Slicing-Based Task Offloading in Space-Air-Ground Integrated Vehicular Networks

Hang Shen, *Member, IEEE,* Yibo Tian, Tianjing Wang, *Member, IEEE,* and Guangwei Bai

Abstract—A slicing-based collaborative task offloading framework for space-air-ground integrated vehicular networks is proposed in this study, which can provide differentiated quality-of-service (QoS) guarantees for task offloading for high-speed vehicles while maximizing the number of completed tasks. A service-oriented radio access network (RAN) slicing framework is presented that supports slicing window adaptation, spectrum and computing resource orchestration, and collaboration among heterogeneous base stations. Based on the queuing model, the collaborative decision-making of RAN slicing and task offloading is modeled as a problem of maximizing the number of long-term task completions, which consists of three subproblems—slicing window division, resource slicing, and task scheduling—which are solved by a multi-access edge computing (MEC)-enabled controller, forming a closed loop with the slicing window as the period. When a new slicing window arrives, the controller determines its duration according to task traffic fluctuations and allocates resources to RAN slices through an optimization method. A double deep Q-learning network (DDQN)-based algorithm is developed for scheduling workflow on small time scales within a slicing window. Simulation results demonstrate that the proposed scheme performs better than existing approaches in terms of adaptability, task completion rate, and control overhead.

Index Terms—Space-air-ground integrated vehicular networks, Slicing window adaptation, RAN slicing, Task scheduling, Deep reinforcement learning.

1 Introduction

The characteristics of the fifth-generation (5G) networks, such as high bandwidth, millisecond-level delay, and ultra-high-density connections, facilitate the development of Internet of vehicles (IoV), which connects vehicles, base stations (BSs), and service providers as a collaborative system and realizes the real-time acquisition of comprehensive information [1]. In-vehicle devices have limited computing and storage capabilities, and therefore do not meet the requirements of high-complexity, data-intensive, and delay-sensitive applications. A feasible solution is the multi-access edge computing (MEC) paradigm [2], by which the computation tasks released by vehicles are offloaded to MEC servers on BSs for processing, and the computation results are sent back to vehicles, thereby supporting low-delay and high-efficiency vehicle services. However, there are issues related to terrestrial RANs, such as limited coverage, rigid network structure, and slow service response [3]. The high mobility of vehicles, complex urban road conditions, and diverse task requirements exacerbate the difficulty of task offloading and resource provisioning.

Space-air-ground integrated vehicular networks (SAGVNs) as promising networking architecture, utilize ground-based networks, supported by air- and space-based networks, which can provide seamless, comprehensive information services for vehicles and meet all-time and all-domain service needs [4]. The ground-based network is composed of cellular BSs, which offer services to areas with heavy traffic of people and vehicles. The air-based network consists of drone BSs, with mobile deployment and line-of-sight (LoS) advantages. The space-based network consists of low earth-orbit (LEO) satellites, and is a crucial structure to achieve global coverage and universal connectivity. Both drones and LEO satellites can serve as MEC platforms [5], [6], providing network access and task offloading for vehicles at the edge of the ground-based network, and areas with poor infrastructure.

With the development of intelligent transportation and autonomous driving, more and more in-vehicle applications are being developed, which are either delay-sensitive (e.g., route planning [7] and collision warning [8]) or delay-tolerant (e.g., high-definition (HD) map downloads [9]). Network slicing technology [10] can divide a physical RAN into multiple isolated virtual networks (i.e., RAN slices), to provide customized services for different applications. RAN slicing can provide differentiated quality-of-service (QoS) for IoV task offloading. The MEC-enbaled controller allocates computing and communication resources for a RAN slice based on information such as task traffic. In the sliced IoV, the offloading strategy decides where to offload tasks based on task attributes, BS loads, vehicle speeds, and routes. A natural step in the evolution of SAGVNs is to extend RAN slicing from ground-based networks to air- and space-based networks to support diverse IoV applications.

1.1 Challenging Issues and Related Works

SAGVN is a dynamic architecture with the features of multi-network integration and high vehicle speed, which bring many challenges to RAN slicing and task offloading:

1) Dynamic adjustment of slicing window. This is a fundamental problem, and the key to balancing overhead and

• *Hang Shen, Yibo Tian, Tianjing Wang, and Guangwei Bai are with the College of Computer and Information Engineering, Nanjing Tech University, Nanjing, 211816, China. E-Mail: {hshen, tianyb97, wangtianjing, bai}@njtech.edu.cn*

Manuscript received 23 Nov. 2022; revised 28 Mar. 2023; accepted 2 Jun. 2023. Date of publication xx, xxxx, 2023; date of current version xx, xx, xxxx. (Corresponding author: Hang Shen.)
Digital Object Identifier no. xxx

QoS. Due to the dynamic nature of the network and the time-varying task traffic, the service provision capability of slices will gradually weaken over time. The MEC controller must periodically reallocate resources to RAN slices. If the slicing window is too short, resource reallocation will be triggered frequently, which brings huge control and computing costs. However, if the slicing window is too long, the fluctuation of task traffic may lead to the destruction of slice performance isolation. Zhang et al. proposed a dynamic RAN slicing framework for IoV, which divides time into multiple equal-length slicing windows, where the optimal resource allocation strategy is calculated for each window [11]. Li et al. proposed a hierarchical soft RAN slicing framework for differentiated service provisioning, which conducts network-level and BS-level resource slicing on both large and small timescales [12]. In the above methods, resources are allocated in a fixed slicing window.

2) Multi-dimensional resource orchestration for multi-tier networks. The traffic in a road network is unevenly distributed in both time and space. There are generally large differences in the deployment, coverage, and resources for heterogeneous BSs. The coupling of resources in heterogeneous networks exacerbates the complexity of decision-making. Most studies have considered only terrestrial networks or a single type of resource slicing. Ye et al. proposed a downlink spectrum resource slicing framework for heterogeneous wireless networks, which achieved differentiated QoS provisioning for machine-type devices and end devices [13]. Peng et al. incorporated a transmit power adjustment mechanism and designed a spectrum slicing strategy based on multi-access edge computing [14]. A spectrum and computing resource slicing framework was proposed by Wu et al. to meet the requirements of task offloading of differentiated QoS in IoV [15]. Peng et al. combined a deep deterministic policy gradient (DDPG) and hierarchical learning to achieve multidimensional resource allocation in vehicular networks [16]. Li et al. presented a resource allocation framework for terrestrial-satellite networks, integrating a multi-agent DDPG algorithm to allocate resources and deploy cache equipment for maximum energy efficiency [17].

3) Collaboration among heterogeneous BSs. The interaction between a high-speed vehicle and a BS is instantaneous and is affected by vehicle speed, direction, and road conditions. The collaboration among air-ground, space-ground, and air-space BSs helps to reduce delay and facilitate task completions. Traditional model optimization and heuristic methods [18], [19], [20] cannot deal with real-time task offloading in dynamic scenarios. By integrating the decision-making advantages of reinforcement learning (RL) and the perceptual benefits of deep learning (DL), deep reinforcement learning (DRL) [21] allows individuals to perceive the environment and act accordingly, to deal with high-dimensional state-action spaces. Apostolopoulos et al. proposed a drone-Assisted framework for making data offloading decisions, allowing users to offload their data to ground or drone-mounted MEC servers [22]. The optimal offloading for each user was formulated as a maximization problem of their satisfaction and treated as a non-cooperative game. Most existing studies on BS collaboration in IoV consider the ground network. Kai et al. proposed a pipeline-based task offloading method by which mobile devices can offload tasks to edge nodes or the cloud according to their computing and communication capabilities [23]. Bai et al. investigated a delay minimization problem for multi-UAV-enabled edge-cloud cooperative offloading [24]. The problem was formulated as a non-convex problem considering network congestion, air-to-ground channels, and cooperative computing. Li et al. proposed a DRL-assisted task division and scheduling algorithm to maintain service continuity by preselecting edge servers, and reduce computing delays through edge-side collaboration [25]. Based on the multi-armed bandit theory, the online and off-policy learning approaches were presented in [26] to predict the offloading latency and select the least congested network. Wang et al. proposed an imitation learning-based task scheduling algorithm to minimize energy consumption under the task latency constraint of vehicular networks [27]. By combing actor-critic (A3C) and deep Q-network (DQN), Dai et al. developed an asynchronous task offloading algorithm to achieve fast convergence in an asynchronous way [28].

1.2 Contributions and Organization

In view of the above challenges, we propose a slicing-based collaborative task offloading framework for SAGVNs, which maximizes the number of completed tasks with differentiated QoS provisioning for task offloading. The main contributions of the study are three folded:

- A service-oriented RAN slicing framework is presented, which supports adaptive slicing window duration, multidimensional resource orchestration, and collaborative task offloading. Based on the queuing model, the decision-making of RAN slicing and task offloading is modeled as an optimization problem to maximize the number of long-term task completions under coupling and resource constraints;
- To balance QoS and overhead, an adaptive strategy for slicing window duration is proposed. During peak traffic hours, the slicing window length is reduced to facilitate resource reallocation. During off-peak periods, the slicing window length is increased to reduce overhead. For each window, an optimization method is applied to solve the spectrum and computing resource allocation problem for slices;
- A task scheduling approach based on the double deep Q-learning network (DDQN) is developed to determine task distribution under small timescales among heterogeneous BSs, where vehicle speed, driving direction, BS workloads, and task type are considered. In simulations, the proposed scheme outperforms existing methods in terms of adaptability, resource utilization, and task completion rate.

The remainder of this paper is organized as follows. Section 2 presents the RAN slicing framework, communication model, and task scheduling framework. In Section 3, the joint optimization of RAN slicing and task scheduling is modeled as a constrained stochastic optimization problem. Section 4 presents the solutions to each subproblem and proposes a joint optimization framework. Section 5 describes simulation experiments for performance evaluation. Section 6 summarizes the study and discusses future prospects. The main notations and variables are listed in Table 1.

TABLE 1
Main Notations and Variables

Symbols	Definition	Symbols	Definition
$a_{m,i,j}$	0-1 variable for establishing an upload connection	$r_{j',i,m}$	Downlink transmission rate for task m's result
$a^{(\ell)}$	Workflow scheduling action at epoch ℓ	$r^{(\ell)}$	Reward given by the environment at epoch ℓ
$\mathcal{A}^{(w)}$	Set of scheduling strategies in window w	$\mathcal{R}_{t,o}$	Set of task receptions of type o in time slot t
$\mathcal{A}_t$	Set of scheduling strategies in time slot t	s_j	Num. of VM instances held by BS j
$b_{m,i,j'}$	0-1 variable for transferring task m to BS j'	$s_{j,o}$	Num. of VM instances allocated to slice o at BS j
$\mathcal{B}$	Set of of ground BS indexes	$s^{(\ell)}$	Environment state for epoch ℓ
c_j	Num. of subchannels held by BS j	$\mathcal{S}^{(w)}$	Set of computing resource allocation strategies
$c_{j,o}$	Num. of subchannels allocated to slice o from c_j	$\mathcal{T}^{(w)}$	Set of scheduling slots in slicing window w
$\mathcal{C}^{(w)}$	Set of subchannel allocation strategies	$U^{(w)}$	Average reward of the system
d_m	Total service delay of task m	$\mathcal{W}/W$	Set/Num. of slicing windows
$\hat{d}_m$	Estimated reception time for the result of task m	y_m	Num. of subchannels allocated to task m
e_m	0-1 variable for the result return of task m	z_m	Num. of subchannels allocated to task m's result
$f^{(w)}$	Duration of slicing window w	$\lambda_{i,o}$	Arrival rate of type o tasks in vehicle i
$H^{(w)}$	Average loss due to incomplete tasks	ε_m	Data size of task m
$\mathcal{I}$	Set of vehicle indexes	τ_m	Required num. of VM instances of task m
$\mathcal{K}$	Set of drone indexes	ι_m	Computation result size of task m
$\mathcal{L}$	Set of satellite indexes	ν_m	Delay constraint of task m
$\mathcal{M}_{j,o}^{(w)}/M_{j,o}^{(w)}$	Set/Num. of type o tasks collected by BS j	$\rho_o^{(w)}$	Service intensity of offloading queue o
$r_{m,i,j}$	Uplink transmission rate of task m	μ_o	Average time for tasks of type o

