} = \phi \cdot \log_2\left(1 + \frac{p_{i,n} \cdot g_{i,n}}{\sigma_0 + \sum_{j=1}^{i} \varrho_j p_{j,n} g_{j,n}}\right) \tag{2}$$

where $g_{i,n} = (\triangle_{i,n})^{-\imath} \jmath_{in}$ represents the instantaneous channel gain from vehicle v_i, $\triangle_{i,n}$ is the instantaneous distance between vehicle v_i and RSU k_n, $\imath$ and $\jmath$ denote the path loss and the fading factor, respectively. $\varrho_j = \{0,1\}$ represents the communication indicator of vehicle v_j, which means whether vehicle v_j is maintaining a communication link with the RSU. ϕ denotes the channel bandwidth, and σ_0 refers to the background noise.

According to Eq. 2, critical interference can be incurred if too many vehicles decide to offload tasks to the RSU, thus leading to disastrous impact on the communication efficiency of the VEC network. In addition, the interference on communication link may cause the changes of the V2I communication range. The instantaneous V2I communication range can be formulated as:

$$\mathcal{R} = \mathcal{R}_0 \times \ell\left(1 + \frac{p_{i,n} \cdot g_{i,n}}{\sigma_0 + \sum_{j=1}^{i} \varrho_j p_{j,n} g_{j,n}}\right) \tag{3}$$

where ℓ is the adjustment coefficient.

In addition, due to the movement of vehicles, the instantaneous distance between the vehicle and RSU is also an important factor that affects the communication quality between vehicles and RSU, which needs to be highlighted.

2.4 Response Time Model

In this section, we propose the response time model in VEC-based task offloading. The response time of a task can be defined as a sum of processing time and queuing time.

The time duration required by the vehicle or RSU to process tasks is defined as the processing time, which is given by

$$p_z = \begin{cases} \frac{\omega_z}{f^{v_i}} & , o_{z,n,x} = 0 \\ \frac{\omega_z}{f^{k_n}} & , o_{z,n,x} = 1 \end{cases} \tag{4}$$

where f^{v_i} and f^{k_n} represent the processing capability of vehicle v_i and RSU k_n, respectively. $o_{z,n,x} = \{0,1\}$ is the task offloading indicator, $o_{z,n,x} = 1$ represents task b_z^j is offloaded to the n-th RSU for processing during the x-th time slot, and $o_{z,n,x} = 0$ refers to task b_z^j is processed locally during the x-th time slot. The queuing time of a task can be divided into two parts: waiting time and communication time. The waiting time represents the time duration of the maximum processing time for all predecessor tasks. The communication time represents the time duration of the maximum transmission time for all input data. To formulate the communication time and waiting time, we need to consider whether the task requires the results of other tasks as input. The waiting time τ_z and the communication time c_z of task b_z^j are defined as follows.

$$\tau_z = \begin{cases} 0 & , \sum_{l=1}^{m} H_{l,z} = 0 \\ \max\{p_l H_{l,z} \mid 1 \leq l < m\} & , \sum_{l=1}^{m} H_{l,z} \neq 0 \end{cases} \tag{5}$$

$$c_z = \begin{cases} \frac{\lambda_z}{\Psi_{i,n}} & , \sum_{l=1}^{m} H_{l,z} = 0 \\ \max\left\{\frac{\gamma_z}{\Psi_{i,n}} H_{l,z} \mid 1 \leq l < m\right\} & , \sum_{l=1}^{m} H_{l,z} \neq 0 \end{cases} \tag{6}$$

Thereby, the response time of application A^j is defined as

$$T_{res}^j = \sum_1^m (p_z + \tau_z + c_z) \tag{7}$$

Consequently, the average application response time at the time of the x-th slot is defined as

$$T_x = \frac{1}{j}\sum_{1}^{j} T_{res}^{j} \tag{8}$$

2.5 Energy Consumption Model

We divide the energy consumption of a task into communication energy consumption and processing energy consumption. Processing energy consumption represents the energy required to complete a single task. Communication energy consumption represents the energy required for data transmission of the predecessor task. The processing energy consumption of task b_z^j is defined as

$$E_z^{proc} = \begin{cases} p_z^{v_i} \times E^{local} & , o_{z,n,x} = 0 \\ p_z^{k} \times E^{mec} & , o_{z,n,x} = 1 \end{cases} \tag{9}$$

where $p_z^{v_n}$ and p_z^k are the time duration required for vehicle v_i and RSU to process task b_z^j, respectively. E^{local} and E^{mec} are the CPU power of vehicle v_i and MEC server, respectively. To define the communication energy consumption, we need to consider whether this task requires the results of other tasks as input. Therefore, we can define the communication energy consumption of a task as

$$E_z^{comm} = \begin{cases} \frac{\lambda_z}{\Psi_{i,n}} \times E^{trans} & , \sum_{l=1}^{m} H_{l,z} = 0 \\ c_z \times E^{trans} & , \sum_{l=1}^{m} H_{l,z} > 0 \end{cases} \tag{10}$$

where E^{trans} is the transmission energy consumption of each time slot. When a task does not have a predecessor task, $\frac{\lambda_i}{\Psi_{i,n}}$ represents the time required for task offloading. c_z represents the communication time of these data, which is given by Eq. 6. Therefore, the total energy consumption for completing task b_z^j is defined as

$$E_z^{total} = E_z^{comm} + E_z^{proc} \tag{11}$$

Thereby, the energy consumption to complete application A_j is defined as

$$E^{A_j} = \sum_{1}^{m} E_z^{total} \tag{12}$$

Consequently, the average energy consumption at the x-th time slot is defined as

$$E_x = \frac{1}{j}\sum_{1}^{j} E^{j} \tag{13}$$

2.6 Problem Definition

In this subsection, we propose the optimization objective of the system, which is the maximization of the utility function. The utility function reflects the QoS of the system. Given that we employ a DRL-based approach to train the offloading decisions, which requires continuous training to achieve relatively optimal results. The reward function in the MDP model of the DRL-based approach must be relatively correlated with the utility function. We define the utility of each time slot as the sum of the average energy consumption difference and the average response time difference between this time slot and the previous time slot. Accordingly, the utility function $U(x)$ is defined as

$$U(x) = \alpha\frac{T_{x-1} - T_x}{T_{x-1}} + \beta\frac{E_{x-1} - E_x}{E_{x-1}} \tag{14}$$

where α and $\beta \in [0, 1]$ are relative weighs of optimization objectives of the system. In practical scenarios, we can set the value of the tuning parameter based on the preferences of users and the network state.

Given that we apply DRL-based methods as a potential solution to the offloading problem, system utility is also defined as part of the environment state. DRL-based methods need to decide actions based on the current state of the system. The specific principle of MESON will be introduced in the next section.

3 PROPOSED SOLUTION

This section presents the MESON task offloading scheme in detail. We introduce the preliminaries of our scheme at the beginning of this section. Next, the MESON task offloading scheme is proposed in four aspects. Finally, we present the complexity analysis of MESON.

3.1 Preliminaries

DRL: The current mainstream DRL algorithms mainly have two categories, policy-based algorithms, and value-based algorithms. The former defines the value function of a state or action to represent the expected reward that can be obtained after reaching a certain state or performing a certain action. These algorithms tend to select the state or action with the greatest value. A typical algorithm of value-based DRL is Deep Q-learning (DQN), which approximates the optimal value function through DNN. However, DQN uses one-step rewards to indirectly update the deep Q-networks to further obtain deterministic policies. This one-step update rule leads to a slow learning process and is not suitable for models with continuous action spaces. Moreover, when describing a certain state in the state space, possibly due to the limitation of the size of observations, different states have the same feature description, further failing to reach the optimal solution. The latter directly use DNN to update the policy by performing a gradient descent method on the total reward. Therefore, it converges faster than value-based DRL algorithms and is suitable for solving problems with continuous action spaces. However, when solving combinatorial optimization problems like task offloading, it still has two shortcomings. First, the method of policy search is more likely to converge to the local extrema instead of the optimum. Second, variance tends to be too large when evaluating strategies.