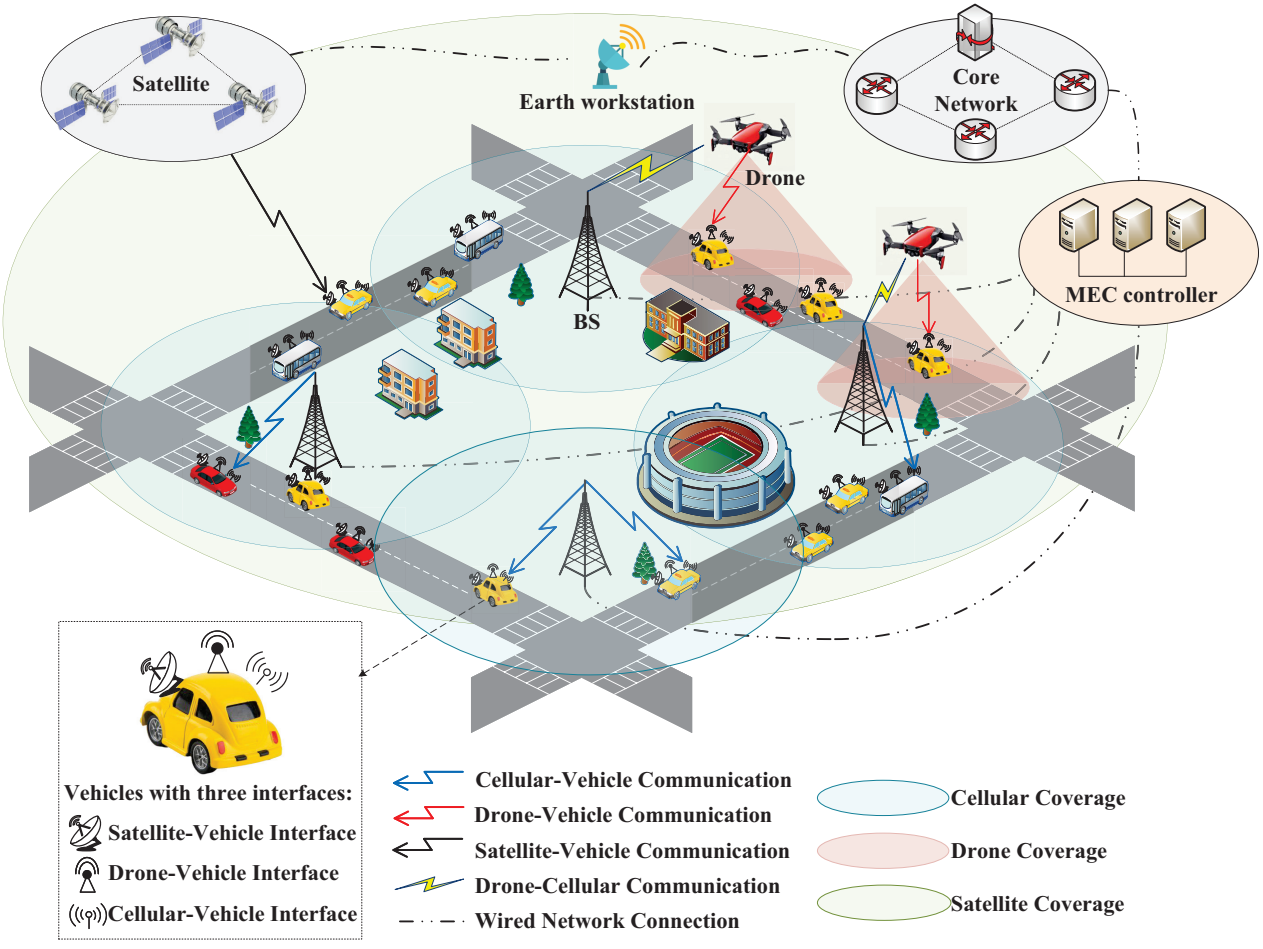


Fig. 1. SAGVN scenario.

2 SAGVN MODEL

Fig. 1 shows an SAGVN composed of an LEO satellite constellation, ground BSs, and drones. Ground BSs and drones have limited coverage, whereas LEO satellites can seamlessly cover the entire road network. The vehicles are equipped with three signal transceivers that can connect to satellites, ground BSs, and drones, while only one can be connected in a single time slot. The satellites are connected to the core network through the ground BSs. Drones that support task processing are pre-deployed, and positions can be adjusted as needed. They can connect to the ground BSs through a line-of-sight link and interact with the core network via ground BSs. An MEC-enabled controller is connected to different types of BSs through wireless relays or the core network and is responsible for multidimensional resource allocation on the RAN side and task scheduling among heterogeneous BSs.

2.1 RAN Slicing Framework

A service-oriented RAN slicing framework is proposed for task offloading, as shown in Fig. 2. The physical resources of each satellite, ground and drone BS are orchestrated into two service slices 1 and 2, for delay-sensitive and delay-tolerant tasks. The sets of satellite, ground, and drone BSs are denoted as $\mathcal{L}$, $\mathcal{B}$, and $\mathcal{K}$, respectively. The length of the slicing window can be adaptively adjusted according to network situations (details in Section 4.2). The time domain is divided into a series of slicing windows of different lengths, each containing multiple scheduling slots of equal length. The duration of slicing window w is denoted as $f^{(w)}$, with a set of scheduling slots denoted as $\mathcal{T}^{(w)}$. The spectrum and computing resources are allocated in units of subchannels and virtual machine (VM) instances. The number of subchannels and VM instances held by BS j are denoted as c_j and s_j. At the beginning of slicing window w, the resources of each BS are sliced according to task scheduling decisions in window $w-1$. Let $c_{j,o}$ and $s_{j,o}$ denote the number of the subchannels and VM instances allocated to slice o at BS j (with $c_j = \sum_{o\in\{1,2\}} c_{j,o}^{(w)}$, and $s_j = \sum_{o\in\{1,2\}} s_{j,o}^{(w)}$). The resource slicing strategy continues until the end of slicing window w. At the beginning of each scheduling slot in $\mathcal{T}^{(w)}$, the controller transfers the collected tasks to appropriate BSs for processing. BSs allocate resources for received tasks and transmit the computation results back to the original vehicle. At the end of each slicing window, the task scheduling decisions in this window are collected for the next window.

2.2 Communication Model

Since a long distance separates a satellite and a vehicle, the influence of vehicle movement on vehicle-to-satellite channel gain can be neglected in a small area. The average channel gain of vehicle i within the coverage of BS $j \in \mathcal{L}\cup\mathcal{B}\cup\mathcal{K}$ is denoted as $g_{i,j}$, which is quantified using the method described by Erceg et al. [29].

The transmit powers of vehicle i and BS j are denoted as p_i and p_j, respectively. During communication with BS j, other BSs can interfere with vehicle i. Spectrum resources in a slice are allocated to each vehicle in units of mutually orthogonal subchannels [30], [31]. Assume that the bandwidth of each subchannel is h. Let y_m denote the number of subchannels allocated to task m. The uplink transmission rate when vehicle i submits task m to BS j is calculated as

$$r_{m,i,j} = y_m h \log_2\left(1 + \frac{p_i g_{i,j}}{\sum\limits_{j'\in\mathcal{L}\cup\mathcal{B}\cup\mathcal{K}\setminus\{j\}} p_{j'} g_{i,j'} + \sigma^2}\right) \quad (1)$$

where σ is the average background noise. Let z_m denote the number of subchannels allocated to the computation result of task m. The downlink transmission rate of transferring the computational result of task m from BS j' to vehicle i is calculated as

$$r_{j',i,m} = z_m h \log_2\left(1 + \frac{p_{j'} g_{i,j}}{\sum\limits_{j\in\mathcal{L}\cup\mathcal{B}\cup\mathcal{K}\setminus\{j'\}} p_j g_{i,j} + \sigma^2}\right). \quad (2)$$

2.3 Task Scheduling Framework

A BS-collaborative framework is designed to take into account the high-speed movement of vehicles. Task offloading and processing no longer rely on a single BS but allow execution at two different BSs. Each BS has processing queues 1 and 2, to buffer delay-sensitive and delay-tolerant tasks, respectively. Similarly, the MEC controller has offloading queues 1 and 2 to buffer the two types of tasks received from heterogeneous BSs. According to network situations, these tasks are transferred to different BSs for collaborative processing. Two examples are described below.

1) Scheduling of delay-sensitive tasks: In Fig. 3(a), a vehicle is located within the coverage of the satellite and ground BS 1 when a task is generated. According to the principle of proximity, the task is collected by ground BS 1 and transferred to offloading queue 1 of the MEC controller. According to the direction and speed of the vehicle, the satellite and drone are selected as candidate collaborative BSs. Due to the low latency requirement, the controller selects the drone, which has a low load, to process the task, where the drone follows the first-come-first-serve (FCFS) rule to allocate resources for the task and transmit the processed results back to the vehicle.
2) Scheduling of delay-tolerant tasks: In Fig. 3(b), a vehicle is within the coverage of the satellite, drone, and ground BS 2. The drone receives the generated task and transfers it to offloading queue 2 of the controller. Based on the speed and direction of the vehicle, the satellites or ground BS 2 are listed as candidate collaborators. The controller selects the satellite with a low load to process the task.

From the above examples, task scheduling should consider vehicle speed, driving direction, and BS workloads.

We next derive task service delay based on queuing theory. Each computation task is characterized by four parameters $\{\varepsilon_m, \tau_m, \iota_m, \nu_m\}$ extending from [15], [32], where ε_m, τ_m, ι_m, and ν_m denote the task data size, the required number of VM instances, task computation result size, and delay constraint of task m.

2.3.1 Offloading Delay

The offloading delay (e.g., step ① in Figs. 3(a) and (b)) refers to the time taken from a task being uploaded by the

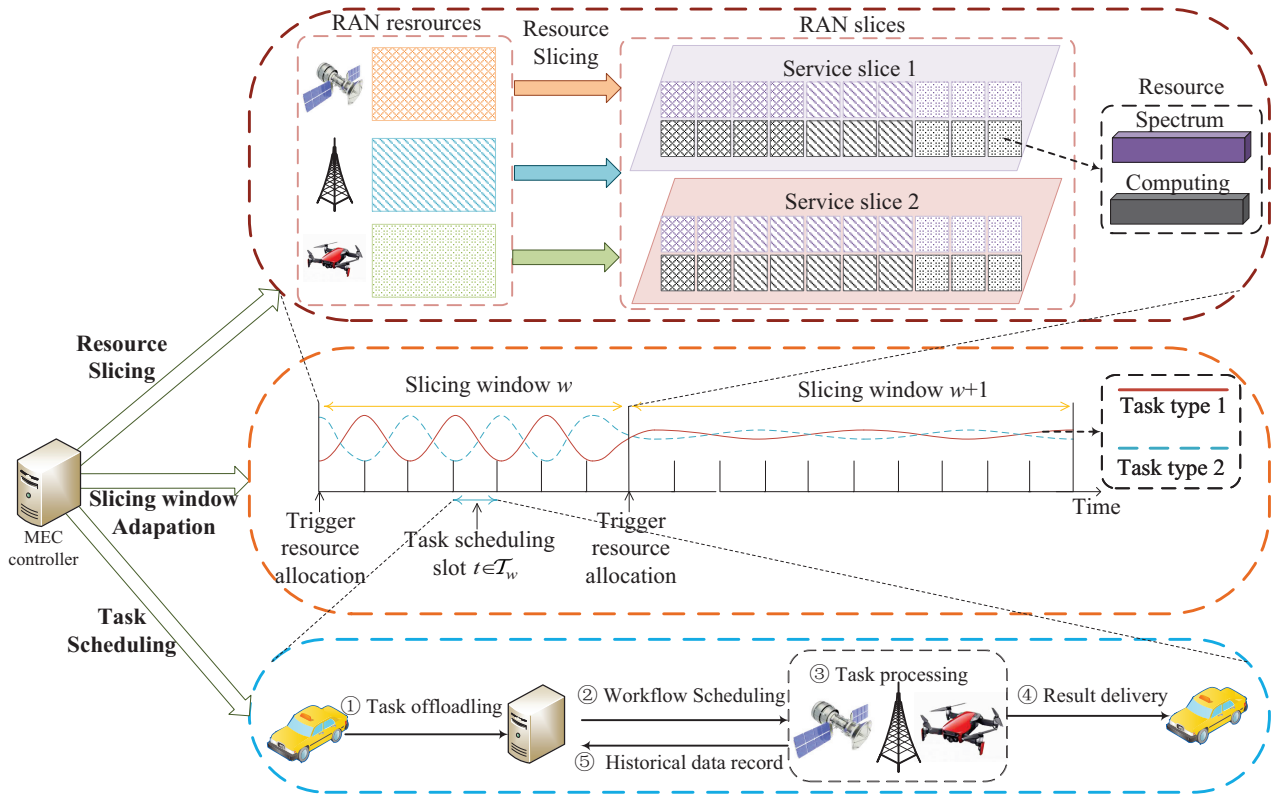


Fig. 2. RAN slicing framework for vehicle task offloading.

receiving BS to the offloading queue at the controller and transferred to the processing queue at the cooperating BS.

Denoted by $\mathcal{I}$ the vehicle set. The set of type o tasks collected by BS j is denoted as $\mathcal{M}_{j,o}^{(w)}$ with $M_{j,o}^{(w)}$ being its cardinality, in which $o = 1$ and $o = 2$ represent delay-sensitive and delay-tolerant task types. Let $a_{m,i,j} = 1$ represent that vehicle i and BS j establishes an uploading connection for task m, and otherwise, $a_{i,m,j} = 0$. According to (1), the average delay to upload a type o task from a vehicle to a BS is

$$\mu_o^{(w)} = \frac{\sum_{j\in\mathcal{L}\cup\mathcal{B}\cup\mathcal{K}} \sum_{m\in\mathcal{M}_{j,o}^{(w)}} \sum_{i\in\mathcal{I}} (\varepsilon_m a_{m,i,j}/r_{m,i,j})}{\sum_{j\in\mathcal{L}\cup\mathcal{B}\cup\mathcal{K}} M_{j,o}^{(w)}}. \quad (3)$$

The task arrivals for individual vehicles and BSs are modeled as Poisson processes as in [32]. Denote $\lambda_{i,o}^{(w)}$ as the arrival rate of type o tasks at vehicle i. The task arrival rate of offloading queue o in the controller is expressed as

$$\lambda_o^{(w)} = \sum_{j\in\mathcal{L}\cup\mathcal{B}\cup\mathcal{K}} \sum_{m\in\mathcal{M}_{j,o}^{(w)}} \sum_{i\in\mathcal{I}} a_{m,i,j}\lambda_{i,o}^{(w)}. \quad (4)$$

Only one task is processed at a time. The task offloading process is modeled as an M/M/1 queue. The service intensity of offloading queue o is defined as

$$\rho_o^{(w)} \triangleq \lambda_o^{(w)}\mu_o^{(w)}. \quad (5)$$

Enqueueing is determined by task arrival, and dequeuing by task assignment. When the enqueue rate is greater than the dequeue rate, the accumulation of tasks may cause an overflow. To ensure queue stability, (5) must satisfy

$$\rho_o^{(w)} < 1, \forall o \in \{1, 2\}. \quad (6)$$

Denote $\Omega(m)$ as the set of tasks queued before task m. Then, the offloading delay of task m is calculated as

$$d_m^{(1)} = \sum_{i\in\mathcal{I}} \sum_{j\in\mathcal{L}\cup\mathcal{B}\cup\mathcal{K}} \sum_{m'\in\{\Omega(m),m\}} \frac{a_{m,i,j}\varepsilon_{m'}}{r_{m',i,j}} + \zeta_m \quad (7)$$

where ζ_m is the delay for submitting task m via the receiving BS and forwarding it via the controller.

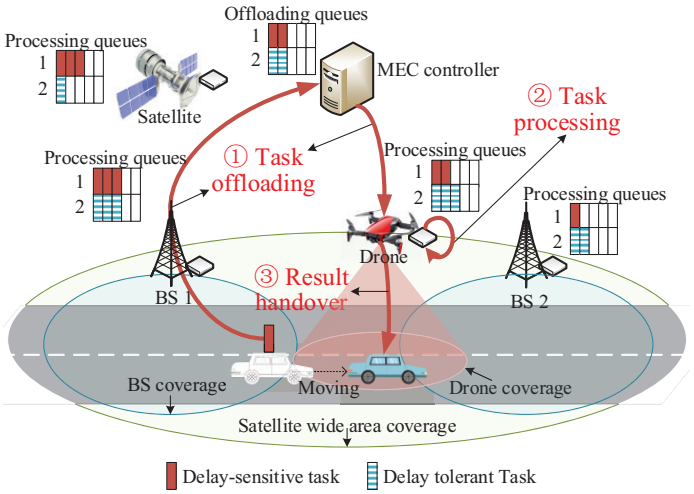
2.3.2 Processing Delay

The processing delay (e.g., step ② in Figs. 3(a) and (b)) is the time from when a task enters a processing queue at a collaborative BS to when the task processing is completed.

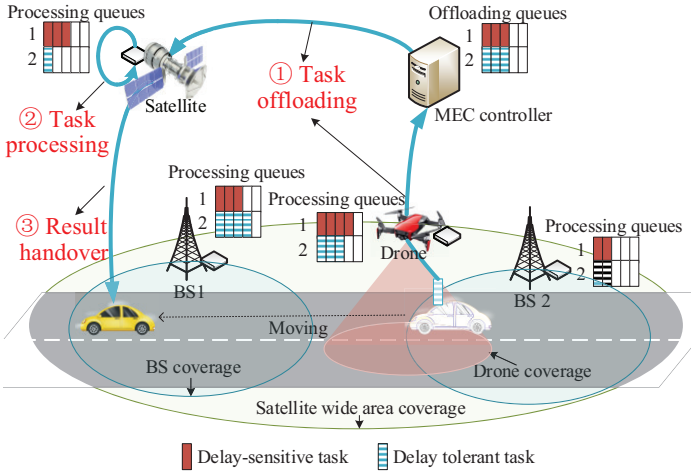
The BS allocates VM instances for each task as needed. Assume that the maximum CPU cycle of each VM instance is $\tau^{(\max)}$Hz per second. If the number of VM instances allocated by BS j for task m is s_m, the average processing time of the tasks in processing queue o in window w is calculated as

$$\mu_{j',o}^{(w)} = \frac{1}{M_{j',o}^{(w)}} \sum_{m\in\mathcal{M}_{j',o}^{(w)}} \frac{\tau_m}{s_m\tau^{(\max)}}. \quad (8)$$

The tasks in the offloading queues in the controller are distributed to the processing queues of different BSs. Let $b_{m,i,j'}$ be 1 if the controller transfers the task m generated by vehicle i to BS j' for collaborative processing, and otherwise be 0. In slicing window w, the proportion of tasks assigned



(a) Delay-sensitive task scheduling (case-1).



(b) Delay-tolerant task scheduling (case-2).

Fig. 3. Examples of collaborative task scheduling.

by the controller to BS j is expressed as

$$\eta_{j',o}^{(w)} = \frac{\sum_{i\in\mathcal{I}} \sum_{m\in\mathcal{M}_{j',o}^{(w)}} b_{m,i,j'}}{\sum_{i\in\mathcal{I}} \sum_{m\in\mathcal{M}_{j',o}^{(w)}} \sum_{j'\in\mathcal{L}\cup\mathcal{B}\cup\mathcal{K}} b_{m,i,j'}}. \tag{9}$$

The arrival of tasks in processing queue o at BS j' is also assumed to follow a Poisson process, where the task arrival rate j' is $\eta_{j',o}^{(w)}\lambda_o^{(w)}$. The task processing is modeled as an M/M/1 queue. Based on (4), (8), and (9), the service intensity of processing queue o in the BS is defined as

$$\rho_{j',o}^{(w)} \triangleq \eta_{j',o}^{(w)}\lambda_o^{(w)}\mu_{j',o}^{(w)}. \tag{10}$$

Similar to (6), (10) must satisfy

$$\rho_{j',o}^{(w)} < 1, \forall j' \in \mathcal{L}\cup\mathcal{B}\cup\mathcal{K}, o\in\{1,2\}. \tag{11}$$

At BS j', the set of tasks queued before task m is denoted as $\Psi_{j'}(m)$, and the processing delay of task m is calculated as

$$d_m^{(2)} = \sum_{i\in\mathcal{I}} \sum_{m'\in\{\Psi_{j'}(m),m\}} \sum_{j'\in\mathcal{L}\cup\mathcal{B}\cup\mathcal{K}} \frac{b_{m,i,j'}\tau_{m'}}{s_m\tau^{(\max)}}. \tag{12}$$

2.3.3 Handover Delay

After a task is processed at collaborative BS j, the BS sends the result back to the vehicle (e.g., step ③ in Figs. 3(a) and (b)). Based on (2), the delay for BS j' to hand over the computation result of task m to vehicle i is

$$d_m^{(3)} = \sum_{i\in\mathcal{I}} \sum_{j'\in\mathcal{L}\cup\mathcal{B}\cup\mathcal{K}} \frac{b_{m,i,j'}\iota_m}{r_{j',i,m}}. \tag{13}$$

Suppose that task m issued by vehicle i is assigned to be processed by BS j'. If the speed vector of vehicle i is $\vec{v}_i$ and the remaining driving distance to leaving the coverage of BS j' is $\omega_{i,j'}$, the remaining time for vehicle i to receive the computation result of task m is estimated as

$$\widehat{d}_m = \sum_{i\in\mathcal{I}} \sum_{j'\in\mathcal{L}\cup\mathcal{B}\cup\mathcal{K}} \frac{b_{m,i,j'}\omega_{i,j'}}{\vec{v}_i} + \varrho \tag{14}$$

where ϱ is an adjustable parameter related to special events. For instance, when vehicle i is about to encounter a red light, ϱ can be set to the duration of the red light.

Combining (14) and ν_m, the deadline of task m generated by vehicle i is rewritten as

$$\min\left\{\nu_m, \widehat{d}_m\right\}. \tag{15}$$

Factors such as driving direction variations may also cause its failed encounter with collaborative BS j'. In this case, even if the task is processed by BS j' within (15), the result cannot be transmitted back to vehicle i.

3 PROBLEM FORMUATION

In the proposed framework, a challenging problem is the joint optimization of slicing window division, resource orchestration, and task scheduling.