DDPG: DDPG is a DRL-based algorithm that combines Actor-Critic (AC) and DNN. AC is a hybrid of policy-based and value-based algorithms, in which it has two modules, actor and critic. The actor selects an action based on probability. Then, the critic judges the action's value, and the actor modifies selection behavior according to the critic's evaluation. DDPG improves the AC method through the powerful training ability of DNN, and it uses experience replay and the target network to perform gradient descent on the main network. In addition, it uses the policy network to output deterministic actions directly. DDPG combines the advantages of two categories of algorithms and utilizes the powerful training capabilities of DL, which has more advantages in solving task offloading decision-making problems with multi-dimensional input and continuous action space than traditional DRL.

3.2 The proposed MESON task offloading scheme

In this subsection, first, the DRL-based task offloading model is presented in 3.2.1. Second, the mobility detection algorithm is proposed to avoid unreasonable offloading decisions in 3.2.2, which improves the training efficiency of MESON. Third, we present the priority determination scheme in 3.2.3, which adjusts the processing order of tasks in RSUs. Finally, we describe the overall training process of the MESON task offloading scheme in 3.2.4.

3.2.1 The DRL-based Task Offloading Model Design

Here, the task offloading model is defined as an MDP, which includes the following aspects.

System state $S(x) \in \mathcal{S}$: the system state $S(X)$ represents the environmental information at the x-th time slot. It mainly includes the instantaneous utility of the system and the instantaneous location of the vehicles. Hence, the system state $S(X)$ can be formulated as $S(x) = \{U(x), L(x)\}$. $U(x)$ represents the average energy consumption of the system, which is given by Eq. 14. $L(x)$ is a set of location information of all vehicles, which is defined as

$$L(x) = \{(lt^{v_i}, gt^{v_i})\} \quad (v_i \in V) \tag{15}$$

where $\{(lt^{v_i}, gt^{v_i})\}$ represents the location of vehicle v_i, lt^{v_i} and gt^{v_i} are the latitude and longitude of vehicle v_i, respectively.

Action space $A(x) \in \mathcal{A}$: in the x-th time slot, each vehicle in the system will make several offloading decisions. $A(x)$ is a set of task offloading decisions of all vehicles, which can be represented by $A(x) = \{a^{v_i}(x)\}$ $(v_i \in V)$, where $a^{v_i}(x)$ represents the offloading decision of vehicle v_i in the x-th time slot. $a^{v_i}(x)$ contains the task offloading decisions of vehicle v_i for each task, represented by $o_{z,n,x} = \{0, 1\}$, which is mentioned above.

Reward function $r(x)$: since the system objective is the long-term maximization of the utility function, our reward function must be relatively correlated with the utility function. Each time slot is defined as an episode. The reward of each episode is the utility difference between this episode and the previous one. As presented in [33, 34], this is the commonly used reward definition mechanism in DRL-based algorithms. The reward function $r(x)$ is defined as

$$r(x) = U_x - U_{x-1} \tag{16}$$

3.2.2 Mobility Detection

As shown in Fig. 3, considering the dynamic nature of vehicles, broken links between vehicles and RSUs caused by vehicles driving away from the coverage of RSUs may further lead to the incompleteness of the offloading tasks. Even if the offloading computation results can be sent back via the backbone network, the delay-sensitive service requirement cannot be met in real-time. As shown in Fig. 4, we assume

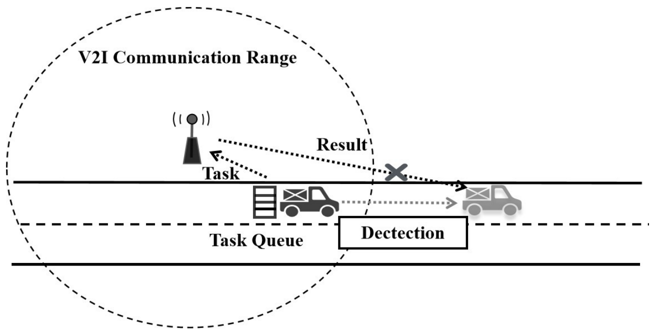


Fig. 3. Description of mobility detection.

that RSUs are evenly distributed in each road section. If the vehicle moves to another road section at the time slot of x_1 before receiving the computation result, the total response time of the computation task is $T = T_1 + T_2 + T_3 + T_4$, where T_1 represents the transmission time of the task, T_2 represents the execution time, T_3 represents the communication time between RSUs, and T_4 represents the transmission time of the result respectively. Under this circumstance, although the computation result can be sent to the vehicle, the service delay is too high. Therefore, processing this task locally may be a better choice than offloading, which eliminates the transmission delay. In general, trajectory prediction is an effective assistant for task offloading in the vehicular network.

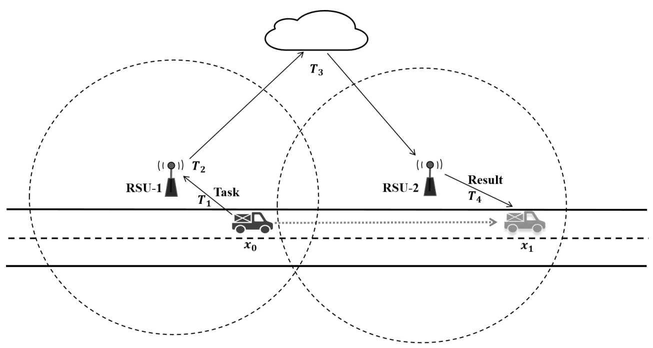


Fig. 4. Description of the impact of vehicle mobility on response time.

Therefore, we propose a mobility detection algorithm to estimate the communication time between vehicles and RSUs and further reduce the action space of the task offloading system. The main idea of this algorithm is to construct a connectivity matrix for each application generated by vehicles. The connectivity matrix indicates whether each task in the application can be successfully completed before the communication link is interrupted. The mobility detection algorithm can be regarded as the pre-processing of the overall scheme. It aims to effectively avoid the DRL-based algorithm to explore unnecessary decision space and improve the training efficiency.

We use a parameter tuple $\varpi_i(x) = \{lt_i(x), gt_i(x), \overrightarrow{\partial_i(x)}\}$ to represent the mobility of vehicle v_i at the x-th time slot, where $lt_i(x)$ and $gt_i(x)$ represent the latitude and longitude and $\overrightarrow{\partial_i(x)}$ represents the modulo of the vector speeds of the vehicle v_i at the time of x slot. The distance between vehicle

v_i and RSU k_n at the x-th time slot is defined as

$$\triangle_{i,n}(x)=\sqrt{\left(lt_i^{\mathrm{v}}(x)-lt_n^{r}(x)\right)^2+\left(gt_i^{\mathrm{v}}(x)-gt_n^{r}(x)\right)^2} \quad (17)$$

To better measure the impact of the mobility of the vehicle on the system, we consider the direction and speed of the vehicle stable within a small time duration. To ensure that the vehicle receives the results from the RSU when it is within the coverage of the RSU, we must estimate the time duration of the vehicle being within the coverage of the RSU. We suppose the coverage of the RSU is $\mathcal{R}$, which is mentioned before. Based on the above definitions, the communication time between vehicle v_i and RSU k_n is defined as

$$T_{i,n}^{\text{con}}(x)=\frac{\mathcal{R}-\triangle_{i,n}(x)}{\overrightarrow{\partial_i(x)}} \quad (18)$$

To maintain a stable communication link while offloading, the communication time between two devices must be greater than or equal to the task response time. Therefore, the matrix of connectivity for application A_j at the x-th time slot is defined as

$$G^{A_j}(x)=[g_{z,n}(x)]_{M\times\mathcal{N}}\,,\quad b_z^j\in\mathcal{B}^j, n\in K \quad (19)$$

where $g_{z,n}(x)=1$, if $T_{i,n}^{\text{con}}(x)\geq(p_z+\tau_z+c_z)$; otherwise, it is 0. This inequality means that the communication time needs to be greater than the response time of the task. The connectivity matrix describes the offloading feasibility of task b_z^j in application A_k based on the mobility of vehicle v_i, where M is the number of tasks in application A_k, $\mathcal{N}$ is the number of RSUs within the communication range at the x-th time slot. $g_{z,n}(x)=1$ means that task b_z^j in application $A_j^{v_i}$ can be offloaded to RSU k_n.

3.2.3 *Priority Determination*

Here, a priority determination algorithm is introduced by fully considering the impact of task priority on the system efficiency and sorting the task queue in RSUs. The priority determination algorithm is mainly aggregated by the following criteria.

- Tasks with the higher computing capability difference between local computing and MEC computing have the higher priority.
- Tasks with strict response constraints have the higher priority.
- Tasks with more successor tasks have the higher priority.

Accordingly, the priority of task b_z^j is defined as

$$P_{b_z^j}=\xi\left(\iota\frac{f^{k_n}}{f^{v_i}}+\chi\frac{d_{max}^j}{\overline{D_{max}^j}}\right) \quad (20)$$

where $P_{b_z^j}$ represents the priority of task b_z^j. Vehicles will process the tasks with the higher priority earlier. ξ represents the number of successor tasks of task b_z^j. It means that the tasks with more successor tasks have the higher priority. ι and χ are weighting coefficients and $\iota+\chi=1$. f^{k_n} and f^{v_i} represent the computation capability of vehicle v_i and RSU k_n, respectively. It means that the tasks with the higher computing capability differences between local computing and MEC computing have the higher priority. d_{max}^j represents the maximum tolerant response time of application A^j which is mentioned before. $\overline{D_{max}^j}$ is the total maximum tolerant time of all applications in the task queue of the RSU, which is defined as

$$\overline{D_{max}^j}=\sum_{0}^{\mho}d_{max}^j \quad (21)$$

where $\mho$ is the number of tasks in the task queue. It denotes that tasks with strict response constraints have the higher priority.

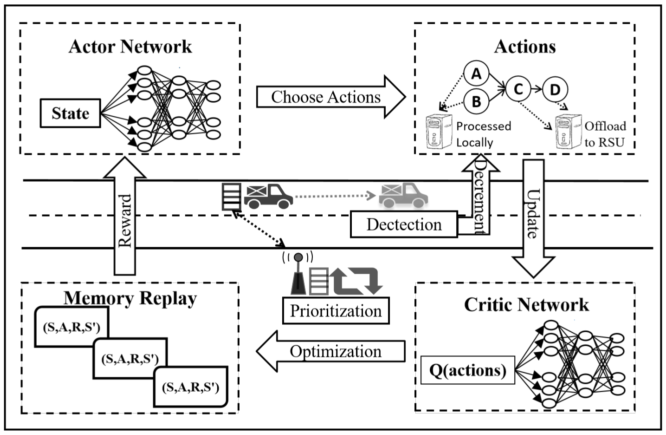


Fig. 5. Framework of MESON task offloading Scheme.

3.2.4 *The Training process of MESON Task Offloading Scheme*

As shown in Fig. 5, the DDPG algorithm is adopted for task offloading, we define the whole VEC-based task offloading system as an MDP, which includes an agent, a reward function $r(x)$, an environment, states $S(x)\in\mathcal{S}$, and actions $A(x)\in\mathcal{A}$. The overall training system includes four networks, i.e., the main actor-network, the target actor-network, the main critic-network, and the target critic-network. The agent continuously interacts with the environment and makes offloading decisions. The agent observes the current environment $S(X)\in\mathcal{S}$ and chooses an action in $\mathcal{A}$ according to the strategy π. The strategy π is the probability distribution of the action based on the current state. Then, the agent will receive the reward $r(x)$ and get into the next state. Our objective is to train an offloading strategy to realize the maximization of the expected reward, which is defined as

$$J=\mathbb{E}_{s_x\sim E,a_x\sim\pi}\left[R_1\right] \quad (22)$$

The critic-network evaluates the action a_x according to the reward function $r(x)$, it calculates the expected discounted reward of the action a_x from state $S(x)$. The expected evaluation $Q^\pi(s_x,a_x)$ of action a_x is defined as

$$Q^\pi(s_x,a_x)=\mathbb{E}_{s_x\sim E,a_x\sim\pi}\left[R_x\mid s_x,a_x)\right] \quad (23)$$

The agent tries to learn the best offloading strategy from its interactions with the environment and adjusts its

Algorithm 1: MESON task offloading scheme

```
Input: VEC network
Output: offloading decisions
1  Initialize the weights of actor's main network $\theta^{\mu}$
     and critic's main network $\theta^{Q}$;
2  Initialize actor's target network $\theta^{\mu'}$ and critic's
     target network $\theta^{Q'}$ by: $\theta^{\mu'} \leftarrow \theta^{\mu}$, $\theta^{Q'} \leftarrow \theta^{Q}$;
3  Initialize $B$;
4  for each episode do
5      Initialize the agent and the environment;
6      Initialize state $S(0)$ and the reward $r(0) = 0$;
7      Set the noise vector by $\mu'(s) = \mu(s \mid \theta^{\mu}) + \Delta\mu$;
8      for $\forall v_i \in V$ do
9          Generate different applications $A_j^{v_i}$ by
             application arrival rate $\Upsilon$ and decompose
             them into several tasks $\mathcal{B}^j$;
10         Calculate the relative distance $\triangle_{i,j}(x)$;
11         Calculate the communication time $T_{i,n}^{\text{con}}(x)$ by
             Eq. (20) and establish the connectivity
             matrix $G^{A_j}(x)$ by Eq. (21);
12         According to the policy $\pi$, select offloading
             decisions in the $G^{A_k}(x)$ matrix by
             $a_x^{v_i} = \mu(s_x \mid \theta^{\mu}) + \Delta\mu$;
13         Execute offloading decisions $a(x)$ from the
             simulation environment;
14     end
15     for $\forall k_n \in K$ do
16         Calculate the priority of tasks $P_{b_z^j}$ by Eq. (17);
17         Sort the task queue by $P_{b_z^j}$;
18     end
19     Receive reward $r(x)$ and transmit to $s(x+1)$;
20     Store $\{s_x, a_x, r_x, s_{x+1}\}$ to replay buffer $B$;
21     Sample a mini-batch of experience
         $\{s_x, a_x, r_x, s_{x+1}\}$ from $B$;
22     Update the critic network by minimizing the loss
         $L(\theta^Q)$ with the samples;
23     Use policy gradient to update actor network
         $\nabla_{\theta^{\mu}} J$;
24     Update the target networks by
         $\theta^{Q'} \leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q'}$ and
         $\theta^{\mu'} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'}$ ;
25 end
```

behavior. According to the strategy π^*, the Bellman optimal equation is defined as follows.

$$Q^*(s_x, a_x) = \mathbb{E}_{s_{x+1}\sim E}\left[r(s_x, a_x) + \gamma \max_{a_{x+1}} Q^*(s_{x+1}, a_{x+1})\right] \tag{24}$$