The total service delay of task m is the summation of (7), (12), and (13), i.e.,

$$d_m = d_m^{(1)} + d_m^{(2)} + d_m^{(3)}. \tag{16}$$

Based on (15) and (16), we define a binary variable,

$$e_m \triangleq \begin{cases} 1, \text{if } \min\left\{\nu_m, \widehat{d}_m\right\} \geq d_m \\ 0, \text{otherwise.} \end{cases} \tag{17}$$

where $e_m = 1$ if and only if the result of task m is successfully delivered within the specified time. In slicing window w, the average reward of the system for completing the tasks is defined as

$$U^{(w)} \triangleq \sum_{j'\in\mathcal{L}\cup\mathcal{B}\cup\mathcal{K}} \sum_{o\in\{1,2\}} \sum_{i\in\mathcal{I}} \sum_{m\in\mathcal{M}_{j',o}^{(w)}} u_{j',o} b_{m,i,j'} e_m \tag{18}$$

where $u_{j',o} \in (0,1)$ is the reward factor for the completion of type o tasks in collaborative BS j'. The average loss due to incomplete tasks is defined as

$$H^{(w)} \triangleq \sum_{j'\in\mathcal{L}\cup\mathcal{B}\cup\mathcal{K}} \sum_{o\in\{1,2\}} \sum_{i\in\mathcal{I}} \sum_{m\in\mathcal{M}_{j',o}^{(w)}} h_{j',o} b_{m,i,j'} (1-e_m) \tag{19}$$

where $h_{j',o} \in \{0,1\}$ is the loss factor for the failure to complete the tasks of type o in collaborative BS j'.

In slicing window w, the strategy sets of spectrum allocation, computing resource allocation are denoted by

$$\mathcal{C}^{(w)} = \left\{ c_{j,o}^{(w)} | o \in \{1,2\}, j \in \mathcal{L} \cup \mathcal{B} \cup \mathcal{K} \right\}$$

and

$$\mathcal{S}^{(w)} = \left\{ s_{j,o}^{(w)} | o \in \{1,2\}, j \in \mathcal{L} \cup \mathcal{B} \cup \mathcal{K} \right\}.$$

In time slot t, the set of task receptions of type o is denoted as $\mathcal{R}_{t,o}$, and the set of scheduling strategies is denoted as $\mathcal{A}_t = \{a_{m,i,j}, b_{m,i,j'} | i \in \mathcal{I}, o \in \{1,2\}, m \in \mathcal{R}_{t,o}, j, j' \in \mathcal{L} \cup \mathcal{B} \cup \mathcal{K}\}$. Given $\mathcal{T}^{(w)}$, the set of scheduling strategies in slicing window w is expressed as

$$\mathcal{A}^{(w)} = \bigcup\nolimits_{t \in \mathcal{T}^{(w)}} \mathcal{A}_t.$$

The slicing window index set and its set cardinality are denoted as $\mathcal{W}$ and W. The maximization of task completion under long-term accumulation is modeled as $\mathcal{P}1$.

$$\mathcal{P}1: \max_{\{f^{(w)}, \mathcal{A}^{(w)}, \mathcal{C}^{(w)}, \mathcal{S}^{(w)}\}_{w \in \mathcal{W}}} \lim_{W \to \infty} \sum_{w \in \mathcal{W}} \left(\frac{U^{(w)} - H^{(w)}}{f^{(w)}} \right)$$

$$\text{s.t.} \begin{cases} \sum_{o \in \{1,2\}} c_{j,o}^{(w)} = c_j, \forall j \in \mathcal{L} \cup \mathcal{B} \cup \mathcal{K} & \text{(20a)} \\ \sum_{o \in \{1,2\}} s_{j,o}^{(w)} = s_j, \forall j \in \mathcal{L} \cup \mathcal{B} \cup \mathcal{K} & \text{(20b)} \\ \sum_{m \in \mathcal{R}_{t,o}} \sum_{i \in \mathcal{I}} \sum_{j \in \mathcal{L} \cup \mathcal{B} \cup \mathcal{K}} (a_{m,i,j} y_m + b_{m,i,j} z_m) \le c_{j,o}^{(w)}, \forall o \in \{1,2\}, t \in \mathcal{T}^{(w)} & \text{(20c)} \\ \sum_{m \in \mathcal{R}_{t,o}} \sum_{i \in \mathcal{I}} \sum_{j \in \mathcal{L} \cup \mathcal{B} \cup \mathcal{K}} b_{m,i,j} s_m \le s_{j,o}^{(w)}, \forall o \in \{1,2\}, t \in \mathcal{T}^{(w)} & \text{(20d)} \\ \sum_{i \in \mathcal{I}} \sum_{j \in \mathcal{L} \cup \mathcal{B} \cup \mathcal{K}} a_{m,i,j} = 1, \forall m \in \mathcal{M}_{j,1}^{(w)} \cup \mathcal{M}_{j,2}^{(w)} & \text{(20e)} \\ \sum_{i \in \mathcal{I}} \sum_{j' \in \mathcal{L} \cup \mathcal{B} \cup \mathcal{K}} b_{m,i,j'} = 1, \forall m \in \mathcal{M}_{j,1}^{(w)} \cup \mathcal{M}_{j,2}^{(w)} & \text{(20f)} \\ (6) \text{ and } (11). & \text{(20g)} \end{cases}$$

The problem is to allocate the spectrum and computing resources to each slice, balance BS loads, and maximize the long-term average number of completed tasks. Constraints (20a) and (20b) demonstrate the requirements on bandwidth and computing resource allocation for slices 1 and 2 at each BS. The number of spectrum and computing resources allocated to tasks by each BS should not exceed the total number of resources held by itself, corresponding to constraints (20c) and (20d). Constraint (20e) means that each vehicle can only connect to a unique BS for task uploading, and (20f) ensures that each task is assigned to a unique BS for processing. Constraint (20g) aims to maintain the stability of each offloading/processing queue. Both resource allocation and task scheduling decisions affect queue stability.

The objective of $\mathcal{P}1$ is a long-term non-smooth maximum function. Constraints (20e) and (20f) contain two binary integer variables, and the variables in (20g) are coupled to each other. Therefore, it is difficult to obtain an exact optimal solution for $\mathcal{P}1$ under conventional optimization methods.

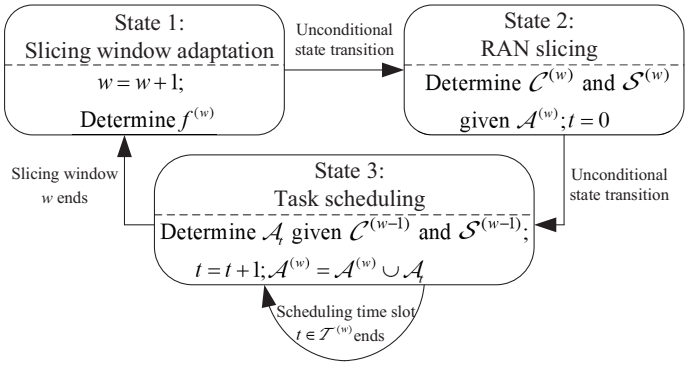


Fig. 4. State machine for the MEC controller.

4 SOLUTION

4.1 Problem-Solving Framework

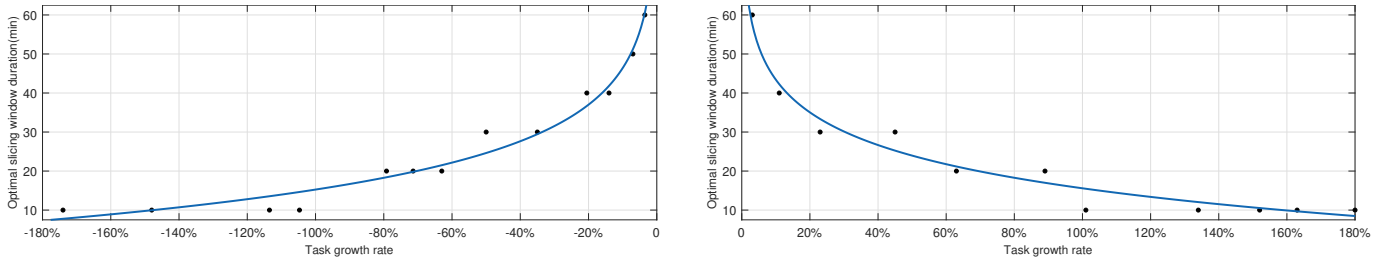
To facilitate processing, $\mathcal{P}1$ is decoupled into three subproblems: 1) adaptive slicing window division; 2) resource allocation (large timescale), and 3) task scheduling (small timescale). These are solved alternately by the MEC controller, forming a closed loop that runs continuously. As shown in Fig. 4, the behavior of the controller is abstracted as a state machine with three states, each corresponding to a subproblem-solution module. When the system reaches a certain state, the corresponding function module is activated. The operations of each state are described below.

- Slicing window adaptation (State 1): When slicing window $w-1$ ends, the controller determines the length $f^{(w)}$ of the slicing window according to task traffic fluctuation (details in Section 4.2);
- RAN slicing (State 2): After the length of slicing window w is determined, the task scheduling in window $w-1$ becomes a known condition for determining resource allocation with respect to $\mathcal{C}^{(w)}$ and $\mathcal{S}^{(w)}$ in window w (details in Section 4.3). Resource reallocation for each slice is made by the controller at the beginning of each slicing window, and remains unchanged until its end;
- Task scheduling (State 3): At the beginning of each scheduling slot in window w, $\mathcal{C}^{(w)}$ and $\mathcal{S}^{(w)}$ are fed into a DDQN algorithm to determine task scheduling (details in Section 4.4). At the end of the last slot in each slicing window, all scheduling decisions within this window are saved as $\mathcal{A}^{(w)}$, which are subsequently used to determine $\mathcal{C}^{(w+1)}$ and $\mathcal{S}^{(w+1)}$.

The detailed solutions to the three subproblems in the closed-loop framework are discussed below.

4.2 Slicing Window Division

In reality, the release of vehicle task requests is time-varying and uncertain. If resource allocation is performed under a fixed slicing window mode, the resource scheduling of RAN slices will be unable to cope with the fluctuation of task arrivals. During the peak traffic period, the proportions of various types of tasks will fluctuate continuously and significantly. At this time, reducing the slicing window length can promote resource redistribution and adapt to fluctuations in



(a) Case-1 ($x < 0, \alpha = -9.36, \beta = 77.5$)

(b) Case-2 ($x > 0, \alpha = -8.37, \beta = 71.25$)

Fig. 5. Fitting of task traffic fluctuation and optimal slicing window duration

task traffic. In off-peak periods, the proportion of different tasks is relatively stable [33], and the window length can be increased to reduce unnecessary overhead.

The optimal match between task traffic fluctuation and slicing window length was explored experimentally. The minimum adjustment interval of the slicing window length was 10 minutes. The system first had a preset short window length, which was then tentatively increased. Multiple initial time points were selected, and the numerical pairs of task traffic fluctuation and optimal slicing window length were collected, and fitted as

$$y = \alpha \log_2 x + \beta. \tag{21}$$

The fitting process is to find α and β that minimize the residual sum of squares. Two fitted curves were generated. Fig. 5(a) corresponds to the situation where the task traffic continues to decrease, in which the slicing window length gradually increases with the decrease of traffic. Fig. 5(b) shows the situation where the task traffic falls, in which the variation is the opposite of Fig. 5(a). It can be seen that the more severe the fluctuation of task traffic, the smaller the optimal slicing window length, which is as expected.

At the end of window $w-1$, the ARIMA-ANN Hybrid model [34] was used to predict the task traffic at the beginning of the next window w, denoting the predicted value as $\varpi^{(w)}$. The ARIMA and ANN models are suitable for processing linear and nonlinear historical data, and to integrate them can improve prediction accuracy. Based on (21) and $\varpi^{(w)}$, the duration of slicing window w is determined as

$$f^{(w)} = \begin{cases} \gamma \left\lfloor \alpha_1 \log_2(-x) + \beta_1 \right\rfloor, x \in (-\infty, 0) \\ \gamma \left\lceil \alpha_2 \log_2 x + \beta_2 \right\rceil, x \in (0, +\infty) \end{cases} \tag{22}$$

where $x = \left(\varpi^{(w)} - \varpi^{(w-1)}\right) / \varpi^{(w-1)}$, γ is a constant representing the smallest unit of slicing window length, and $\lceil \cdot \rceil$ and $\lfloor \cdot \rfloor$ denote rounding up and down.

4.3 Resource Slicing

The resource slicing subproblem is to maximize task completions by allocating spectrum and computing resources across RAN slices, i.e.,

$$\mathcal{P}1.1: \max_{\{\mathcal{C}^{(w)}, \mathcal{S}^{(w)}\}_{w \in \mathcal{W}}} \lim_{W \to \infty} \sum_{w \in \mathcal{W}} \left(\frac{U^{(w)} - H^{(w)}}{f^{(w)}} \right)$$
$$\text{s.t. } (20a), (20b), (20c), \text{and } (20d).$$

According to (18) and (19), the decision of each slicing window is independent, and the tasks within the window are independently allocated resources. In the real world, the task traffic does not fluctuate continuously in adjacent slicing windows. Based on $\mathcal{A}^{(w-1)}$, the controller can calculate the amount of spectrum and computing resources required for each slice in window w. Accordingly, $\mathcal{P}1.1$ is transformed to a one-shot optimization of maximizing task completions in window w as $\mathcal{P}1.1a$.

$$\mathcal{P}1.1a: \max_{\mathcal{C}^{(w)}} \frac{U^{(w)} - H^{(w)}}{f^{(w)}}$$
$$\text{s.t. } (20a), (20b), (20c), \text{and } (20d).$$

$\mathcal{P}1.1a$ belongs to a multi-constraint multivariate function extremum problem. A Lagrange multiplier was used to solve the problem by transforming a multivariate and multi-constraint optimization problem to a multivariate unconstrained extremum problem. Then $\mathcal{P}1.1a$ is converted to $\mathcal{P}1.1b$ by taking $\mathcal{C}^{(w)}$ and $\mathcal{S}^{(w)}$ as input parameters to the problem. The optimal resource allocation scheme for $\mathcal{P}1.1b$ can be obtained by gradient descent.

4.4 DDQN-Based Task Scheduling

The task scheduling subproblem is to maximize task completions by selecting collaborative BSs for tasks, i.e.,

$$\mathcal{P}1.2: \max_{\{\mathcal{A}_t\}_{t \in \mathcal{T}^{(w)}, w \in \mathcal{W}}} \lim_{W \to \infty} \sum_{w \in \mathcal{W}} \left(\frac{U^{(w)} - H^{(w)}}{f^{(w)}} \right)$$
$$\text{s.t. } (20e), (20f), \text{and } (20g).$$

As described in Section 4.3, the resource allocation in each slicing window is independent. When the resource allocation is determined, the task scheduling in each slicing window is also independent. Therefore, the long-term optimization problem in $\mathcal{P}1.2$ can be decomposed into a short-term optimization problem for a single slicing window, which is a Markov decision problem with a finite horizon.

The task scheduling subproblem within a slicing window is constructed as a Markov decision process (MDP). The MEC controller is abstracted as an agent, making workflow scheduling action $a^{(\ell)}$ according to $s^{(\ell)}$, the environment state for training epoch ℓ. The reward given by the environment is denoted as $r^{(\ell)}$. The controller updates the environment state to $s^{(\ell+1)}$ according to state transition probability $\Pr(s^{(\ell+1)} | s^{(\ell)}, a^{(\ell)})$. The expressions of the state space, action space, and reward are as follows.

- **State Space**: Task scheduling should consider real-time information about tasks, vehicles, slice workloads. Let l_i and $\varphi_{j,o}$ denote the position of vehicle i

$$\mathcal{P}1.1\text{b}: \max_{\{\mathcal{C}^{(w)},\mathcal{S}^{(w)}\}} F(\mathcal{C}^{(w)},\mathcal{S}^{(w)},\kappa_1,\kappa_2)$$

$$= (\frac{U^{(w)}-H^{(w)}}{f^{(w)}}) + \kappa_1 \sum_{t\in\mathcal{T}^{(w)}} \sum_{o\in\{1,2\}} \sum_{j\in\mathcal{L}\cup\mathcal{B}\cup\mathcal{K}} \left(c_{j,o} - \sum_{m\in\mathcal{R}_{t,o}} \sum_{i\in\mathcal{I}} \sum_{j\in\mathcal{L}\cup\mathcal{B}\cup\mathcal{K}} (a_{m,i,j}y_m + b_{m,i,j}z_m) \right)$$

$$+ \kappa_2 \sum_{t\in\mathcal{T}^{(w)}} \sum_{o\in\{1,2\}} \sum_{j\in\mathcal{L}\cup\mathcal{B}\cup\mathcal{K}} \left(s_{j,o} - \sum_{m\in\mathcal{R}_{t,o}} \sum_{i\in\mathcal{I}} \sum_{j\in\mathcal{L}\cup\mathcal{B}\cup\mathcal{K}} b_{m,i,j}s_m \right)$$

and the number of tasks in processing queue o at BS j. Then the state of training epoch ℓ is expressed as

$$s^{(\ell)} = \{\varepsilon_m, \tau_m, \iota_m, \nu_m\}_{o\in\{1,2\},t\in\mathcal{T}^{(w)},m\in\mathcal{R}_{t,o}} \cup \{l_i,\vec{v}_i\}_{i\in\mathcal{I}} \cup \{c_{j,o}^{(w)}, s_{j,o}^{(w)}, \varphi_{j,o}^{(w)}\}_{j\in\mathcal{L}\cup\mathcal{B}\cup\mathcal{K},o\in\{1,2\}}. \quad (23)$$

- **Action Space**: The workflow scheduling action made by the system in training epoch ℓ is

$$a^{(\ell)} = \mathcal{A}^{(\ell)} \quad (24)$$

where $\mathcal{A}^{(\ell)}$ is the set of task scheduling decisions in training epoch ℓ, i.e., the controller assigns a set of tasks to collaborative BSs. Under (20e) and (20f), the decision variable for each action is 0 or 1, which is determined by the current state;

- **Reward**: The reward reflects the pros and cons of actions performed in a certain state. The goal of the system is converted from maximizing the number of task completions to maximizing the reward. Based on (18) and (19), the reward is

$$r^{(\ell)}(s^{(\ell)},a^{(\ell)}) = U^{(\ell)} - H^{(\ell)} \quad (25)$$

where $U^{(\ell)}$ is the sum of the rewards obtained for completing the tasks in training epoch 1, and $H^{(\ell)}$ is the sum of the losses for task failures in training epoch ℓ. The task scheduling action decides which BSs cooperatively handle the tasks. If a task is completed, the environment will provide a reward to recognize the action. Meanwhile, a penalty mechanism is intoduced to prevent decisions that could cause a high BS load or destabilize the processing queue.

In the MDP, task scheduling refers to the process that the controller maximizes its rewards by allocating tasks in the offloading queue to different collaborative BSs,

$$\mathcal{P}1.2\text{a}: \max_{\pi\in\prod} \sum_{\ell} [\delta^{(\ell)} \cdot r^{(\ell)}(s^{(\ell)},a^{(\ell)})|\pi]$$

where $\prod$ is the set of all possible task allocation strategies, and $\delta^{(\ell)} \in (0,1)$ is the discount factor in epoch ℓ. Due to the unpredictability of request releases, state transitions are difficult to determine. The problem cannot be solved by model-based methods such as value iteration and strategy iteration [35]. A realistic solution is to use a model-free scheme that does not rely on state transition probabilities. However, due to the complexity of task scheduling in SAGVNs, traditional model-free RL algorithms are unable to process complex action and state spaces. The DQN algorithm, as an improvement of Q-learning, does not rely on prior knowledge and can adapt to large action-state spaces. DDQN separately trains the evaluation and target networks to avoid overestimation caused by bootstrapping. Therefore, a DDQN-based method is proposed for solving task scheduling on small time scales.

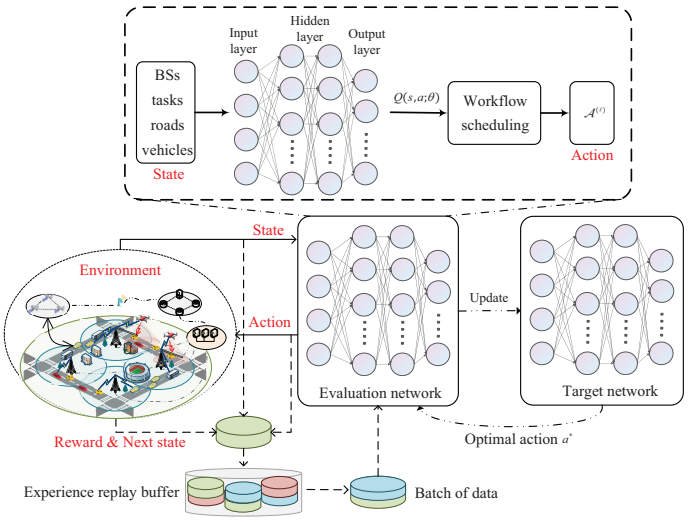


Fig. 6. DDQN structure for task scheduling.

In the state space, the reward of each action is estimated and stored into a Q-table. The action value function is denoted as $Q(s^{(\ell)},a^{(\ell)})$. The maximum reward for each state in the Q-table represents the maximum possible reward in the future. By querying the Q-table, the action with the maximum reward in each state is determined as

$$a^* = \arg\max_a Q^*(s^{(\ell)},a^{(\ell)}), a^{(\ell)} \in \pi. \quad (26)$$

Applying the Bellman equation to (26), the value in the Q-table can be obtained as

$$Q(s^{(\ell+1)},a^{(\ell+1)}) = Q(s^{(\ell)},a^{(\ell)}) + \phi\left[r^{(\ell)} + \upsilon \max_{a^*} Q(s^{(\ell+1)},a^{(\ell+1)}) - Q(s^{(\ell)},a^{(\ell)})\right] \quad (27)$$

where ϕ is the learning rate and υ is the greedy probability.