According to the definition of $Q^*(s_x, a_x)$, the optimal policy π^* is defined as

$$\pi^*(s) = \arg\max_{a\in\mathcal{A}} Q^*(s, a) \tag{25}$$

We can update the Q value by using the Time Difference (TD), which is defined as

$$\begin{aligned} &Q(s_x, a_x) \leftarrow Q(s_x, a_x) + \\ &\mu\left[r(s_x, a_x) + \sigma \max_{a_{x+1}} Q(s_{x+1}, a_{x+1}) - Q(s_x, a_x)\right] \end{aligned} \tag{26}$$

where μ is the learning rate, and σ is the discount factor, respectively. DDPG uses the experience buffer B to store the experience tuple $e_x = \{s_x, a_x, r_x, s_{x+1}\}$ of the agent, and randomly selects a batch of samples from B to calculate the loss function, which is used to update the critic-network, $Y(B)$ represents the sampled mini-batch. The update process of critic-network is defined as

$$\begin{aligned} &L\left(\theta^Q\right) = \mathbb{E}_{(s,a,r,s')\sim} \\ &Y(B)\left[\left(r + \gamma Q\left(s', \mu\left(s \mid \theta^{\mu'}\right) \mid \theta^{Q'}\right) - Q\left(s, a \mid \theta^Q\right)\right)^2\right] \end{aligned} \tag{27}$$

The actor-network deterministically maps state s to specific actions and uses policy gradient to optimize actions. The policy gradient process for updating the actor-network is defined as

$$\begin{aligned} &\nabla_{\theta^{\mu}} J \approx \mathbb{E}_{(s,a,r,s')\sim} \\ &Y(B)\left[\nabla_a Q\left(s, a \mid \theta^Q\right)\Big|_{a=\mu(s|\theta^{\mu})} \cdot \nabla_{\theta^{\mu}} \mu\left(s \mid \theta^{\mu}\right)\right] \end{aligned} \tag{28}$$

The MESON task offloading scheme is shown in Algorithm 1. We deploy MESON distributively in the decision-making module on each RSU, in which the RSUs can provide offloading services for vehicles within the V2I communication range. The system initializes the weights of the neural network and replay buffer (Lines 1-3). At the beginning of each episode, the VEC network is initialized (Lines 5-6). To fully explore the state space, we need to consider both exploration and exploitation. Thereby, at each episode, we initialize the noise vector (Line 7). All vehicles in the system generate applications by arrival rate Υ (Line 9). The connectivity matrix $G^{A_j}(x)$ is established by RSUs to ensure the stable link while offloading (Lines 10-11). The decision-making module in RSUs will make offloading decisions for each vehicle within the V2I communication range according to the policy π (Line 12). After executing the task offloading of each vehicle, RSUs sort the task queue by the priority of tasks (Lines 16-17). At the end of each episode, the experience tuple $\{s_x, a_x, r_x, s_{x+1}\}$ is stored in replay buffer B (Line 20). Finally, the policy gradient is used to update the agent's actor and critic network (Lines 21-24).

3.3 Complexity Analysis

In this section, we propose the theoretical complexity analysis of the MESON task offloading scheme. According to the analyzing model described in [35], it can be concluded that the time complexity and the space complexity of DDPG can be described as follows:

$$\begin{aligned} &O\left(\sum_{j=0}^{J-1} u_{actor}, j u_{actor}, j{+}1{+}\sum_{k=0}^{K-1} u_{critic}, k u_{critic}, k{+}1\right) \\ &+ O(N(s)). \end{aligned} \tag{29}$$

$$\begin{aligned} &O\left(\sum_{j=0}^{J-1} u_{actor}, j u_{actor}, j{+}1{+}\sum_{k=0}^{K-1} u_{critic}, k u_{critic}, k{+}1\right) \\ &+ O(N(s)) + O(N). \end{aligned} \tag{30}$$

The comparison between Algorithm 1 and the original DDPG algorithm reveals that MESON is different from it in Lines 10-12 and Lines 16-17. In Lines 10-12, we filter the offloading decisions, which decreases the action space, further leading to the decrement of space complexity and the improvement of learning efficiency. In Lines 16-17, we sort the task queue by $P_{b_z^j}$, which increases the time complexity. The time complexity in these lines is $O\left(NlogN\right)$. These comparisons show that our MESON scheme can effectively improve the learning efficiency of the network with a little additional time complexity.

4 RESULTS AND DISCUSSION

This section presents the performance evaluation of MESON. First, we introduce the evaluation metrics and scenarios. Second, we present the approaches for comparison. Third, we analyze the objective performance evaluation of MESON by comparing it with the existing schemes. Fourth, we present a summary of our evaluation results.

4.1 Evaluation metrics and scenarios

Our simulation experiments were conducted on a Win10 64-bit operating system with an AMD Ryzen 7 4800H CPU with 2.90GHz processor and 16GB RAM. Our simulation platform is conducted with Python 3.6 and TensorFlow 1.15.0 to implement the MESON task offloading scheme, in which we have made the platform open-source at https://github.com/NetworkCommunication/meson. Two simulation maps were downloaded from OpenStreetMap and generated traffic data flows by Simulation of Urban Mobility (SUMO). These maps are real urban scenes in Liaoning Province, China. The sizes of these two areas are $1500m$×$1500m$ and $3000m$×$3000m$, respectively. The general descriptions of these two maps are shown in Fig. 6. The first map has 9 main roads and 27 RSUs, and the second map has 13 main roads and 78 RSUs. RSUs are evenly distributed along each main road. More detailed simulation environment parameters are presented in Table 3. Then, we introduce the following performance metrics to evaluate the simulation results.

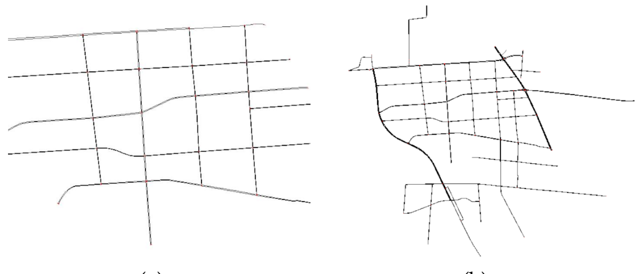


Fig. 6. (a) The simulation map of 1500m×1500m and (b) The simulation map of 3000m×3000m.

- *Total Reward* (TR) indicates the performance of the optimization objective of the system. The convergence speed of the total reward for each episode reflects the training efficiency of intelligent algorithms.

TABLE 3
Simulation parameters.

Parameter	Value
Network Size	1500m×1500m, 3000m×3000m
Number of RSUs	27, 78
Maximum Communication Range	600m
Number of Vehicles	500, 1000, 1500, 2000
Average Speed	0∼23m/s
Learning Rate	0.004
Discount Factor	0.99
Batch Size	64
Episode	2000
Application Arrival Rate	0.35
Number of Tasks in Applications	1∼4
Data Size of Tasks	0.8Mb∼1.2Mb
Length of Unit Time Slot	0.1s
Unit of Energy Consumption	1J
Maximum Tolerance Time Durations	1.5s, 1.8s, 2.1s, 2.4s, 2.7s, 3.0s
Processing Capability of RSUs	2.8G cycles/s∼3.2G cycles/s
Processing Capability of Vehicles	0.6G cycles/s∼0.8G cycles/s

- *Average Application Response time* (ART) is the key metric that can directly reflect network efficiency and communication quality in VEC. Further details on this metric are described in Section 2.4.
- *Average Energy Consumption* (AEC) is defined as the sum of the communication energy consumption and the processing energy consumption. Further details on this metric are described in Section 2.5.
- *Success Rate* (SR) is expressed as the number of applications completed within its deadline divided by the total number of applications. SR acts as a vital performance metric to evaluate the reliability and stability of the offloading system.

To comprehensively evaluate the performance of MESON, extensive experimental simulations are done with different parameter constraints.

- *Number of vehicles*: to demonstrate the impact of different vehicle densities on MESON, we set up comparative experiments with a different number of vehicles.
- *Maximum tolerance time durations*: to evaluate the performance of MESON in a delay-sensitive system, we compare the SR of various schemes under different maximum tolerance time durations.
- *Number of tasks in applications*: this constraint represents the complexity of applications. The performance of all schemes when dealing with applications with high complexity can better reflect their stability.

4.2 Approaches for comparison

We evaluate the superiority of the proposed scheme by comparing the basic algorithms and the intelligent algorithms. The specific introduction of each algorithm is listed below.

- *Local*: it is an offloading scheme without computation support from MEC [36]. Each vehicle in this scheme

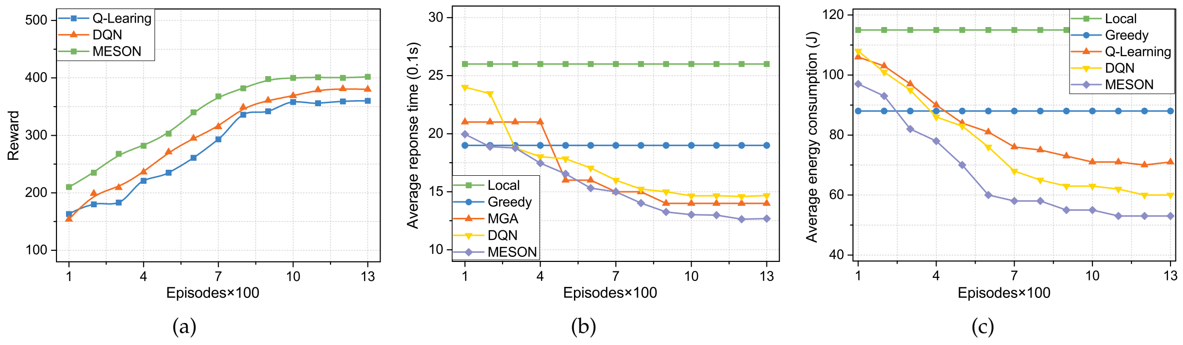


Fig. 7. (a) TR, (b) ART, and (c) AEC under various number of episodes.

needs to execute applications by its computing capability and cannot offload its tasks to any RSUs.

- *Greedy*: in this scheme, each vehicle in the system selects the current most suitable offloading decision by choosing an RSU or processing locally according to a certain optimization objective [37]. This scheme does not consider the long-term impact of the offloading decision on the system, but only selects the local optimum.
- *Q-Learning*: it is an offloading scheme based on RL [38]. The main idea of this scheme is to construct a Q-table with the current state of the system and the offloading decision and then select the offloading decision according to the Q value.
- *SA-DQN*: SA-DQN is an offloading scheme based on DQN [39]. It comprehensively considers task dependencies and different computing resources. DQN is a reinforcement learning algorithm based on Q-learning. The main idea of this scheme is to approximate the optimal value function through DNN.
- *MGA*: MGA is an improved scheduling scheme based on a genetic algorithm [40]. A genetic algorithm is an iterative process. At each iteration, the objective of the optimization problem of each individual is evaluated. According to the evaluation, the newly generated solution is used in the next iteration. Typically, the iterative process terminates when the maximum number of iterations is reached or a satisfactory level of evaluation is reached. This scheme takes into account the instability, heterogeneity, and inter-dependences of computing tasks.

It is also worth mentioning that MESON adopts a hybrid intelligent technique including AC and DNN. DNN uses the output features of the previous layer as the input of the next layer for feature learning. After feature mapping, the features of the existing space samples are mapped to another feature space, to learn a better feature representation of the existing input. DNN is a multi-layer unsupervised neural network, which is the fundamental idea of DL. DQN integrates DNN's powerful training capabilities. In this paper, we verify the superiority of MESON by taking DQN as a comparative counterpart. AC is a general architecture in RL. It provides a feasible solution for solving continuous decision-making problems in the field of RL. However, nowadays researchers are not inclined to use the AC method alone, but regard it as an inherent mode to solve dynamic problems and combine it with other cutting-edge methods. AC architecture is generally recognized as a valuable idea in the AI field, but its performance cannot be compared with some new training methods nowadays. Therefore, we did not take them as counterparts.

4.3 Evaluation of TR

Fig. 7(a) illustrates the TR for a different number of episodes. It is worth mentioning that to improve the validity of the data, we average the rewards generated by every hundred episodes. As shown, with an increasing number of episodes, the TR of all three schemes shows an upward trend. This tendency occurs because these three schemes are all intelligent algorithms based on reinforcement learning. The reinforcement learning system needs to accumulate experience to obtain strategies. With the continuous optimization of the strategy, the rewards in each episode will also increase. Therefore, with an ascending number of episodes, the TR is continuously improved. In our proposed MESON, we consider vehicle mobility as a critical metric to avoid many unreasonable decisions and improve the completion rate of tasks. MESON has smaller action space and state space than traditional reinforcement learning methods. Thus, MESON shows outstanding performance in the early episodes and keeps a high-speed increasing trend. When the number of episodes exceeds 900, MESON tends to converge. Q-learning and DQN tend to converge when the number of episodes exceeds 1000. As shown, MESON has the best training performance and convergence speed than other traditional reinforcement learning methods. The key reason is that MESON pays attention to the whole network performance and aims to provide reliable computation support in diversified environments rather than a certain metric. Moreover, MESON has a smaller action space than other schemes through mobility detection. Therefore, the improvement of the comprehensive performance of the offloading system enables MESON to obtain the highest TR.

4.4 Evaluation of ART

(1)*Varying number of episodes.* Fig. 7(b) shows the comparison of five schemes for evaluating the ART with a different number of episodes. Among all the comparisons, Local has the longest delay because when each task is calculated locally in vehicles, MEC servers cannot provide computing support for vehicles, and the computation capability of

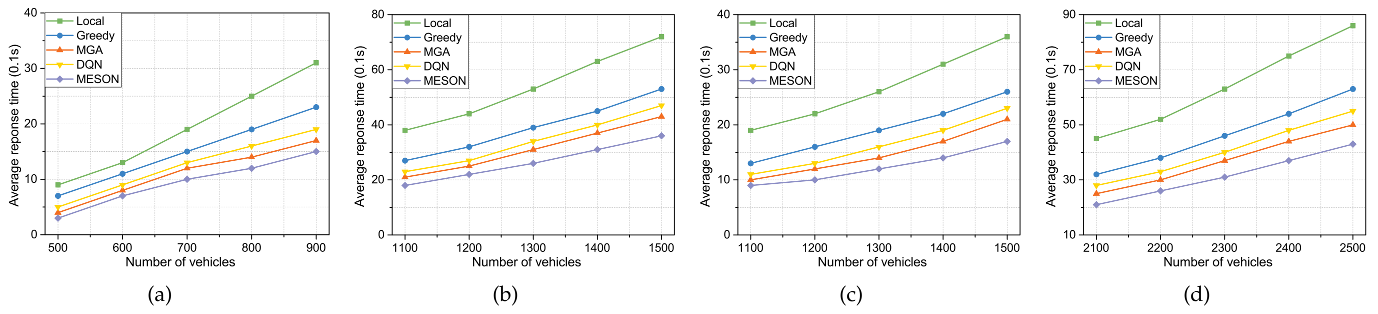


Fig. 8. (a) and (b) ART under different density of vehicles in an area of 1.5km×1.5km; (c) and (d) ART under different density of vehicles in an area of 3km×3km.