DDQN-based collaborative task scheduling is named as Algorithm 1. Compared with DQN, Algorithm 1 adds two modules: the experience replay pool and the target network. The experience replay mechanism builds a data pool, storing data obtained during the model and environment interaction in the form of $(s^{(\ell)},a^{(\ell)},r^{(\ell)},s^{(\ell+1)})$ as samples. The

Algorithm 1: DDQN-based task scheduling

input : $\{\varepsilon_m, \tau_m, \iota_m, \nu_m\}_{o\in\{1,2\}, t\in\mathcal{T}^{(w)}, m\in\mathcal{R}_{t,o}} \cup \{l_i, \vec{v}_i\}_{i\in\mathcal{I}} \cup \{c_{j,o}^{(w)}, s_{j,o}^{(w)}, \varphi_{j,o}^{(w)}\}_{j\in\mathcal{L}\cup\mathcal{B}\cup\mathcal{K}, o\in\{1,2\}}$

output: π^*

Initialize DDQN parameters and $Q(s^{(\ell)}, a^{(\ell)})$;
Initialize experience replay buffer;
Initialize evaluation network parameters by selecting random weight θ;
Initialize target network parameters by $\theta^- \leftarrow \theta$;
for $episode \leftarrow 1$ **to** $|\mathcal{T}^{(w)}|$ **do**
  Initialize $s^{(1)}$ by observing the environment;
  for $\ell \leftarrow 1$ **to** $\ell^{(\max)}$ **do**
    With a probability select a random action $a^{(\ell)}$, otherwise select $a^{(\ell)} \leftarrow \pi(s^{(\ell)})$;
    Execute $a^{(\ell)}$, observe $s^{(\ell+1)}$ and reward $r^{(\ell)}$;
    Store quadruple $(s^{(\ell)}, a^{(\ell)}, r^{(\ell)}, s^{(\ell+1)}$ in the experience replay buffer;
    if *experience replay buffer is not empty* **then**
      Sample a batch of quads from the buffer;
    if $\ell == \ell^{(\max)}$ **then**
      $Q(s^{(\ell)}, a^{(\ell)}|\theta) \leftarrow r^{(\ell)}$;
      Break;
    else
      $Q(s^{(t)}, a^{(t)}|\theta^-) \leftarrow r^{(\ell)} + \eta \cdot \max_{a^*} Q'(s^{(t+1)}, a^{*(t+1)}|\theta^-)$;
      Update the evaluation network parameters via gradient descent θ;
      $\pi(s^{(t)}) \leftarrow \arg\max_{a^*} Q(s^{(\ell)}, a^{*(\ell)}|\theta)$;
      $\theta^- \leftarrow \theta$ every k iterations;
return Strategy set π^* in slicing window w

samples are randomly selected from the stored memory data units to update the parameters of neural networks. Since $s^{(\ell)}$ and $s^{(\ell+1)}$ estimated by DQN are time-dependent, DQN suffers from overestimation bias with a reduced training effect. The experience replay pool in DDQN randomly selects samples during neural network training. Disrupting sample correlation and using data multiple times helps to make better task-offloading decisions than DQN. In addition, the evaluation network and target network have the same structure and asynchronous parameters, where θ is used to select an optimal action, and θ^- to evaluate the Q-value of the action. The single neural network in DQN determines both action selection and strategy evaluation. The training of this network relies on (26) for parameters update. From (26), continuing to estimate based on valuation will bring about overestimation. DDQN separates action selection and strategy evaluation from each other. The evaluation network is used to select the optimal action. After the k-step iterative calculation, the evaluation network weight (θ) is copied to the target network weight (θ^-) for evaluating the Q-value of the optimal action. By delaying parameter updates, DDQN reduces the correlation for evaluation and target networks, reducing the risk of overestimating Q-values.

TABLE 2
Default parameter settings

Parameters	Values
Background noise power (σ^2)	-110 dBm
Bandwidth of each subchannel (h)	180 kHz
CPU cycle of each VM instance ($\tau^{(\max)}$)	10 GHz
Number of subchannels held by each ground/drone/satellite BS	15/10/40
Number of VM instances held by each ground/drone/satellite BS	15/8/30
Deadline of delay-sensitive/-tolerant task	0.05-1s/3-10s
Arrival rate of delay-sensitive/-tolerant tasks at vehicle i ($\lambda_{i,1}/\lambda_{i,2}$)	4/20 req/s
Data size of delay-sensitive/-tolerant tasks	2000/9000bits
Computation result size of task m (ι_m)	1000-5000bits

TABLE 3
Implementation of Baseline Approaches

Baseline approach	Slicing window	Resource allocation	Task Scheduling
Baseline-1	Static [15], [36]	Section 4.2	Section 4.3
Baseline-2	Static [15], [36]	Section 4.2	DQN [37]
Baseline-3	Section 4.1	Section 4.2	DQN [37]
Baseline-4	Section 4.1	[16], [36]	Max-SINR [38]
Baseline-5	Static [15], [36]	[16], [36]	Max-SINR [38]

4.5 Computational Complexity Analysis

The advantage of the proposed DDQN-based algorithm is that each time slot can process a batch of computing task requests simultaneously, with improved processing speed and environmental adaptability. In this subsection, we analyze the complexity of Algorithm 1 by comparing it with the DQN algorithm. Assume that the computational complexity of DQN training N training episodes is $\mathcal{O}(N)$. DDQN adds a target network based on DQN to reduce the negative impact of data correlation. However, this neural network does not require additional training, and its weight parameters are copied from the evaluation network every k-step. a^* is selected by the evaluation network, and $Q(s, a^*)$ is obtained from the target network. In this way, the action selection and the strategy evaluation operations are separated to reduce data correlation. Compared with DQN, the proposed DDQN-based algorithm improves the rationality of decision-making with almost no increase in the cost of model training. Accordingly, the complexity of Algorithm 1 can be approximated as $\mathcal{O}((1 + 1/k)N)$.

5 PERFORMANCE EVALUATION

Simulations were carried out to verify the effectiveness and superiority of the proposed method. For a four-lane highway that is 1000 meters long; the origin was set at the starting point of the highway. The scenario contained two ground BSs, three drones, and one satellite. The satellite covered the entire highway, and the two ground BSs each covered about 500m. The drones hovered above the highway at an altitude of 120m, with an effective coverage radius of 80m. The satellite, ground BS, and drone have the transmit powers of 27w, 40dBm, and 0.1w, respectively.

The traffic flow trace was selected from OpenITS [1], an open road network traffic data platform. The vehicle density was set to 0.4 vehicles/m^2. Autonomous vehicle platooning and HD map downloading simulate delay-sensitive and delay-tolerant tasks. The CPU of the model training platform is AMD Ryzen5 3500X with six cores and six threads, and the graphics card is NVIDIA GeForce GTX 1660 SUPER. Default simulation parameters are shown in Table 2, where the deadlines of latency-sensitive and latency-tolerant tasks and the computation result size are randomly generated in the given ranges.

Four methods were selected as baselines, each including three functional modules: slicing window adjustment, resource allocation, and task scheduling. Table 3 presents the implementation details of each baseline.

5.1 Effect of Training Epochs

We evaluated the effect of the number of training epochs on performance. The DRL-based algorithms used the same settings with 4000 pieces of data. In DRL, the learning speed and training effect are affected by the update period and learning rate. The agent's propensity for long- and short-term rewards is affected by the discount rate in the cumulative discounted reward. In the simulation regarding Fig. 7(a), we studied the reward and convergence of the proposed method with initial learning rates of 0.1, 0.005, and 0.001. Model training was performed offline. The training time was similar, taking about 12 minutes to execute 100 training episodes. There was a proportional relationship between the reward and the number of completed tasks. In the first 20 training epochs, the rewards first increased rapidly, and then the increase slowed. Due to randomly selected parameters, the agent could not adapt to the environment at the initial stage. Only after learning with a large amount of data could the data correlation be captured and the parameters updated. When the learning rate was at a high level of 0.1, the reward obtained by the proposed solution converged to a maximum value of 1500. The reward fluctuation throughout the process was apparent. When the learning rate was 0.001, the reward converged to about 2700, where the system fell into a local optimum, and the training effect could not be improved even by increasing the training epoch. The algorithm's performance with a learning rate of 0.005 was better compared to the other two learning rates. Not only was the highest reward obtained, but the training effect steadily improved with the increase in the number of training epochs. The reward value rose to 3157 when training epochs reached 100.

Fig. 7(b) shows the number of tasks completed by different methods. Baseline-4 only considered link quality, and did not involve model training, hence its results are used as a reference. After five training epochs, the number of completed tasks by the proposed algorithm and baseline-3 surpassed that of baseline-4 and then continued to rise steadily. The number of tasks completed by the proposed scheme was always higher than baseline-3. As seen in Fig. 7(c), the task failure rate of baseline-4 remained at 29%. As a variant of DQN, DDQN reduces data correlation, and thus has better learning and convergence performance. After 100 training epochs, the task failure rates of the proposed method and baseline-1 were about 21% and 25%, respectively. The former was always lower than the latter.

1. www.openits.cn

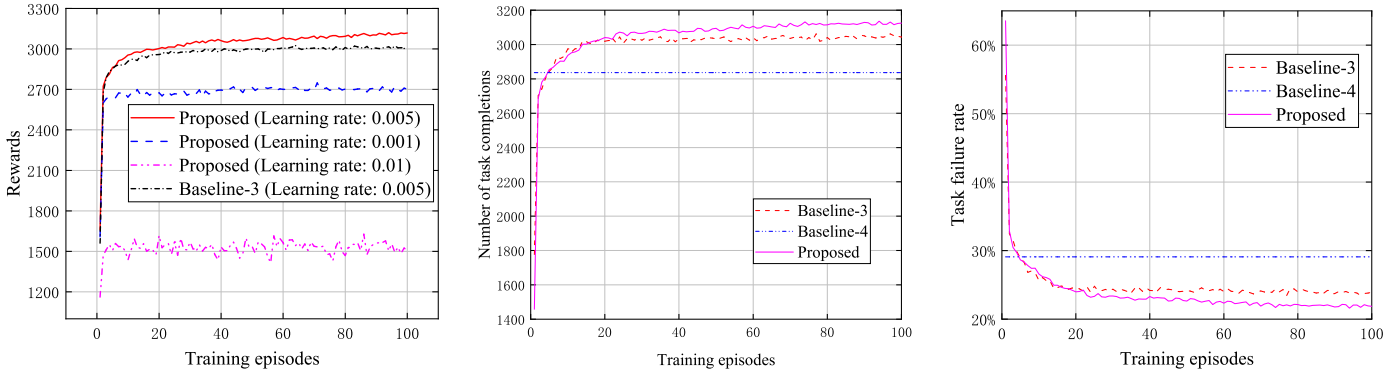
5.2 Impact of Slicing Window Strategy

We verified the performance of the proposed adaptive slicing window strategy. As seen in Fig. 8(a), with the increase of the ratio of delay-sensitive tasks, the task failure rate of baseline-1 and baseline-2 with static slicing window showed an upward trend. Yet, the proposed method and baseline-3 with dynamic window division had stable task failure rates, indicating that the proposed slicing window mode had high adaptability to workload fluctuations. Fig. 8(b) shows the number of slicing windows generated by different methods within two hours. When a new window arrives, the controller will trigger resource reallocation of RAN slices, resulting in a huge signaling overhead. From Figs. 8(a) and (b), the number of windows and the task failure rate of the proposed method were lower than those of baseline-1, suggesting that the proposed method can provide higher-quality services with lower overhead, validating the effectiveness of window adaptation.

We further examined the behavior of different slice window partitioning strategies in response to fluctuations in computing task requests during morning peak hours (6:00 am to 8:00 am). The static strategy divided the timeline evenly into two slicing windows (see Fig. 9(a)). In contrast, the proposed strategy divided the timeline into three slicing windows unevenly (see Fig. 9(c)) by capturing the fluctuations in the number of tasks. Specifically, the slice window's length gradually decreases with an increased task traffic rate. From Figs. 9(b) and (d), the task completion rate was almost the same from 6:00 am to 7:00 am, no matter whether dynamic or static strategies were adopted because the traffic peak had not yet arrived. However, from 7:00 am to 8:00 am, the task completion rate of the proposed strategy remained above 80%, which was higher than that of the static window division strategy. The average task completion rate of baseline-1 with a static policy was only 71%, and that of baseline-5 was even lower than 50%. During the peak hours of traffic flow, the number of windows divided by the proposed scheme may be higher than that of the static strategy. However, throughout the day, the number of slicing windows divided by the former was significantly smaller than that of the latter, confirmed by the results in Fig. 8. The proposed scheme can enhance network management's agility and balance control overhead and QoS from the above results.

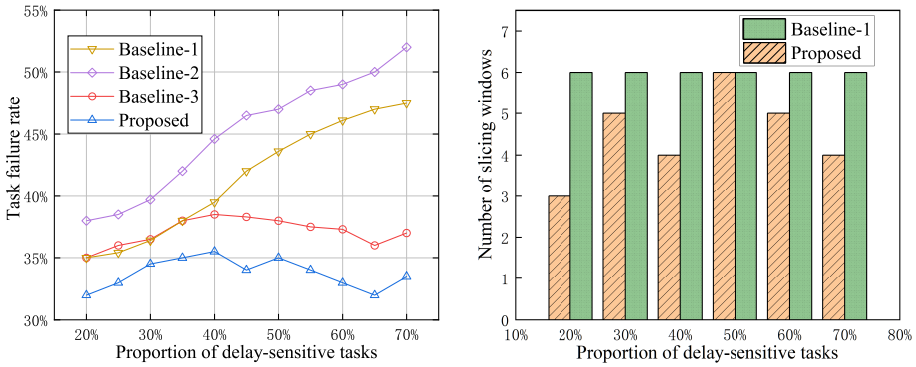
5.3 Impact of Resources and Workloads

Fig. 10(a) shows the effect of spectrum resources on the task failure rate when the number of computing resources is fixed at 15. The task failure rate of each method gradually decreased and stabilized at about 10%. Sufficient spectrum resources enable the controller to have more options, although not the only condition for performance improvement. Next, the effect of computing resources was studied when the number of subchannels was fixed at 20. In Fig. 10(b), the task failure rate dropped rapidly in the initial stage, but when the number of computing resources reached



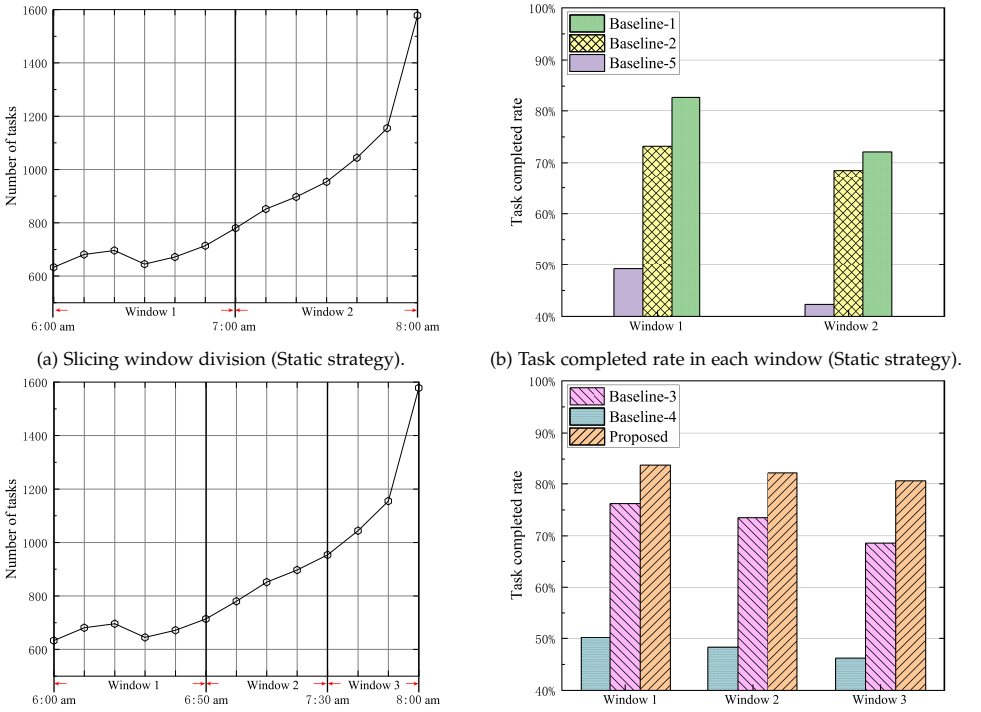
(a) Reward acquisition. (b) Number of tasks completions. (c) Task failure rate.

Fig. 7. Effect of training epochs.



(a) Task failure rate (b) Number of slicing windows

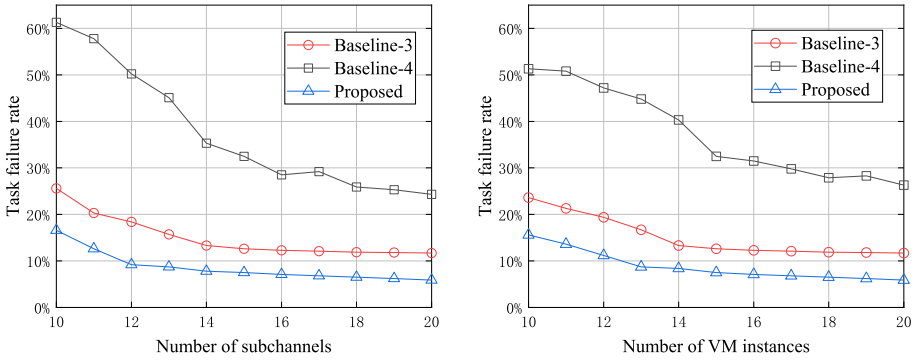
Fig. 8. Impact of delay-sensitive task proportion under different slicing window modes.



(a) Slicing window division (Static strategy). (b) Task completed rate in each window (Static strategy).

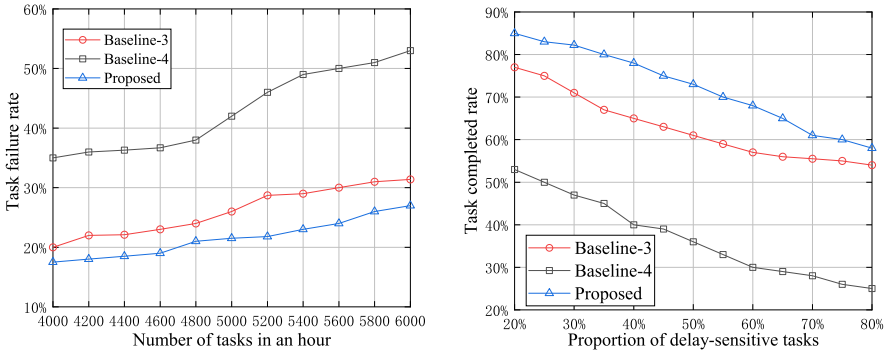
(c) Slicing window division (Proposed strategy). (d) Task completed rate in each window (Proposed strategy).

Fig. 9. Slicing window behavior and its effect on task completion rate.



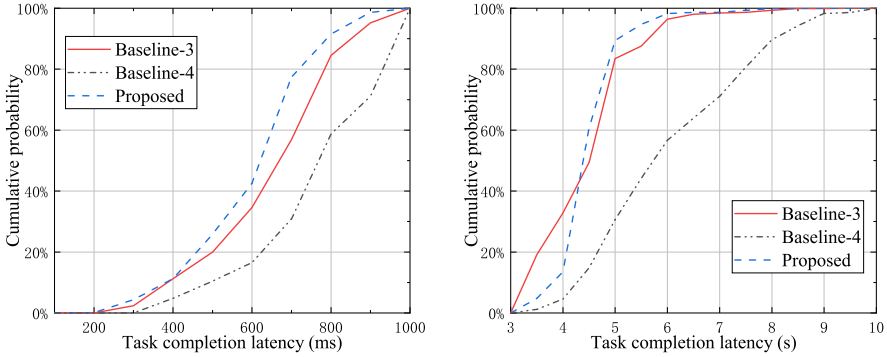
(a) Impact of spectrum resource variation. (b) Impact of computing resource variation.

Fig. 10. Impact of the number of available spectrum and computing resources.



(a) Task failure rate under different workloads. (b) Task failure rate under task proportion.

Fig. 11. Impact of workloads on task completion rate.



(a) Latency-sensitive tasks. (b) Latency-tolerant tasks.

Fig. 12. CDF of task service delay

15, to further increasing the computing resources did not improve performance. At this time, a bottleneck was created by the spectrum resources.

Next, we simulated the situation where the number of released tasks continued to increase over a period of one hour, as shown in Fig. 11(a). Due to resource constraints, the task failure rate generally showed an uptrend. Due to the lack of flexibility, the task failure rate of SINR-based baseline-4 increased from 35% to 52%, while that of DQN-based baseline-2 and baseline-3 increased from 20% to 31%. However, thanks to the collaboration of heterogeneous BSs, the task failure rate of the proposed method increased from 18% to 28% and remained lower than that of the other approaches. Similarly, increases in the proportion of delay-sensitive tasks also decreased the task completion rate. In Fig. 11(b), when the proportion of delay-sensitive tasks was 0.2, the task completion rate of baseline-4 was 52%, and those of baseline-3 and the proposed method were 78% and 85%, respectively. When the proportion of delay-sensitive tasks was 0.8, the task completion rates of the proposed method, baseline-3, and baseline-4 were 58%, 53%, and 26%, respectively. The task scheduling strategy generated by the proposed solution was more reasonable than other methods.

Lastly, we observed the cumulative distribution function (CDF) of service latency distribution for those tasks completed under latency constraints. From Figs. 12(a) and (b), the curves of the proposed scheme are on the left side of the other approaches for both latency-sensitive and latency-

tolerant tasks, which means that the proposed solution can handle more tasks in the same period than other methods.

5.4 Scalability Analysis

We conducted a scalability analysis to demonstrate the efficiency and robustness of the proposed framework.

1) Scalability at the service-type level. The number of slices on each satellite, ground BS, or drone can be increased to support more vehicle services. However, creating more slices will inevitably increase the pressure on resource allocation, and more task types require high service stability and robustness. For the former case, we can refer to the case in Fig. 10, demonstrating that the proposed framework can maintain an acceptable task completion rate when communication and computing resources are insufficient. For the latter case, the results about varying task proportions, as in Figs. 11 and 12, can be used for reference, confirming the efficiency and robustness of the proposed framework.

2) Scalability at the service-coverage level. The considered scenario, managed by an MEC controller, is reproducible. The proposed solution is controller-centric and can be deployed to MEC controllers in multiple areas to extend the service coverage. The number and position of drones in each service area can be adjusted as needed. Using the learning-based approach, each controller tailors a policy that matches the locale's environment. Moreover, the number and location of drones can be adjusted as needed. After modification and adaptation, the proposed scheme can be applied to special scenarios (e.g., drone RANs, satellite-ground networks, and satellite-drone networks).

Besides efficiency and robustness, the proposed framework keeps the control overhead low through the proposed slicing window adaptive mechanism and realizes the dual optimization of QoS and control overhead. This feature is beneficial to support more types of vehicle services and large-scale scenarios.

6 Conclusion

In this paper, we have presented a slicing-based task offloading solution for SAGVNs to support diverse IoV services with differentiated QoS requirements. Unlike traditional RAN slicing approaches, the proposed framework integrates slicing window adaptation, resource slicing, and DDQN-based task scheduling, with the ability to adapt to vehicle task traffic fluctuations without future information. Trace-driven simulation results confirm that the proposed algorithm can effectively increase the number of task competitions, especially in the case of a high proportion of delay-sensitive tasks. Regarding adaptability, the proposed scheme is not constrained by the number of drones and deployment locations. With slicing window adaptation, it can balance the network-wide control overhead and QoS according to task traffic variation. For scalability, the proposed framework can support more vehicle services based on RAN slicing. Algorithms in this framework can be deployed to controllers in different areas to expand service scope. In addition to task offloading, the proposed framework has the potential to support services such as content distribution and data collection. Our ongoing work will develop a federated DRL-based algorithm for large-scale SAGVNs.

Acknowledgment

This research was supported in part by the National Natural Science Foundation of China under Grants 61502230, and 61501224, the Natural Science Foundation of Jiangsu Province under Grant BK20201357, the Six Talent Peaks Project in Jiangsu Province under Grant RJFW-020, and the Postgraduate Research and Practice Innovation Program of Jiangsu Province under Grant KYCX21_1141.

References

[1] W. Zhuang, Q. Ye, F. Lyu, N. Cheng, and J. Ren, "SDN/NFV-empowered future IoV with enhanced communication, computing, and caching," *Proc. IEEE*, vol. 108, no. 2, pp. 274–291, 2020.