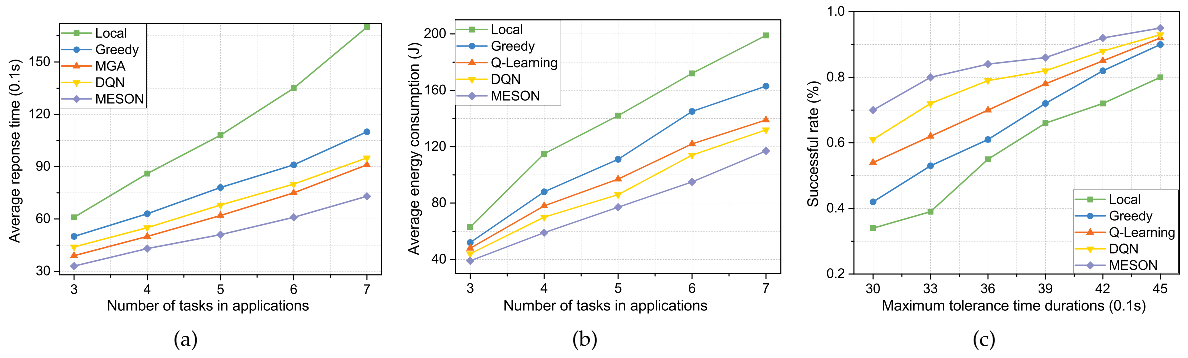


Fig. 9. (a) ART under various number of tasks in applications, (b) AEC under various number of tasks in applications, and (c) SR with different maximum tolerance time durations.

vehicles is insufficient. This situation results in the highest ART in comparison. Among all the comparisons, Greedy achieves the best performance when the number of episodes is less than 200. The main reason is that the greedy algorithm only considers current optimal solutions instead of long-term system rewards. Therefore, Greedy can show better performance than other intelligent algorithms in the early episodes. With an increasing number of episodes, the ART of other schemes shows a downward trend. This is because these four schemes need to achieve their best performance through training or iteration. Compared to other intelligent schemes, MESON achieves the best performance in ART when the number of episodes exceeds 700. This can be explained in three aspects: First, MESON uses DDPG based on DRL to support complex tasks with large state spaces and continuous action spaces. Therefore, MESON can achieve better performance when solving computational offloading problems with high-dimensional inputs and continuous action spaces. Second, MESON uses mobility detection to avoid many wrong decisions, which makes MESON have a higher convergence speed than other intelligent algorithms. Third, MESON considers the priority of tasks, which reduces the queuing time of each task. Thus, the efficiency of the offloading system is improved. Therefore, the results show that MESON has an excellent performance in ART.

(2)*Varying density of vehicles*. To evaluate the effect of different vehicle densities on the performance of ART and verify the scalability of our experiments, we use two simulation maps downloaded from OpenStreetMap and generated traffic data flows by SUMO. These maps are real urban scenes in Liaoning Province, China. The sizes of these two areas are $1500m$×$1500m$ and $3000m$×$3000m$, respectively. Fig. 8 show the vehicle density on the ART. As the number of vehicles expands, the ART of all schemes increases because the number of vehicles reflects the traffic situation. To prove the scalability of our scheme, we conduct two sets of experiments to evaluate the performance of all schemes. As shown, Fig. 8(a) and Fig. 8(c) show the changes of ART with low-density of vehicles, and 8(b) and Fig. 8(d) show that with high-density of vehicles, respectively. The ART difference between MESON and other schemes increases as the vehicle density increases. For instance, when the number of vehicles is 500, the ART difference between MESON and Local is about 5 slots; and when the number of vehicles reaches 900, the difference between MESON and Local is about 9 slots. The performance of schemes under high vehicle density reflects their scalability and robustness. Compared to other schemes, MESON achieves the best performance in ART under all vehicle densities for the following reasons. First, the ART of the offloading system can be affected by a variety of factors. Our proposed MESON comprehensively considers various factors of computation offloading of vehicles in urban traffic scenarios, rather than a certain metric. Second, our proposed MESON leverages a priority determination algorithm to sort the task queue. In the congested section of the road, the number of tasks also increases, and the processing order of tasks will be critical. Therefore, MESON has excellent scalability and robustness.

(2)*Varying number of tasks in applications*. Fig. 9(a) shows the relationship between the ART and the task number in applications. The number of tasks in applications represents the complexity of applications. The ART of completing an application is the sum of the queuing time and processing time of all tasks in this application. With the increasing number of tasks in applications, MESON has greater advantages than other schemes. The main reason for this tendency is that MESON determines the processing order according to the characteristics of the dependent tasks. MESON enables the queuing time of tasks to become smaller through the priority determination algorithm. This advantage becomes more pronounced when the number of tasks in the application is high.

4.5 Evaluation of AEC

(1)*Varying number of episodes*. Fig. 7(c) shows the comparison of five schemes for evaluating the AEC under a varying number of episodes. Among all the comparisons, Greedy achieves the best performance when the number of episodes is less than 200 because the greedy algorithm only considers current optimal solutions instead of long-term system rewards. When the number of episodes exceeds 1000, the AEC of all schemes tends to be stable. As shown, MESON is superior to other schemes in terms of convergence speed and performance on AEC. This can be explained in two aspects: first, MESON reduces the action space and state space through mobility detection, which ensures the training efficiency of the system and reduces the complexity of the system. Second, MESON will weigh a variety of factors to make offloading decisions to ensure that the AEC of the system is minimized to the greatest extent.
(2)*Varying number of tasks in applications*. Fig. 9(b) shows the relationship between the AEC and the task number in applications. The number of tasks in applications represents the complexity of applications. As shown, for all schemes, the AEC increases as the task number in applications increases because more power is required when the system handles applications of high complexity. With an increasing number of tasks in applications, the AEC difference between all schemes also increases. Especially, when the number of tasks in applications is 3, the AEC difference between MESON and Local is about 24 J; and when the number of tasks in applications reaches 7, the difference between MESON and Local is about 82 J. The performance of all schemes when dealing with applications of high complexity can better reflect their energy-saving effect. MESON achieves the lowest AEC under all number of tasks in applications. When processing task offloading with data dependency, MESON will determine the processing order according to the computing capability difference and the structure of applications, thereby reducing the queuing time of tasks and further leading to the decrement of transmission energy consumption.

4.6 Evaluation of SR

Fig. 9(c) shows the relationships between the SR and the maximum tolerance time durations duration of applications. For all schemes, the SR difference between all schemes decreases as the maximum tolerance time durations increase. For instance, when the maximum tolerance time duration is 45 slots, the SR difference between MESON and Local is about 0.15; and when the maximum tolerance time duration is 30 slots, the difference between MESON and Local is about 0.36. The maximum tolerance time duration reflects the urgency of applications. We can evaluate the reliability of all schemes in processing delay-sensitive applications by observing SR. Compared to other schemes, MESON achieves the best performance in SR, especially when the maximum tolerance time duration is relatively low. This is mainly beneficial for the following reasons. First, MESON will process tasks with strict response constraints. Second, MESON will determine the processing order of tasks according to the computation capability difference between vehicles and RSUs. Third, MESON uses mobility detection to ensure communication quality. Therefore, MESON has the highest SR among all schemes.

4.7 Summary of results

We evaluate the superiority of the proposed scheme by comparing the basic algorithm and the intelligent algorithms. In terms of basic algorithms, we first consider scheduling all computation tasks in local computing. The necessity of the offloading scheme can be demonstrated by comparing the differences between the proposed algorithm and Local. This is also a common way to measure the performance of an offloading system. Second, we compare the proposed scheme with Greedy. We can consider Greedy as an enumeration algorithm. The main idea of this type of algorithm is to use an enumeration method to find the optimal solution in the current state. However, such algorithms do not take into account future system reward expectations. The optimal solution in the current state does not guarantee long-term system benefits. Greedy can achieve better results in the early stage of training. However, with the increase in the number of iterations, the intelligent algorithm gradually tends to converge. In the end, the intelligent algorithm outperforms the Greedy algorithm.

To further verify the superiority of MESON, we compared other intelligent algorithms. In terms of intelligent algorithms, we compare Q-Learning and DQN based on reinforcement learning, and MGA based on genetic algorithms. These types of algorithms are relatively common and advanced algorithms in the field of task offloading. We verify the training effect and convergence speed of the proposed algorithm by comparing the system reward under different iterations. Additionally, to verify the practicability and comprehensive performance of MESON, we evaluate multiple optimization objectives with different experimental parameters. The simulation results demonstrate that MESON is superior compared to other task offloading schemes in terms of the average response time, average system energy consumption, and offloading successful rate.