[2] R. Meneguette, R. De Grande, J. Ueyama, G. P. R. Filho, and E. Madeira, "Vehicular edge computing: Architecture, resource management, security, and challenges," *ACM Comput. Surv.*, vol. 55, no. 1, pp. 1–46, 2023.

[3] H. Shen, Y. Heng, N. Shi, T. Wang, and G. Bai, "Drone-small-cell-assisted spectrum management for 5G and beyond vehicular networks," in *IEEE ISCC*, 2022, pp. 1–8.

[4] B. Cao, J. Zhang, X. Liu, Z. Sun, W. Cao, R. M. Nowak, and Z. Lv, "Edge–cloud resource scheduling in space–air–ground-integrated networks for Internet of vehicles," *IEEE Internet Things J.*, vol. 9, no. 8, pp. 5765–5772, 2022.

[5] T. Pfandzelter, J. Hasenburg, and D. Bermbach, "Towards a computing platform for the LEO edge," in *ACM EdgeSys*, 2021, pp. 43–48.

[6] B. Mao, F. Tang, Y. Kawamoto, and N. Kato, "Optimizing computation offloading in satellite-UAV-served 6G IoT: A deep learning approach," *IEEE Netw.*, vol. 35, no. 4, pp. 102–108, 2021.

[7] V.-L. Nguyen, R.-H. Hwang, and P.-C. Lin, "Controllable path planning and traffic scheduling for emergency services in the Internet of vehicles," *IEEE Trans. Intell. Transp. Syst.*, 2021.

[8] L. Tao, Y. Watanabe, Y. Li, S. Yamada, and H. Takada, "Collision risk assessment service for connected vehicles: Leveraging vehicular state and motion uncertainties," *IEEE Internet Things J.*, vol. 8, no. 14, pp. 11 548–11 560, 2021.

[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, 2020.

[10] C. Sexton, N. Marchetti, and L. A. DaSilva, "Customization and trade-offs in 5G RAN slicing," *IEEE Commun. Mag.*, vol. 57, no. 4, pp. 116–122, 2019.

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

[12] J. Li, W. Shi, P. Yang, Q. Ye, X. S. Shen, X. Li, and J. Rao, "A hierarchical soft RAN slicing framework for differentiated service provisioning," *IEEE Wireless Commun.*, vol. 27, no. 6, pp. 90–97, 2020.

[13] Q. Ye, W. Zhuang, S. Zhang, A.-L. Jin, X. Shen, and X. Li, "Dynamic radio resource slicing for a two-tier heterogeneous wireless network," *IEEE Trans. Veh. Technol.*, vol. 67, no. 10, pp. 9896–9910, 2018.

[14] H. Peng, Q. Ye, and X. Shen, "Spectrum management for multi-access edge computing in autonomous vehicular networks," *IEEE Trans. Intell. Transp. Syst.*, vol. 21, no. 7, pp. 3001–3012, 2019.

[15] W. Wu, N. Chen, C. Zhou, M. Li, X. Shen, W. Zhuang, and X. Li, "Dynamic RAN slicing for service-oriented vehicular networks via constrained learning," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 7, pp. 2076–2089, 2020.

[16] H. Peng and X. Shen, "Deep reinforcement learning based resource management for multi-access edge computing in vehicular networks," *IEEE Trans. Netw. Sci. Eng.*, vol. 7, no. 4, pp. 2416–2428, 2020.

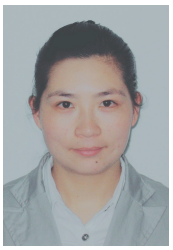
[17] X. Li, H. Zhang, H. Zhou, N. Wang, K. Long, S. Al-Rubaye, and G. K. Karagiannidis, "Multi-agent DRL for resource allocation and cache design in terrestrial-satellite networks," *IEEE Trans. Wireless Commun.*, to be published, DOI:10.1109/TWC.2022.3231379.

[18] M. Chen, Y. Hao, L. Hu, K. Huang, and V. K. Lau, "Green and mobility-aware caching in 5G networks," *IEEE Trans. Wireless Commun.*, vol. 16, no. 12, pp. 8347–8361, 2017.

[19] J. Ji, K. Zhu, D. Niyato, and R. Wang, "Joint cache placement, flight trajectory, and transmission power optimization for multi-UAV assisted wireless networks," *IEEE Trans. Wireless Commun.*, vol. 19, no. 8, pp. 5389–5403, 2020.
[20] X. Sun and N. Ansari, "Jointly optimizing drone-mounted base station placement and user association in heterogeneous networks," in *IEEE ICC*, 2018, pp. 1–6.
[21] E. Karimi, Y. Chen, and B. Akbari, "Task offloading in vehicular edge computing networks via deep reinforcement learning," *Comput. Commun.*, vol. 189, pp. 193–204, 2022.
[22] P. A. Apostolopoulos, G. Fragkos, E. E. Tsiropoulou, and S. Papavassiliou, "Data offloading in UAV-assisted multi-access edge computing systems under resource uncertainty," *IEEE Trans. Mobile Comput.*, vol. 22, no. 1, pp. 175–190, 2023.
[23] C. Kai, H. Zhou, Y. Yi, and W. Huang, "Collaborative cloud-edge-end task offloading in mobile-edge computing networks with limited communication capability," *IEEE Trans. on Cogn. Commun. Netw.*, vol. 7, no. 2, pp. 624–634, 2020.
[24] Z. Bai, Y. Lin, Y. Cao, and W. Wang, "Delay-aware cooperative task offloading for multi-UAV enabled edge-cloud computing," *IEEE Trans. Mobile Comput.*, to be published, DOI:10.1109/TMC.2022.3232375.
[25] M. Li, J. Gao, L. Zhao, and X. Shen, "Deep reinforcement learning for collaborative edge computing in vehicular networks," *IEEE Trans. on Cogn. Commun. Netw.*, vol. 6, no. 4, pp. 1122–1135, 2020.
[26] A. Bozorgchenani, S. Maghsudi, D. Tarchi, and E. Hossain, "Computation offloading in heterogeneous vehicular edge networks: On-line and off-policy bandit solutions," *IEEE Trans. Mobile Comput.*, vol. 21, no. 12, pp. 4233–4248, 2022.
[27] X. Wang, Z. Ning, S. Guo, and L. Wang, "Imitation learning enabled task scheduling for online fting," *IEEE Trans. Mobile Comput.*, vol. 21, no. 2, pp. 598–611, 2022.
[28] P. Dai, K. Hu, X. Wu, H. Xing, and Z. Yu, "Asynchronous deep reinforcement learning for data-driven task offloading in MEC-empowered vehicular networks." in *IEEE INFOCOM*, 2021, pp. 1–10.
[29] V. Erceg, L. J. Greenstein, S. Y. Tjandra, S. R. Parkoff, A. Gupta, B. Kulic, A. A. Julius, and R. Bianchi, "An empirically based path loss model for wireless channels in suburban environments," *IEEE J. Sel. Areas Commun.*, vol. 17, no. 7, pp. 1205–1211, 1999.
[30] S. Zhang, H. Luo, J. Li, W. Shi, and X. Shen, "Hierarchical soft slicing to meet multi-dimensional QoS demand in cache-enabled vehicular networks," *IEEE Trans. Wireless Commun.*, vol. 19, no. 3, pp. 2150–2162, 2020.
[31] Y. Chen, Y. Wang, M. Liu, J. Zhang, and L. Jiao, "Network slicing enabled resource management for service-oriented ultra-reliable and low-latency vehicular networks," *IEEE Trans. Veh. Technol.*, vol. 69, no. 7, pp. 7847–7862, 2020.
[32] Y. Sun, S. Zhou, and J. Xu, "EMM: Energy-aware mobility management for mobile edge computing in ultra dense networks," *IEEE J. Sel. Areas Commun.*, vol. 35, no. 11, pp. 2637–2646, 2017.
[33] D. Chen, Y.-C. Liu, B. Kim, J. Xie, C. S. Hong, and Z. Han, "Edge computing resources reservation in vehicular networks: A meta-learning approach," *IEEE Trans. Veh. Technol.*, vol. 69, no. 5, pp. 5634–5646, 2020.
[34] C. N. Babu and B. E. Reddy, "A moving-average filter based hybrid ARIMA–ANN model for forecasting time series data," *Applied Soft Computing*, vol. 23, pp. 27–38, 2014.
[35] X. Yang, J.-Q. Hu, J. Hu, and Y. Peng, "Asynchronous value iteration for markov decision processes with continuous state spaces," in *IEEE WSC*, 2020, pp. 2856–2866.
[36] Q. Ye, W. Shi, K. Qu, H. He, W. Zhuang, and X. Shen, "Joint RAN slicing and computation offloading for autonomous vehicular networks: A learning-assisted hierarchical approach," *IEEE Open J. Veh. Technol.*, vol. 2, pp. 272–288, 2021.
[37] F. Dai, G. Liu, Q. Mo, W. Xu, and B. Huang, "Task offloading for vehicular edge computing with edge-cloud cooperation," *World Wide Web*, no. 25, pp. 1999–2017, 2022.
[38] Y. Gao, W. Wu, J. Dong, Y. Yin, and P. Si, "Deep reinforcement learning based node pairing scheme in edge-chain for IoT applications," in *IEEE GLOBECOM*, 2020, pp. 1–6.

Hang Shen (Member, IEEE) received the Ph.D. degree (with honors) in Computer Science from the Nanjing University of Science and Technology. He worked as a Full-Time Postdoctoral Fellow with the Broadband Communications Research (BBCR) Lab, Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON, Canada, from 2018 to 2019. He is currently an Associate Professor with the Department of Computer Science and Technology, Nanjing Tech University, Nanjing, China. His research interests involve radio access network slicing, space-air-ground integrated networks, network security, and privacy protection. He is an Associate Editor for the IEEE ACCESS and an Academic Editor of the *Mathematical Problems in Engineering*. He was a Guest Editor for the *Peer-to-Peer Networking and Applications* and a TPC member of the Annual International Conference on Privacy, Security and Trust (PST) 2021. He is a member of the IEEE Computer Society, Communication Society, and Vehicular Technology Society and a member of the ACM.

Yibo Tian received the BS degree in Computer Science from Central South University, Changsha, China. He is currently an MS student at the Department of Computer Science and Technology, Nanjing Tech University, Nanjing, China. His research interests include space-air-ground integrated networks and edge intelligence for vehicular networks.



Tianjing Wang (Member, IEEE) holds a B.Sc. in Mathematics at the Nanjing Normal University in 2000, an M.Sc. in Mathematics at the Nanjing University in 2002, and a Ph.D. in Signal and Information System at the Nanjing University of Posts & Telecommunications in 2009. From 2011 to 2013, she was a Postdoctoral Fellow with the School of Electronic Science and Engineering, Nanjing University of Posts and Telecommunications. From 2013 to 2014, she was a Visiting Scholar with the Department of Electrical and Computer Engineering, State University of New York at Stony Brook. She is now an Associate Professor with the Department of Communication Engineering at Nanjing Tech University. Her research interests include vehicular networks and distributed machine learning.

Guangwei Bai received the B.Eng. and M.Eng. degrees in computer engineering from Xi'an Jiaotong University, Xi'an, China, in 1983 and 1986, respectively, and the Ph.D. degree in Computer Science from the University of Hamburg, Hamburg, Germany, in 1999. From 1999 to 2001, he worked at the German National Research Center for Information Technology, Germany, as a Research Scientist. In 2001, he joined the University of Calgary, Calgary, AB, Canada, as a Research Associate. Since 2005, he has been working at Nanjing Tech University, Nanjing, China, as a Professor in Computer Science. From October to December 2010, he was a Visiting Professor with the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON, Canada. His research interests include architecture and protocol design for communication networks, QoS, multimedia networking, network security, and location-based services. He is a member of the ACM and a Distinguished Member of the CCF.