5 Conclusion

In this paper, a novel VEC-based computation offloading model is proposed with the consideration of the data dependency of tasks. The reduction of the average response time and the average energy consumption of the system are defined as a combinatorial optimization problem. The proposed offloading system can effectively provide a computation service in real-time with little energy consumption. A mobility-aware task offloading scheme based on DRL that leverages the DDPG algorithm to train the offloading strategy is proposed to solve the combinatorial optimization problem. To deal with the time-varying trajectories of vehicles in urban scenarios, we design a mobility detection algorithm to combine with the DDPG algorithm, which ensures stable communication links between vehicles and RSUs. The mobility detection algorithm also realizes the decrement of the size of the action space, further leading to the improvement of training efficiency. We also design a novel priority determination algorithm to prioritize the task queue of RSUs after the offloading process for improving the system stability and the success rate of tasks. The priority of the task is aggregated by employing multi-criteria, including the computation capability, the maximum tolerance time durations, and the features of the dependent task. The simulation results demonstrate that MESON is superior compared to other task offloading schemes in terms of the average response time, average system energy consumption, and offloading successful rate.

Acknowledgment

This paper is supported in part by the Natural Science Foundation of Liaoning Province under Grant 2020-MS-237, and in part by the Liaoning Provincial Department of Education Science Foundation under Grant JYT19052. Al-Dubai would like to acknowledge the support of the UK Engineering and Physical Sciences Research Council (EPSRC) programme Grant: COG-MHEAR (Grant Reference: EP/T024917/1).

References

[1] L. Liu, C. Chen, Q. Pei, S. Maharjan, and Y. Zhang, "Vehicular edge computing and networking: A survey," *Mobile Networks and Applications*, vol. 26, pp. 1145–1168, jul 2020.

[2] L. Liu, C. Chen, Q. Pei, S. Maharjan, and Y. Zhang, "Vehicular edge computing and networking: A survey," *Mobile Networks and Applications*, vol. 26, no. 3, pp. 1145–1168, 2021.

[3] T. Q. Dinh, J. Tang, Q. D. La, and T. Q. S. Quek, "Offloading in mobile edge computing: Task allocation and computational frequency scaling," *IEEE Transactions on Communications*, vol. 65, no. 8, pp. 3571–3584, 2017.

[4] M. Chen and Y. Hao, "Task offloading for mobile edge computing in software defined ultra-dense network," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 3, pp. 587–597, 2018.

[5] M. Du, Y. Wang, K. Ye, and C. Xu, "Algorithmics of cost-driven computation offloading in the edge-cloud environment," *IEEE Transactions on Computers*, vol. 69, no. 10, pp. 1519–1532, 2020.

[6] M. Song, Y. Lee, and K. Kim, "Reward-oriented task offloading under limited edge server power for multi-access edge computing," *IEEE Internet of Things Journal*, vol. 8, no. 17, pp. 13425–13438, 2021.

[7] Y. Wang, P. Lang, D. Tian, J. Zhou, and D. Zhao, "A game-based computation offloading method in vehicular multiaccess edge computing networks," *IEEE Internet of Things Journal*, vol. PP, no. 99, pp. 1–1, 2020.

[8] K. Zhang, Y. Mao, S. Leng, Q. Zhao, L. Li, X. Peng, L. Pan, S. Maharjan, and Y. Zhang, "Energy-efficient offloading for mobile edge computing in 5g heterogeneous networks," *IEEE access*, vol. 4, pp. 5896–5907, 2016.

[9] X. Zhu, Y. Luo, A. Liu, M. Z. A. Bhuiyan, and S. Zhang, "Multiagent deep reinforcement learning for vehicular computation offloading in iot," *IEEE Internet of Things Journal*, vol. 8, no. 12, pp. 9763–9773, 2020.

[10] J. Zhao, Q. Li, Y. Gong, and K. Zhang, "Computation offloading and resource allocation for cloud assisted mobile edge computing in vehicular networks," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 8, pp. 7944–7956, 2019.

[11] J. Li, X. Cao, D. Guo, J. Xie, and H. Chen, "Task scheduling with uav-assisted vehicular cloud for road detection in highway scenario," *IEEE Internet of Things Journal*, vol. 7, no. 8, pp. 7702–7713, 2020.

[12] J. Zhang, H. Guo, and J. Liu, "Adaptive task offloading in vehicular edge computing networks: a reinforcement learning based scheme," *Mobile Networks and Applications*, vol. 25, no. 5, pp. 1736–1745, 2020.

[13] T. Alfakih, M. M. Hassan, A. Gumaei, C. Savaglio, and G. Fortino, "Task offloading and resource allocation for mobile edge computing by deep reinforcement learning based on sarsa," *IEEE Access*, vol. 8, pp. 54074–54084, 2020.

[14] J. Sun, Q. Gu, T. Zheng, P. Dong, A. Valera, and Y. Qin, "Joint optimization of computation offloading and task scheduling in vehicular edge computing networks," *IEEE Access*, vol. 8, pp. 10466–10477, 2020.

[15] R. Zhang, P. Cheng, Z. Chen, S. Liu, B. Vucetic, and Y. Li, "Calibrated bandit learning for decentralized task offloading in ultra-dense networks," *IEEE Transactions on Communications*, vol. 70, no. 4, pp. 2547–2560, 2022.

[16] H. Guo, J. Liu, and J. Lv, "Toward intelligent task offloading at the edge," *IEEE Network*, vol. 34, no. 2, pp. 128–134, 2020.

[17] G. Manogaran, G. Srivastava, B. A. Muthu, S. Baskar, P. Mohamed Shakeel, C.-H. Hsu, A. K. Bashir, and P. M. Kumar, "A response-aware traffic offloading scheme using regression machine learning for user-centric large-scale internet of things," *IEEE Internet of Things Journal*, vol. 8, no. 5, pp. 3360–3368, 2021.

[18] B. Dab, N. Aitsaadi, and R. Langar, "Q-learning algorithm for joint computation offloading and resource allocation in edge cloud," in *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pp. 45–52, 2019.

[19] K. Jiang, H. Zhou, D. Li, X. Liu, and S. Xu, "A q-learning based method for energy-efficient computation offloading in mobile edge computing," in *2020 29th International Conference on Computer Communications and Networks (ICCCN)*, pp. 1–7, 2020.

[20] B. Tian, L. Wang, Y. Ai, and A. Fei, "Reinforcement learning based matching for computation offloading in d2d communications," in *2019 IEEE/CIC International Conference on Communications in China (ICCC)*, pp. 984–988, 2019.

[21] Z. Tang, J. Lou, F. Zhang, and W. Jia, "Dependent task offloading for multiple jobs in edge computing," in *2020 29th International Conference on Computer Communications and Networks (ICCCN)*, pp. 1–9, 2020.

[22] G. Zhao, H. Xu, Y. Zhao, C. Qiao, and L. Huang, "Offloading dependent tasks in mobile edge computing with service caching," in *IEEE INFOCOM 2020 - IEEE*

Conference on Computer Communications, pp. 1997–2006, 2020.

[23] A. Akbar and P. R. Lewis, "The importance of granularity in multiobjective optimization of mobile cloud hybrid applications," *Transactions on Emerging Telecommunications Technologies*, vol. 30, oct 2018.

[24] Z. Yu, J. Hu, G. Min, Z. Zhao, W. Miao, and M. S. Hossain, "Mobility-aware proactive edge caching for connected vehicles using federated learning," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, pp. 5341–5351, aug 2021.

[25] L. Zhao, Y. Liu, A. Y. Al-Dubai, A. Y. Zomaya, G. Min, and A. Hawbani, "A novel generation-adversarial-network-based vehicle trajectory prediction method for intelligent vehicular networks," *IEEE Internet of Things Journal*, vol. 8, no. 3, pp. 2066–2077, 2021.

[26] D. Jeong, M. Baek, and S.-S. Lee, "Long-term prediction of vehicle trajectory based on a deep neural network," in *2017 International Conference on Information and Communication Technology Convergence (ICTC)*, pp. 725–727, 2017.

[27] Y. Xing, C. Lv, and D. Cao, "Personalized vehicle trajectory prediction based on joint time-series modeling for connected vehicles," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 2, pp. 1341–1352, 2020.

[28] Y. Hui, Z. Su, T. H. Luan, C. Li, G. Mao, and W. Wu, "A game theoretic scheme for collaborative vehicular task offloading in 5g hetnets," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 12, pp. 16044–16056, 2020.

[29] J. Wang, J. Hu, G. Min, W. Zhan, Q. Ni, and N. Georgalas, "Computation offloading in multi-access edge computing using a deep sequential model based on reinforcement learning," *IEEE Communications Magazine*, vol. 57, no. 5, pp. 64–69, 2019.

[30] S. Pan, Z. Zhang, Z. Zhang, and D. Zeng, "Dependency-aware computation offloading in mobile edge computing: A reinforcement learning approach," *IEEE Access*, vol. 7, pp. 134742–134753, 2019.

[31] J. Shi, J. Du, J. Wang, J. Wang, and J. Yuan, "Priority-aware task offloading in vehicular fog computing based on deep reinforcement learning," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 12, pp. 16067–16081, 2020.

[32] J. Wang, J. Hu, G. Min, A. Y. Zomaya, and N. Georgalas, "Fast adaptive task offloading in edge computing based on meta reinforcement learning," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, pp. 242–253, jan 2021.

[33] J. Wang, J. Hu, G. Min, W. Zhan, A. Y. Zomaya, and N. Georgalas, "Dependent task offloading for edge computing based on deep reinforcement learning," *IEEE Transactions on Computers*, vol. 71, no. 10, pp. 2449–2461, 2022.

[34] X. Zhu, Y. Luo, A. Liu, M. Z. A. Bhuiyan, and S. Zhang, "Multiagent deep reinforcement learning for vehicular computation offloading in iot," *IEEE Internet of Things Journal*, vol. 8, no. 12, pp. 9763–9773, 2021.

[35] C. Qiu, Y. Hu, Y. Chen, and B. Zeng, "Deep deterministic policy gradient (ddpg)-based energy harvesting wireless communications," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 8577–8588, 2019.

[36] J. Lu, Q. Li, B. Guo, J. Li, Y. Shen, G. Li, and H. Su, "A multi-task oriented framework for mobile computation offloading," *IEEE Transactions on Cloud Computing*, vol. 10, no. 1, pp. 187–201, 2022.

[37] F. Wei, S. Chen, and W. Zou, "A greedy algorithm for task offloading in mobile edge computing system," *China Communications*, vol. 15, no. 11, pp. 149–157, 2018.

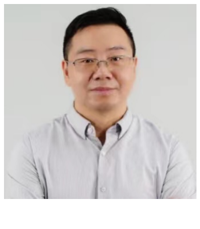
[38] F. Jiang, W. Liu, J. Wang, and X. Liu, "Q-learning based task offloading and resource allocation scheme for internet of vehicles," in *2020 IEEE/CIC International Conference on Communications in China (ICCC)*, pp. 460–465, 2020.

[39] B. Lin, K. Lin, C. Lin, Y. Lu, Z. Huang, and X. Chen, "Computation offloading strategy based on deep reinforcement learning for connected and autonomous vehicle in vehicular edge computing," *Journal of Cloud Computing*, vol. 10, no. 1, pp. 1–17, 2021.

[40] F. Sun, F. Hou, N. Cheng, M. Wang, H. Zhou, L. Gui, and X. Shen, "Cooperative task scheduling for computation offloading in vehicular cloud," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 11, pp. 11049–11061, 2018.

Liang Zhao (Member, IEEE) is a Professor at Shenyang Aerospace University, China. He received his Ph.D. degree from the School of Computing at Edinburgh Napier University in 2011. Before joining Shenyang Aerospace University, he worked as associate senior researcher in Hitachi (China) Research and Development Corporation from 2012 to 2014. He is also a JSPS Invitational Fellow (2023). He was listed as Top 2 % of scientists in the world by Standford University (2022). His research interests include ITS, VANET, WMN and SDN. He has published more than 150 articles. He served as the Chair of several international conferences and workshops, including 2022 IEEE BigDataSE (Steering Co-Chair), 2021 IEEE TrustCom (Program Co-Chair), 2019 IEEE IUCC (Program Co-Chair), and 2018-2022 NGDN workshop (founder). He is Associate Editor of Frontiers in Communications and Networking and Journal of Circuits Systems and Computers. He is/has been a guest editor of IEEE Transactions on Network Science and Engineering, Springer Journal of Computing, etc. He was the recipient of the Best/Outstanding Paper Awards at 2015 IEEE IUCC, 2020 IEEE ISPA, 2022 IEEE EUC and 2013 ACM MoMM.

Enchao Zhang is a student at Shenyang Aerospace University, China. He is currently studying for his M.S. degree in Shenyang Aerospace University. His research interests include mobile edge computing, computation offloading, digital-twins, content caching and resource allocation.



Shaohua Wan (Senior Member, IEEE) received Ph.D. degree from School of Computer, Wuhan University in 2010. He is currently a Professor with the Shenzhen Institute for Advanced Study, University of Electronic Science and Technology of China. From 2016 to 2017, he was a visiting professor at with the Department of Electrical and Computer Engineering, Technical University of Munich, Germany. His main research interests include deep learning for Internet of Things. He is an author of over 150 peer-reviewed research papers and books, including over 40 IEEE/ACM Transactions papers such as TII, TITS, TOIT, TNSE, TMM, TCSS, TOMM, TETCI, PR, etc., and many top conference papers in the fields of edge intelligence.

Ammar Hawbani is an associate professor of networking and communication algorithms in the School of Computer Science and Technology at USTC. He received the B.S., M.S. and Ph.D. degrees in Computer Software and Theory from USTC, in 2009, 2012 and 2016, respectively. From 2016 to 2019, he worked as Postdoctoral Researcher in the School of Computer Science and Technology at USTC. His research interests include IoT, WSNs, WBANs, WMNs, VANETs, and SDN. Letters.

Ahmed Y. Al-Dubai is Professor of Networking and Communication Algorithms in the School of Computing at Edinburgh Napier University, UK. He received the PhD degree in Computing from the University of Glasgow in 2004. His research interests include Communication Algorithms, Mobile Communication, Internet of Things, and Future Internet. He received several international awards.

Geyong Min is a Professor of High Performance Computing and Networking in the Department of Computer Science within the College of Engineering, Mathematics and Physical Sciences at the University of Exeter, United Kingdom. He received the PhD degree in Computing Science from the University of Glasgow, United Kingdom, in 2003, and the B.Sc. degree in Computer Science from Huazhong University of Science and Technology, China, in 1995. His research interests include Computer Networks, Wireless Communications, Parallel and Distributed Computing, Ubiquitous Computing, Multimedia Systems, MPeter Nicol Russell Chair Professor of Computer Science

Albert Y. ZOMAYA is the Peter Nicol Russell Chair Professor of Computer Science and Director of the Centre for Distributed and High-Performance Computing at the University of Sydney. To date, he has published $>$ 700 scientific papers and articles and is (co-)author/editor of$>$30 books. A sought-after speaker, he has delivered > 250 keynote addresses, invited seminars, and media briefings. His research interests span several areas in parallel and distributed computing and complex systems. He is currently the Editor in Chief of the ACM Computing Surveys and served in the past as Editor in Chief of the IEEE Transactions on Computers (2010-2014) and the IEEE Transactions on Sustainable Computing (2016-2020).